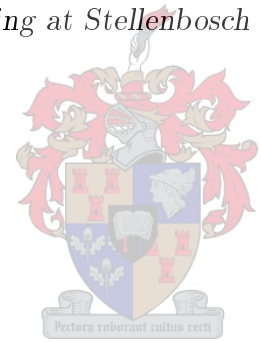
Super-resolution Imaging

by

Stéfan Johann van der Walt

Dissertation presented for the degree Doctor of Philosophy in Engineering at Stellenbosch University



Promotor: Prof. Barend M. Herbst
Applied Mathematics, Mathematical Sciences

December 2010

Declaration

By submitting this dissertation electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the owner of the authorship thereof (unless to the extent explicitly otherwise stated) and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

S.J. van der Walt 23 April 2010

Abstract

Super-resolution Imaging

S.J. van der Walt

Applied Mathematics
Stellenbosch University
Private Bag X1, Matieland 7602,
South Africa

Dissertation: PhDEng

December 2010

Super-resolution imaging is the process whereby several low-resolution photographs of an object are combined to form a single high-resolution estimation. We investigate each component of this process: image acquisition, registration and reconstruction. A new feature detector, based on the discrete pulse transform, is developed. We show how to implement and store the transform efficiently, and how to match the features using a statistical comparison that improves upon correlation under mild geometric transformation. To simplify reconstruction, the imaging model is linearised, whereafter a polygon-based interpolation operator is introduced to model the underlying camera sensor. Finally, a large, sparse, over-determined system of linear equations is solved, using regularisation. The software developed to perform these computations is made available under an open source license, and may be used to verify the results.

Samevatting

Super-resolusie Beeldvorming

S.J. van der Walt

Toegepaste Wiskunde
Universiteit van Stellenbosch
Privaatsak X1, 7602 Matieland, Suid-Afrika

Proefskrif: PhDIng

Desember 2010

In super-resolusie beeldvorming word verskeie lae-resolusie foto's van 'n onderwerp gekombineer in 'n enkele, hoë-resolusie afskatting. Ons ondersoek elke stap van hierdie proses: beeldvorming, -belyning en hoë-resolusie samestelling. 'n Nuwe metode wat staatmaak op die diskrete pulstransform word ontwikkel om belangrike beeldkenmerke te vind. Ons wys hoe om die transform effektief te bereken en hoe om resultate kompak te stoor. Die kenmerke word vergelyk deur middel van 'n statistiese model wat bestand is teen klein lineêre beeldvervormings. Met die oog op 'n vereenvoudigde samestellingsberekening word die beeldvormingsmodel gelineariseer. In die nuwe model word die kamerasensor gemodelleer met behulp van veelhoek-interpolasie. Uiteindelik word 'n groot, yl, oorbepaalde stelsel lineêre vergelykings opgelos met behulp van regularisering. Die sagteware wat vir hierdie berekeninge ontwikkel is, is beskikbaar onderhewig aan 'n oopbron-lisensie en kan gebruik word om die gegewe resultate te verifieer.

To my father, who opened so many doors for me,
and to my mother, who taught me to cross their thresholds with a smile.

Acknowledgements

As with many a large undertaking, doctoral research affects the candidate's life in ways unanticipated beforehand. I'd like to thank those friends, colleagues and family members who supported me during this fascinating (and sometimes daunting!) journey. At the risk of neglecting some, I'd like to mention a few individuals and institutions by name:

- My study advisor, Prof. Ben Herbst, who allowed me the freedom to explore and almost certainly had more faith in me than I deserved.
- DPSS (Defence Peace, Safety & Security, a branch of the CSIR), the National Research Foundation and Stellenbosch University who supported me financially. Additional funding for overseas travel was provided by Enthought, Inc., the Python Software Foundation and the Helen Wills Neuroscience Institute of the University of California at Berkeley.
- Francois Malan, whose insights during discussions on the subtler aspects of digital photography proved invaluable.
- Michèle—without your encouragement and love, this text may never have been written.

Data Sources

- Douglas Kerr, who gratiously allowed me to use a figure from his paper entitled " *The Proper Pivot Point for Panoramic Photography*".
- The "library" data sequence used in Chapter 7 is by Barbara Levienaise-Obadia, University of Surrey. Tomas Pajdla and Daniel Martinec, CMP, Prague provided the "text" dataset. Both sets were collected by David Capel and are available for download from `http://www.robots.ox.ac.uk/~vgg/data`.
- Chelsea the Cat, who proved decidedly difficult to photograph, makes her appearance in Chapters 4 and 7.
- An illustration of the steps taken during gradient-based optimisation (see Chapter 7) was made by Oleg Alexandrov, who kindly released his work into the public domain.

Contents

Abstract iii

Samevatting iv

Contents vii

1 Introduction 1

2 Image acquisition 8
- 2.1 Introduction . . . 8
- 2.2 Digital photography . . . 9
  - 2.2.1 Lens distortions . . . 9
  - 2.2.2 Effects due to motion . . . 9
  - 2.2.3 Sensor layout . . . 13
  - 2.2.4 Diffraction limited photography . . . 14
  - 2.2.5 Noise processes . . . 14

3 The Discrete Pulse Transform 20
- 3.1 Introduction . . . 20
- 3.2 Background theory . . . 21
- 3.3 Implementation . . . 22
  - 3.3.1 Pulse representation and manipulation . . . 23
  - 3.3.2 Algorithm for computing the 2D Discrete Pulse Transform 26
  - 3.3.3 Benchmark . . . 29

4 Feature detection and matching 34
- 4.1 Introduction . . . 34
- 4.2 Feature detection using the discrete pulse transform . . . 35
  - 4.2.1 Scale space . . . 35
  - 4.2.2 Pulse strength . . . 35
- 4.3 Matching and correspondence . . . 39
  - 4.3.1 Statistical feature comparison . . . 39
  - 4.3.2 Other matching methods . . . 42
- 4.4 Conclusion . . . 45

5 Accurate image registration **47**
5.1 Introduction . . . 47
5.2 The transformation matrix . . . 48
5.3 Sparse registration: Estimating a homography from correspondences . . . 50
5.3.1 Least-squares estimation . . . 50
5.3.2 Estimation in the presence of outliers . . . 54
5.4 Dense registration methods . . . 58
5.4.1 Error minimisation . . . 58
5.4.2 Pyramidal methods . . . 60
5.4.3 Mutual information . . . 61
5.4.4 Log-polar registration . . . 62
5.5 Photometric registration . . . 64

6 Super-resolution image processing **73**
6.1 Introduction . . . 73
6.2 The dawn of super-resolution . . . 74
6.2.1 Why is super-resolution possible? . . . 76
6.2.2 Maximum a posteriori estimate . . . 76
6.2.3 Pixel probabilities . . . 77
6.2.4 Camera parameters . . . 78
6.2.5 Boundary effects . . . 79
6.2.6 The point-spread function . . . 79
6.2.7 Solving the linear system . . . 80
6.3 Other approaches . . . 80
6.3.1 Averaging . . . 80
6.3.2 Map and deblur . . . 81
6.3.3 Pan-sharpening . . . 82
6.3.4 Compressive sampling / compressed sensing . . . 82

7 Super-resolution as a sparse linear problem **85**
7.1 Introduction . . . 85
7.2 The camera matrix, A . . . 86
7.3 Linear interpolation operators . . . 89
7.3.1 Bilinear interpolation . . . 89
7.3.2 Polygon-based interpolation . . . 92
7.4 Solving the large, sparse least-squares problem $A\mathbf{x} = \mathbf{b}$ . . . 98
7.4.1 Iterative optimisation methods . . . 100
7.4.2 Iterative-interpolation super-resolution . . . 103
7.5 Structural metrics . . . 103

7.6 Sensitivity to photometric registration 105
7.7 Recursive implementation 105
7.8 Results 105

8 Conclusion **113**

A Data-set format **A-1**

B Software API **B-1**

List of Symbols

Vectors and Matrices:

$\mathbf{x}$ Column vector (lower-case, bold)

$\mathbf{x}^T$ Row vector

A Matrix (upper-case, plain)

I Identity matrix

Gradient and derivatives:

$\frac{\partial f}{\partial x}$ Partial derivative of $f(x, y)$ with respect to x.

∇ Gradient

$$\nabla f = \frac{df(\mathbf{x})}{d\mathbf{x}} = \left[\begin{array}{cccc} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_n} \end{array} \right]^T$$

Sets:

card(V) Cardinality of (number of elements in) the set V.

$A \bigcup B$ Union of the sets A and B.

$A \bigcap B$ Intersection of the sets A and B.

Chapter 1

Introduction

> *"Unlike many other branches of science, students of digital image warping benefit from the direct visual realization of mathematical abstractions and concepts. As a result, readers are fortunate to have images clarify what mathematical notation sometimes obscures. This makes the study of digital image warping a truly fascinating and enjoyable endeavor."*
>
> — George Wolberg [Wol90] in a quote equally valid for computer vision and digital image processing.

Satellites cannot be re-launched and, as far as we are aware, time travel is not possible[1]. These constraints indicate two situations in which super-resolution imaging could be applied. For example, a satellite designed to monitor crop growth may later be retargetted for military observation. For the latter purpose, the resolution (or sampling interval) of recorded signals, such as photographs or videos, need to be much higher. In most cases, it would not be cost-effective to refurbish the satellite camera with a new lens (although this is sometimes done, as with NASA's Hubble service missions). Another example is security footage, typically taken with a low-cost camera. A criminal act is committed, but the face of the perpetrator cannot be distinguished due to the low resolution of the footage. In both these situations, a signal that would otherwise be of little use may yield important information after applying image super-resolution.

The Reginald Denny case was the first time that super-resolution techniques were used in a United States courtroom [Mor97]. Denny, a truck driver, was nearly beaten to death during the 1992 Los Angeles riots, but his assailants were captured on video by a news helicopter. To prove the identity of one of the mobsters, super-resolution-like techniques were applied to a video segment showing a rose tattoo on his arm [NYN93]. In the end, this evidence was not enough for a conviction, but it paved the way for similar image processing technology in the American courtroom.

[1] This statement may be refuted by future time-travellers.

No matter how high the resolution of a camera system, there often exists the need to improve it. This was the driving force for research done at NASA Ames that led to the first mathematical super-resolution formulation [CKK+93]. Making the best of the existing situation, they applied their techniques to photos taken during the Pathfinder mission, the result of which is shown at the beginning of Chapter 6 in Figure 6.1.

We should point out that "resolution" is a nebulous concept. It is easy to increase the resolution of an image by simply scaling (resizing) it, possibly using an interpolation method such as Lanczos or cubic interpolation. While this increases the number of pixels, it does not add any *information* to the image. In this work, we think of improved resolution as an increase in high-frequency information—the detail that provides definition on a small scale. We aim to make blurred text legible, or to recognise faces that were previously unrecogniseable.

We can therefore describe image super-resolution as a class of algorithms that combine several low-resolution (LR) images into a single high-resolution (HR) image of improved detail. If all low-resolution images are identical, no improvement is possible, but in practice even images taken of a static object from the same camera position contain differing information due to signal noise. For example, a common technique used by amateur astronomers is to photograph a section of the sky with an inexpensive CCD-camera mounted on a tracking telescope. Any one of the resulting images is of little value on its own, but by adding them together the contribution of the signal is raised with respect to zero-mean noise; stars and galaxies appear as if by magic.

To perform super-resolution imaging, we need to model the entire photographic process: from the time the light reflects off a surface until digital data arrives on our computers. This process encompasses a large number of distortions, noise processes and other non-linear transformations—too many to model realistically. We therefore make numerous assumptions and simplifications to find a suitable linear model that leads to viable computation. Our model is expressed simply as the (overdetermined) set of linear equations,

$$A\mathbf{x} = \mathbf{b},$$

where $\mathbf{x}$ is a representation of the scene in high-resolution (this is what we would like to estimate), $\mathbf{b}$ are our low resolution images and A is a matrix that encompasses the entire photographic process.

Almost all our effort is spent finding the matrix A that models the relation of the available low-resolution photographs to the desired high-resolution scene photograph all the way down to individual pixel level. Thereafter, finding $\mathbf{x}$ is

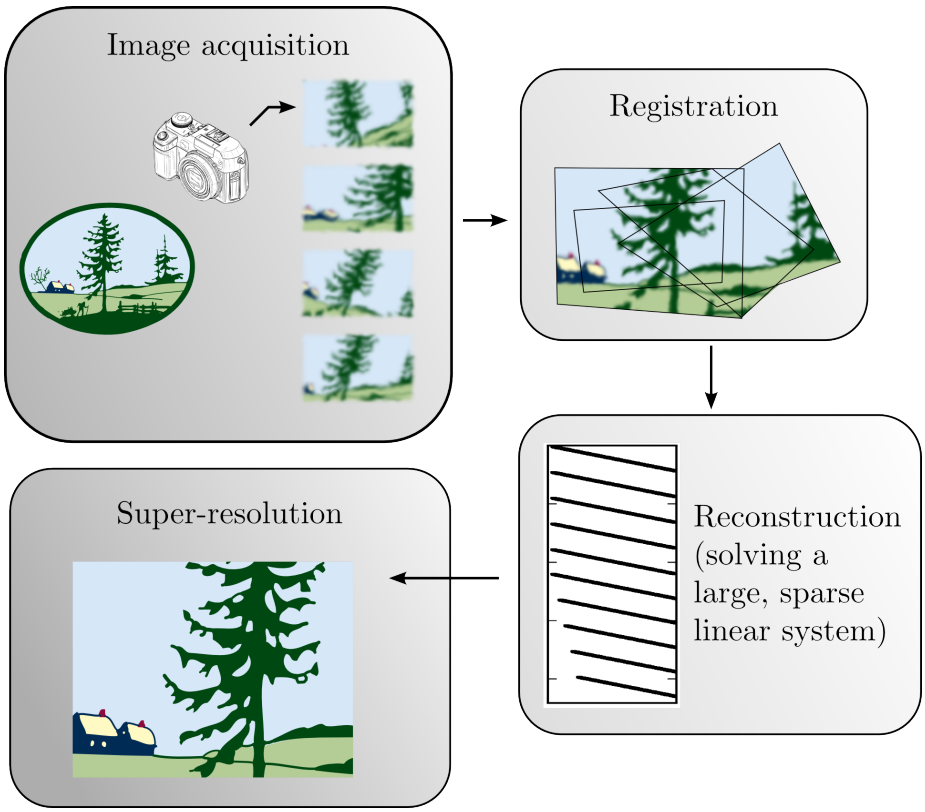


Figure 1.1: *An overview of the super-resolution process.*

a matter of solving a large, sparse linear system.

The results shown in Chapter 7 provide convincing evidence that super-resolution does work—maybe not as well as in the popular television series Crime Scene Investigation—but well, nonetheless.

Structure of this document

Super-resolution imaging is not a single operation, but rather a combined sequence of algorithms, each of which is discussed in a separate chapter of this dissertation (see Figure 1.1). Each chapter opens with a concise introduction to the topic, accompanied by a list of references to existing literature. Due to the generic nature of these subjects, a comprehensive list is not feasible; instead, we attempt to collect a representative sub-set of influential papers in each field.

An important complement to this document is the accompanying software package. Developed under a free, open source license, this Python and C based library implements all the fundamental algorithms required to continue further research on super-resolution. Snippets of code are given throughout the text to

illustrate its use. Functions are documented more completely in Appendix B-1.

Chapter 2 discusses image acquisition, the process of capturing digital data from photographic hardware such as cameras or scan-sensors. We introduce the different degradations experienced during digital image formation, in anticipation of an imaging model required for reconstruction. Suitable camera configurations are discussed, and noise suppression techniques are suggested.

Chapter 3 gives an overview of the Discrete Pulse Transform (DPT), as well as its efficient implementation in two dimensions, in preparation for the identification of features in the next chapters.

Chapters 4 and 5 go hand in hand, and describe how to align two images where one has undergone a geometrical transformation. Chapter 5 discusses two popular methods: direct and feature based registration. For the latter purpose, Chapter 4 develops a feature detector based on the Discrete Pulse Transform of Chapter 3. Chapter 5 also treats photometric correction: adjusting the histogram of two images to be more similar.

Once all images are accurately aligned, Chapter 6 constructs the super-resolution problem, based on a model of the image acquisition pipeline from Chapter 2. It also lists a number of fast heuristic super-resolution methods. Chapter 7 shows how to set up the super-resolution problem as an over-determined set of linear equations, and how to solve the least squares problem using iterative sparse methods.

The appendices, starting on page A-1, treat topics that are deemed ancillary to the main discussion.

Contributions

The basic ideas in super-resolution are well established; for example, the framework provided by Cheeseman *et al.* in 1993 [CKK+93] is still used today. Close investigation of seemingly different approaches such as [Ban09] often reveals similar strategies, and, unless a revolutionary new approach is discovered, improvement lies in the detail of existing algorithms. We therefore carefully investigate each component of the existing super-resolution framework, and make improvents where possible. These contributions are listed below, and are further detailed in corresponding chapters of the dissertation.

Acquisition

Pre-processing is avoided in the literature due to its tendency to remove or destroy the information required to perform super-resolution reconstruction. We show how certain noise-removal techniques, popularised in astronomical and

forensic signal processing, can be used without adversely affecting reconstruction.

Registration

For feature-based registration, we introduce a new feature detector, based on a two-dimensional extension of the discrete pulse transform. Feature correspondences are found using a statistical matching algorithm.

Reconstruction

To obtain a super-resolution estimate, a large, sparse system of linear equations, $A\mathbf{x} = \mathbf{b}$, has to be solved. The solution is obtained using an iterative method such as LSQR, while forcing the result to be close to a prior estimate. It is posed that the camera process can be modelled simply using a linear interpolation operator. The bilinear interpolator causes oversmoothing and the appearance of artefacts, therefore we introduce a new linear interpolator, based on polygon geometry, to ameliorate these problems.

Software

> *"For the computational scientist, code is an embodiment of science, a way to test and use ideas to make predictions, and a way to gain insight via obtained data."* — D.E. Stevenson [Ste99]

A super-resolved image is the result of a series of computations done by a complex piece of software. Usually, we would not trust experimental outcomes without examining the methodology followed in accompanying papers and there is no reason why software should be treated any differently [HR94].

A free and open source Python-based software framework was written to obtain the results shown in this dissertation. We hope this will encourage others to inspect our methods so that they may improve upon them. We welcome the use of our code in research and teaching, and appreciate contributions in the form of suggestions, bug reports or fixes. We include algorithms for:

- Accurate image alignment via feature matching, dense registration or the log polar transform.
- Feature detection using the Discrete Pulse Transform.
- Super resolution using a selection of operators, norms, optimisation methods and other adjustments.
- Wavelet image denoising.

- Miscellaneous tasks, such as image warping, RANSAC, polygon clipping, chirp z-transformation, phase correlation, normalised cross-correlation, summed area tables, joint histogram and mutual information computations.

Software

Super Resolution Software (`http://mentat.za.net/supreme`**)**
Throughout the text, boxes like these appear with instructions on how to use the accompanying software. Please ensure that the following dependencies are satisfied:

Python (`http://www.python.org`)

A free and open-source, general purpose programming language popular in scientific computing.

NumPy (`http://numpy.scipy.org`)

An N-dimensional array package.

SciPy (`http://scipy.org`)

Scientific computing tools.

IPython (`http://ipython.org`)

An enhanced interactive shell.

Matplotlib (`http://matplotlib.sf.net`)

A package for 2- and 3-dimensional plotting.

These packages are available under three prominent platforms (Linux, Mac OS X and Windows). They are pre-packaged under almost all Linux distributions. For Windows a single-executable installer can be obtained from either `http://pythonxy.com` or `http://code.enthought.com`. At the same URL, Enthought provides a DMG for Mac OS X.

Bibliography

[Ban09] V. Bannore. *Iterative-Interpolation Super-Resolution Image Reconstruction: A Computationally Efficient Technique*. Springer, 2009.

[CKK+93] P. Cheeseman, B. Kanefsky, R. Kraft, J. Stutz, and

R. Hanson. Super-resolved surface reconstruction from multiple images. In G. R. Heidbreder, editor, *Proceedings of the Thirteenth International Workshop on Maximum Entropy and Bayesian Methods*. Kluwer Academic, 1993.

[HR94] L. Hatton and A. Roberts. How accurate is scientific software? *IEEE Transactions on Software Engineering*, 20(10):785–797, 1994.

[Mor97] L.C. Morrison. Computerized image processing in the Reginald Denny beating trial. In L.I. Rudin and S.K. Bramble, editors, *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 39–50, 1997.

[NYN93] Prosecution rests its case in Los Angeles riot trial. Saratosa Herald-Tribune, September 1993.

[Ste99] D.E. Stevenson. A critical look at quality in large-scale simulations. *Computing in Science & Engineering*, 1(3):53–63, 1999.

[Wol90] George Wolberg. *Digital Image Warping*. IEEE Computer Society Press, Los Alamos, 1990.

Chapter 2

Image acquisition

> *"Everyone is likely to be familiar with the concept of image resolution, but unfortunately, too much emphasis is often placed on this single metric. Resolution only describes how much detail a lens is capable of capturing – and not necessarily the quality of the detail that is captured. Other factors therefore often contribute much more to our perception of the quality and sharpness of a digital image."*
>
> *— Sean McHugh, Cambridge in Colour,* `http://cambridgeincolour.com`.

2.1 Introduction

The path a photon follows from where it is reflected off an object to where it finally reaches the camera sensor is influenced by many factors. The initial direction of scattering is determined by the object surface, whereafter temperature differences and particles in the atmosphere influence it further. When the camera is reached, several lense elements have to be traversed, and at the aperture variation due to diffraction may be witnessed. Once at the sensor the photon excites a charge which may be offset by noise from the electronics.

The description above is hardly a comprehensive overview of all factors involved, but it emphasises the complexity of image formation. In order to perform super-resolution image reconstruction, we either need to model these effects or choose to ignore them. The following chapter mentions a few prominent effects, and motivates our choice for ignoring most of them in our imaging model.

For a detailed overview of image formation, we refer to Sidney F. Ray's comprehensive "Applied Photographic Optics" [Ray02].

2.2 Digital photography

2.2.1 Lens distortions

Radial distortion

The camera lens may introduce a number of distortions, the most common of which is radial (or barrel) distortion. Radial distortion occurs when a lens is not symmetrically designed with regard to the aperture stop. Figure 2.1 shows radial distortion in a real photograph.

Radial distortion can be measured and removed before reconstruction, but this destroys some super-resolution information. Instead, the parameters may be included in the transformation model [CJAE05]. We choose to neglect radial distortion, since super-resolution is almost always executed on a small region inside a much larger image, where the distortion is negligible.

Vignetting

The effect of reduced peripheral image illumination is known as vignetting. Vignetting occurs naturally (see [Ray02, p. 131], the "$\cos^4\theta$ law of illumination"), due to lens design or due to part of the camera extending into the field of view. It is not hard to measure and correct for vignetting, but in our data-sets most objects of interest are in the centre of the frame.

Chromatic aberration

The refractive indices of optical objects vary according to wavelength. In different colour band objects may therefore appear at slightly differing positions, as seen in Figure 2.2. In our experiments, we assume that chromatic aberration is not severe.

2.2.2 Effects due to motion

Geometric transformation

Rotating or translating a camera causes similar changes in the resulting photographs. As part of our model, we need to estimate these changes so that pixels in different input photos may be related to one another (the topic of Chapter 5). This estimation process is much simplified if we assume that the transformation is linear, a valid assumption if the object lies on a plane or far away from the camera. The need for super-resolution often arises in these circumstances, when photographs of a far-away object is taken, yielding only low-resolution representations.

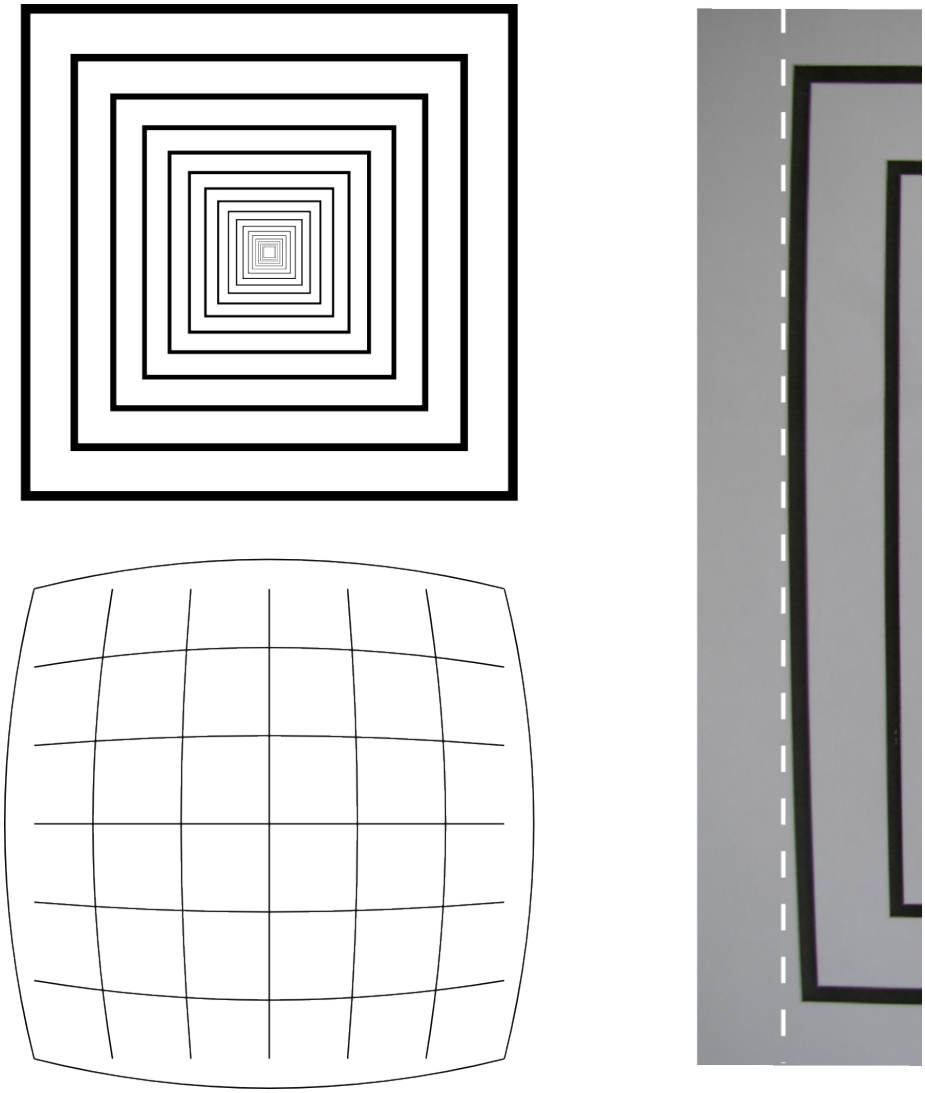


Figure 2.1: *Barrel distortion. A test pattern (top left) was photographed; a small section of the photo is shown on the right. On the photo, the black line is curved, as seen when comparing to the superimposed dashed white line. Radial (or barrel) distortion, shown on the diagram at the bottom left, is witnessed.*

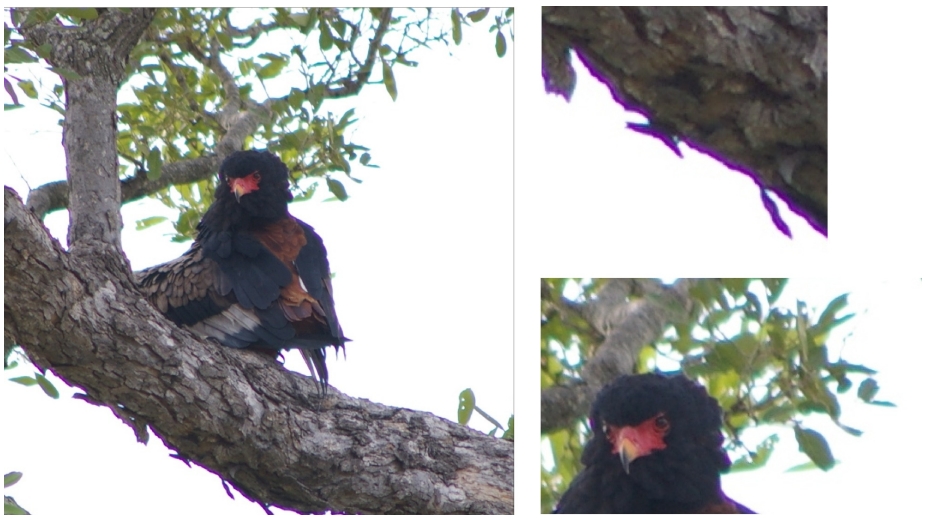


Figure 2.2: *Chromatic aberration, prominently visible as coloured bands around edges.*

Parallax

When the target object is close, movement of the camera may introduce parallax. By carefully rotating the camera around its pivot point (not its front or rear nodal point), parallax can be avoided as illustrated in Figure 2.3. Figure 2.4 shows a practical example. If parallax is present, two images are no longer related by a linear transformation, and a more complex motion model is required. In our experiments, it is assumed that objects are either far away from the camera, or that the camera was rotated around its pivot point so that no parallax occurs.

Motion blur

If either the camera or the target object moves while the shutter is open, a single scene point illuminates multiple sensor pixels. This causes a smearing effect, known as motion blur. Techniques such as deconvolution, more suited to this problem than super-resolution, are often used to remove this effect.

Lighting changes

Movement of the target object, camera or light sources may introduce varying object lighting. To address this problem requires finding the positions of light sources and the structure of an object based on lighting—both ill-posed problems. We therefore assume that lighting remains constant across all input frames; a reasonable assumption given that frames are often taken in quick succession (e.g., in video footage).

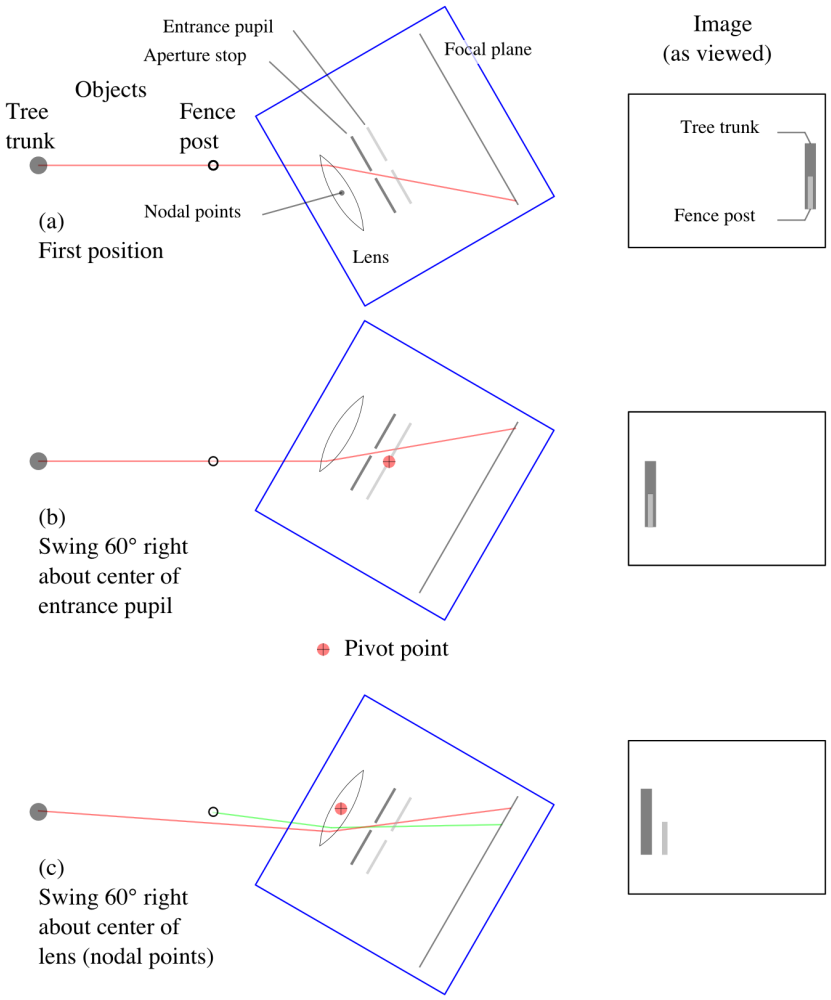


Figure 2.3: *"Rotation of camera with 'behind the lens' aperture stop" reproduced from* The Proper Pivot Point for Panoramic Photography *[Ker08] by Douglas A. Kerr with permission of the author.*

(a) Rotating the camera around any point other than its pivot point introduces parallax. Note how the bunny's hand disappears behind the bus.

(b) The camera is rotated around a point nearby the pivot point. Parallax is noticably reduced.

Figure 2.4: *Rotating a camera may introduce parallax. Here we show what happens when the camera is rotated around an arbitrary point (top) or the pivot point (bottom). The pivot point was estimated by eye, so some parallax may still be present in (b).*

2.2.3 Sensor layout

In colour imaging there are three predominant ways of separating light into red, green and blue. The first is to use a beam splitter to redirect incoming light through three separate filters. These days, a more integrated approach is followed whereby the light is filtered once it arrives at the imaging sensor. This leads to the second approach, patented in the 1970s by B.E. Bayer of the Eastman Kodak Company (see Figure 2.5), of placing a mask over the imaging sensor (Figure 2.6) so that alternating pixel elements capture different colours. The resulting image has slightly lower resolution, since the different colour values must be combined ("demosaicked") to form a full-colour image. The third approach, used in Foveon's X3 sensor, makes use of silicon's color absorption properties to read red, green and blue values from a single pixel element. Data from an X3 sensor can be handled as three separate monochrome images, unlike the values obtained from a Bayer filtered sensor.

While this dissertation only treats monochrome images, we aim to model the Bayer demosaicking process as part of future research into colour super-resolution.

2.2.4 Diffraction limited photography

Even with a perfect lens, the resolution of a photograph is limited by diffraction. Diffraction occurs whenever a wave encounters an obstacle; for example when light is forced through a small opening, such as a camera's aperture. Instead of travelling in a straight line, the light spreads out resulting in the interference pattern known as an Airy disc. For very small apertures, the width of the Airy disc is wide relative to individual sensor pixels; the resulting overlap causes a loss of contrast. The impulse response of an imaging system, known as its Point Spread Function (PSF), is an Airy disc if diffraction is the only limitation. The Gaussian function approximates the Airy disc well, and is used as the camera PSF in our imaging model.

2.2.5 Noise processes

> *The operation of a CCD is often compared to measuring the spatial distribution of rainfall over a field by placing an array of buckets on the field. Following a storm, the buckets are systematically transferred by conveyor belts to a metering station where the amount of water in each bucket is measured. Each measurement then represents the amount of rainfall at a particular location on the field.*
> — G.E. Healey and Raghava Kondepundy [HK94], paraphrasing J. Kristian and M. Blouke [KB82], memorably describe the functioning of a CCD shift register.

Imaging noise depends on the underlying sensor technology. This discussion treats the noise characteristics of CCD (Charge-coupled device) sensors, which are commonly found in consumer digital cameras and are used extensively for astro-photography. The same reasoning, but with different specifics, may be applied to CMOS (Complementary metal-oxide-semiconductor) sensors.

In [FM06] it is shown that the prevalence of noise sources change for different exposure levels. At low intensities, readout noise (also known as amplifier noise) is prominent, overshadowed at medium intensities by shot noise, in itself a combination of photon and dark noise. At high exposure levels, fixed pattern noise due to slight variations in pixel geometries and sensitivities is most relevant [Jan01].

United States Patent [19]
Bayer

[11] **3,971,065**
[45] **July 20, 1976**

[54] **COLOR IMAGING ARRAY**

[75] Inventor: **Bryce E. Bayer**, Rochester, N.Y.

[73] Assignee: **Eastman Kodak Company**, Rochester, N.Y.

[22] Filed: **Mar. 5, 1975**

[21] Appl. No.: **555,477**

[52] **U.S. Cl.** ..... **358/41**; 350/162 SF; 350/317; 358/44
[51] **Int. Cl.²** **H04N 9/24**
[58] **Field of Search** ..... 358/44, 45, 46, 47, 358/48; 350/317, 162 SF; 315/169 TV

[56] **References Cited**
UNITED STATES PATENTS

2,446,791	8/1948	Schroeder	358/44
2,508,267	5/1950	Kasperowicz	358/44
2,884,483	4/1959	Ehrenhaft et al.	358/44
3,725,572	4/1973	Kurokawa et al.	358/46

Primary Examiner—George H. Libman
Attorney, Agent, or Firm—George E. Grosser

[57] **ABSTRACT**

A sensing array for color imaging includes individual luminance- and chrominance-sensitive elements that are so intermixed that each type of element (i.e., according to sensitivity characteristics) occurs in a repeated pattern with luminance elements dominating the array. Preferably, luminance elements occur at every other element position to provide a relatively high frequency sampling pattern which is uniform in two perpendicular directions (e.g., horizontal and vertical). The chrominance patterns are interlaid therewith and fill the remaining element positions to provide relatively lower frequencies of sampling.

In a presently preferred implementation, a mosaic of selectively transmissive filters is superposed in registration with a solid state imaging array having a broad range of light sensitivity, the distribution of filter types in the mosaic being in accordance with the above-described patterns.

11 Claims, 10 Drawing Figures

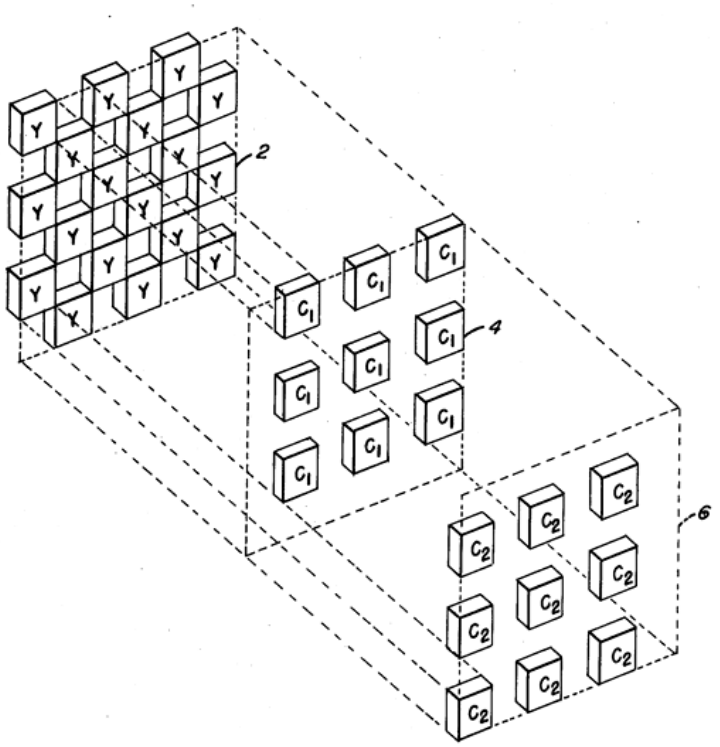


Figure 2.5: *Front page of B.E. Bayer's US patent.*

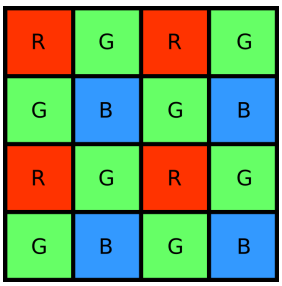


Figure 2.6: *Pixel colouring in the Bayer pattern.*

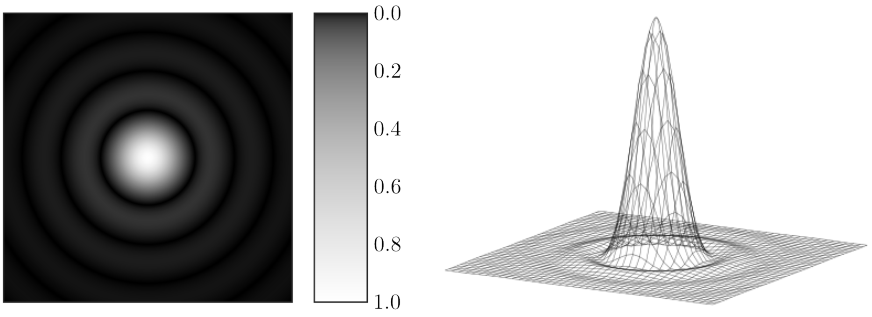


Figure 2.7: *The Airy disc (2D and 3D representations) which shows the intensity distribution of light that travelled through a circular aperture.*

Readout noise is modelled as white noise, while photon and dark noise are Poisson-distributed—often approximated as Gaussian in the literature. Fixed pattern noise is a gain factor that differs for each pixel element.

Noise removal

Super-resolution estimation depends on the combination of input frames to provide missing high-frequency content, destroyed due to aliasing. Slight shifts in camera position yield small, localised changes which we exploit to do the reconstruction. Notably, any noise removal process that combines neighbouring pixels destroys this information. The process combines pixels from different frames, corresponding to the same scene position, a process in some ways similar to averaging. As such, noise from zero-mean processes often do not disturb the reconstruction significantly.

Of the above, the only process modelled as having a non-zero mean is fixed pattern noise [HK94], which can be detected and removed, as described below. Note that care has to be taken with noise removal: it is easy to distort other zero-mean noise sources to have non-zero means. This might be the reason why noise removal is not applied in super-resolution literature.

Flat-field correction

The technique described here for removing fixed pattern noise, called flat-field correction, is commonplace in astro-photography—so commonplace, in fact, that we were unable to trace the first paper on the topic. It is not as well known in images restoration, so we elaborate below, following the description and notation from [Fri09] (note their use of bold capital letters to indicate vectors).

Neglecting noise, the camera response function is often modelled as

$$\mathbf{I}_{opt} = (g\mathbf{Y})^{\gamma}$$

where $\mathbf{Y}$ (whose elements lie between 0 and 1) is the light incident on the sensor, $\mathbf{I}_{opt}$ is the ideal output image, g is a channel gain (different for red, green and blue) and γ is a gamma correction factor. The gamma correction factor models the sensors' tendency to compress high intensity values and to expand low intensities. Adding noise, separating the zero-average fixed pattern noise gain, $\mathbf{K}$, from all other sensor noise sources, $\mathbf{\Omega}$, yields

$$\mathbf{I} = g^{\gamma}\left[(\mathbf{1}+\mathbf{K})\mathbf{Y}+\mathbf{\Omega}\right]^{\gamma}.$$

All multiplications are point-wise. An additional term, $\mathbf{Q}$, is added to model process noise such as quantisation or JPEG compression:

$$\mathbf{I} = g^{\gamma}\left[(\mathbf{1}+\mathbf{K})\mathbf{Y}+\mathbf{\Omega}\right]^{\gamma}+\mathbf{Q}.$$

Using the Taylor series expansion of $(1+x)^{\gamma} = 1+\gamma x+\mathcal{O}\left(x^2\right)$ at $x=0$, the above becomes

$$\begin{aligned}
\mathbf{I} &= (g\mathbf{Y})^{\gamma}\left[\mathbf{1}+\mathbf{K}+\mathbf{\Omega}/\mathbf{Y}\right]^{\gamma}+\mathbf{Q} \\
&\approx (g\mathbf{Y})^{\gamma}\left[\mathbf{1}+\gamma\mathbf{K}+\gamma\mathbf{\Omega}/\mathbf{Y}\right]+\mathbf{Q} \\
&= \mathbf{I}_{opt}+\mathbf{I}_{opt}\gamma\mathbf{K}+\mathbf{\Theta} \\
&= \mathbf{I}_{opt}+\mathbf{I}_{opt}\mathbf{N}+\mathbf{\Theta} \\
&= (\mathbf{1}+\mathbf{N})\mathbf{I}_{opt}+\mathbf{\Theta}
\end{aligned}$$

where $\mathbf{N}=\gamma\mathbf{K}$, and $\mathbf{\Theta}$ represents the modelling noise, $\gamma\mathbf{I}_{opt}\mathbf{\Omega}/\mathbf{Y}+\mathbf{Q}$. Given d different images, $\mathbf{I}_d$, we intuitively estimate $\mathbf{N}$ as

$$\hat{\mathbf{N}} = \sum_{d}\frac{\left(\mathbf{I}_d-\hat{\mathbf{I}}_{d,opt}\right)}{\hat{\mathbf{I}}_{d,opt}}$$

where $\hat{\mathbf{I}}_{d,opt}$ is an estimation of the ideal, noise-free version of $\mathbf{I}_d$. This assumes that $\mathbf{\Theta}$ is independent of $\mathbf{I}_{opt}$ and distributed normally with zero-mean. Of course, $\mathbf{\Theta}$ is not independent of $\mathbf{I}_{opt}$, but since it is so much smaller in amplitude the assumption has no noticable impact. The corrected image is

$$\mathbf{I}_{corrected} = \frac{\mathbf{I}}{\mathbf{1} + \hat{\mathbf{N}}}.$$

It can be shown that the variance of $\mathbf{N}$ is proportional to the variance of the input image, and is inversely proportional to its amplitude squared [Fri09]. This observation suggests that $\mathbf{N}$ is best estimated from a smooth photo of a bright object (although the sensor must not saturate). A bright area of the sky with the camera set out of focus works well.

Software

An estimate of $\hat{\mathbf{I}}_{d,opt}$, the ideal, noise-free version of $\mathbf{I}_d$, is obtained by applying a denoising filter to $\mathbf{I}_d$. The wavelet filter suggested in [Fri09] and described in [MKRM99] is implemented as `supreme.noise.dwt_denoise`.

In practice, astronomers follow a different procedure. They point their telescopes at a region of uniformly lit sky during sunrise or sunset, or a sheet inside the dome. This produces a "flat-field image" of the form

$$\mathbf{I}_{flat} = (\mathbf{1} + \mathbf{N})C + \mathbf{\Theta}$$

where C is the mean photo intensity. Given this data, it is no longer necessary to estimate $\mathbf{N}$, since a corrected image is simply

$$\begin{aligned} \mathbf{I}_{corrected} &= C\frac{\mathbf{I}}{\mathbf{I}_{flat}} \\ &= C\frac{(\mathbf{1} + \mathbf{N})\mathbf{I}_{opt} + \mathbf{\Theta}}{(\mathbf{1} + \mathbf{N})C + \mathbf{\Theta}_{flat}} \\ &\approx \mathbf{I}_{opt} + \mathbf{\Theta}'. \end{aligned}$$

This procedure, which corrects for fixed pattern noise, can be applied without destroying sensitive super-resolution information. Note that some modern cameras perform flat-field correction as part of the built-in image processing pipeline.

Bibliography

[CJAE05] L. Castillo-Jimenez and M. Arias-Estrada. Super-resolution with integrated radial distortion correction. *Sixth Mexican International Conference on Computer Science (ENC'05)*, pages 165–173, 2005.

[FM06] Hilda Faraji and W. James MacLean. CCD noise removal in digital images. *IEEE Transactions on Image Processing*, 15(9):2676–85, September 2006.

[Fri09] J. Fridrich. Digital Image Forensics. *IEEE Signal Processing Magazine*, 26(2):26–37, 2009.

[HK94] G.E. Healey and R. Kondepudy. Radiometric CCD camera calibration and noise estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(3):267–276, 1994.

[Jan01] James R. Janesick. *Scientific charge-coupled devices*. SPIE Press, 2001.

[KB82] J. Kristian and M. Blouke. Charge-coupled devices in astronomy. *Scientific American*, 247:67–74, October 1982.

[Ker08] Douglas A. Kerr. The Proper Pivot Point for Panoramic Photography. http://doug.kerr.home.att.net/pumpkin/Pivot_Point.pdf, 2008.

[MKRM99] M. Mihcak, I. Kozintsev, K. Ramchandran, and P. Moulin. Low-complexity image denoising based on statistical modeling of wavelet coefficients. *IEEE Signal Processing Letters*, 6(12):300–303, 1999.

[Ray02] S.F. Ray. *Applied photographic optics*. Focal Press, 2002.

Chapter 3

The Discrete Pulse Transform

Parts of this chapter were excerpted from [FRV09]*, co-authored with* ***I. Fabris-Rotelli, University of Pretoria.***

3.1 Introduction

The Discrete Pulse Transform (DPT) decomposes a signal into a collection of pulses. In one dimension, a pulse is characterised by its start and end position as well as by its amplitude. In other words, a pulse is a number of adjacent positions that have a constant value. Similarly, in two dimensions, a pulse describes a connected region (adjacent values in the 4- or 8-connected sense) over which function values are constant (see Fig. 3.1).

The Discrete Pulse Transform is related to morphological image processing techniques, of which an overview is given in [SW09]. Unlike the discrete Fourier and wavelet transforms, the DPT is not a discretization of a continuous model [RL06], but is developed entirely in the discrete domain. Its ability to construct an image scale space is used in Chapter 4 to identify feature points.

The DPT is based on the $LULU$-operators, first suggested by CH Rohwer (his book, [Roh05], gives a thorough overview). The name $LULU$ derives from the L and U operators that are combined to produce the DPT—we'll examine

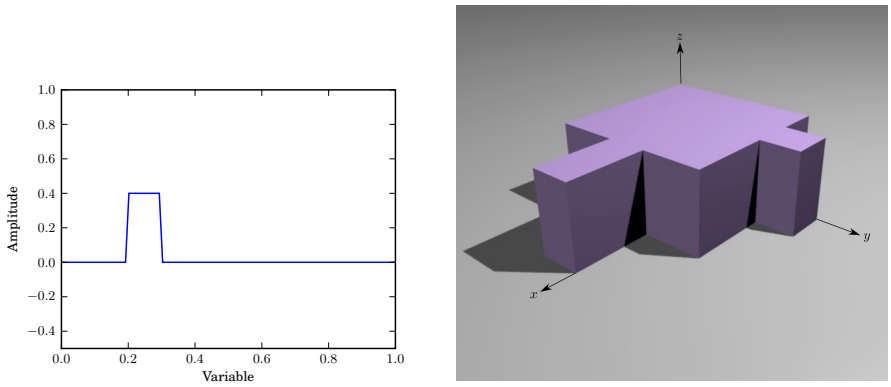


Figure 3.1: *Pulses in one and two dimensions. For the two-dimensional pulse, the amplitude is indicated by the z-axis, whereas the position is determined by x and y.*

these in more detail. The one-dimensional DPT was extended to two dimensions and higher by Anguelov and Fabris-Rotelli [AFR08].

In this chapter, we detail a fast and memory-efficient implementation of the two-dimensional discrete pulse transform.

3.2 Background theory

For a rigorous mathematical overview of the theory involved, we refer to the sources cited in the introduction. Here, we simply give a brief overview of the L and U operators in one dimension and how they can be extended to two dimensions.

Definition 1. In [LR06], the one-dimensional L_m and U_m operators, applied to a sequence $X = \begin{bmatrix} x_0 & x_1 & \dots & x_{N-1} \end{bmatrix}$, is defined as

$$\begin{aligned} L_m &= \vee_m \wedge_m \\ U_m &= \wedge_m \vee_m \end{aligned}$$

where

$$(\wedge_m X)_j = \min_{k=j-m,\dots,j} x_k \qquad \text{and} \qquad (\vee_m X)_j = \max_{k=j,\dots,j+m} x_k.$$

The simplest way of handling k outside $[0, N-1]$ is to neglect those values in the running minimum and maximum. Importantly, note that L_m removes all peaks of duration m, while U_m removes all troughs of duration m. After application of L and U, the sequence has fewer extrema than before, which leads to the naming $LULU$-smoother.

The L_m and U_m operators are applied in succession to form the levels of the one-dimensional discrete pulse transform, D_k. Starting with $X_0 = X$ (where the 0 in X_0 indicates an iteration number, not an index),

$$\begin{aligned} X_1 &= L_1 U_1 (X_0) \\ D_1 &= X_0 - X_1, \end{aligned}$$

whereafter the process is repeated to give

$$\begin{aligned} X_k &= L_k U_k (X_{k-1}) \\ D_k &= X_{k-1} - X_k. \end{aligned}$$

Note that D_k contains only pulses (consecutive elements with the same value) of duration k, and is otherwise zero. After a number of iterations, the sequence

X becomes monotone (or constant), whereafter the LU operator has no effect. The original sequence is reconstructed by summing the different levels of the decomposition:

$$X = \sum_L D_L.$$

Extension to two dimensions

The two-dimensional discrete pulse transform is obtained in much the same way, although the definitions of L_m and U_m have to be adapted. The extension to the two-dimensional case is given by Anguelov and Fabris-Rotelli (we refer to [AFR08] for an overview).

In two dimensions, we say that two pixels are connected if they are neighbours—typically in a 4- or 8-connected sense, i.e., North, South, East, West and possibly the diagonal directions. A connected set on (x, y) is the set including (x, y) and pixels connected to (x, y) via any other connected pixel. For example, $f(x, y)$ is connected to $f(x + 2, y)$ via $f(x + 1, y)$. We call such a connected set $\mathcal{N}(x, y)$. Specifically, a connected set containing $n + 1$ elements (that is, (x, y) itself plus n connections) is denoted $\mathcal{N}_n(x, y)$.

Definition 2. The two-dimensional operators, L_n and U_n, are defined as

$$\begin{aligned} L_n f(x,y) &= \max_{V \in \mathcal{N}_n(x,y)} \min_{(i,j) \in V} f(i,j), \ (x,y) \in \mathbb{Z}^2, \\ U_n f(x,y) &= \min_{V \in \mathcal{N}_n(x,y)} \max_{(i,j) \in V} f(i,j), \ (x,y) \in \mathbb{Z}^2. \end{aligned}$$

3.3 Implementation

The implementation given here was designed for the two-dimensional case and was first presented as [FRV09]. In the meantime, D. Laurie (who, together with C. Rohwer, first described a fast algorithm for calculating the discrete pulse transform in one dimension [LR06]) proposed a more general graph-based representation of the process that should allow an efficient implementation in higher dimensions. At the time of writing, no optimised implementation is available for comparison. In [SW09], a similar idea is discussed in the context of connected operators.

When decomposing an image into pulses using the DPT, the number of pulses may vary from approximately 30,000 for a typical 300×300 image to over a hundred thousand for a 500×500 image. We need an efficient storage scheme to represent such a large number of pulses in memory. Furthermore, we need to be able to calculate certain attributes of the pulses (such as the

area and the boundary values) rapidly, which excludes other common storage schemes such as run-length encoding.

Section 3.3.1 describes how to represent pulses efficiently in memory, how to calculate their area and boundaries, and how to merge two adjacent pulses. Section 3.3.2 then outlines a procedure for computing the two-dimensional discrete pulse transform.

3.3.1 Pulse representation and manipulation

Storage

The storage scheme used is based on the popular Compressed Sparse Row (CSR) format, [DGL89, BBC+94], for representing sparse matrices. Using this scheme, the matrix

$$\begin{bmatrix} 5 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 9 & 0 \end{bmatrix} \text{ is written as}$$

- `values` $= \begin{bmatrix} 5 & 1 & 2 & 3 & 6 & 9 \end{bmatrix}$
- `columns` $= \begin{bmatrix} 0 & 2 & 3 & 3 & 1 & 3 \end{bmatrix}$
- `row_offset` $= \begin{bmatrix} 0 & 3 & 4 & 4 & 6 \end{bmatrix}$

The values of the non-zero elements are stored in `values`, and their column-positions given by `columns`. Each entry of `row_offset` specifies an offset into `columns`, indicating the starting position of a new row. In the example above, we see that the second row (second element of `row_offset`) starts at position 3 of `columns`. The number of elements in row j is given by `row_offset`$[j+1] -$ `row_offset`$[j]$.

When storing two-dimensional pulses, we know that the pulse

- may only occupy a small portion of the image,
- has a single amplitude value across the pulse and
- consists of regions connected horizontally (as well as vertically and diagonally, but that is not relevant here).

We therefore modify the storage structure, so that the pulse

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix} \text{ is written as}$$

- `value` $= 1$
- `columns` $= \begin{bmatrix} 0 & 5 & 1 & 3 & 4 & 5 & 1 & 4 \end{bmatrix}$
- `start_row` $= 1$
- `row_offset` $= \begin{bmatrix} 0 & 2 & 6 & 8 \end{bmatrix}$

Instead of specifying column values, `columns` now indicates the start and past-end indices of the one-dimensional pulses that comprise the rows. The values of `row_offset`, as in the previous example, specify where in `columns` each new row starts. The pulse may only cover a few rows of the entire image, therefore we use `start_row` to indicate the first occurrence, saving us from storing every single row.

As an example, consider the third row of the two-dimensional pulse above, which consists of two one-dimensional pulses: the first stretching from column 1 up to (but excluding) 3, the other from 4 up to 5. Since we are interested in the third row (row number 2), and we only start recording rows at `start_row` $= 1$, we find the corresponding column indices in `row_offset`$[2-1] = 2$. At position 2, `columns` contains 1, 3 and 4, 5 as expected.

An advantage of this storage scheme is that it can also be used to store connected regions, a capability we exploit later to initialise the algorithm.

Queries

Given a pulse in the above format, we would like to calculate the following queries rapidly:

Area / number of non-zeros The area of the pulse is the sum of the lengths of the one-dimensional pulses comprising its rows. Each such length is given as the corresponding difference between the pulse start-end positions in `columns`. In the example above, the area would be $(5-0)+(3-1)+(5-4)+(4-1) = 5+2+1+3 = 11$.

Adjacent Set / Boundary positions Each pulse has four or more boundary positions – connected to the pulse in a 4- or 8-connected sense (see Fig. 3.2 for an illustration of the 4-connected case) – that form the adjacent set. To find the

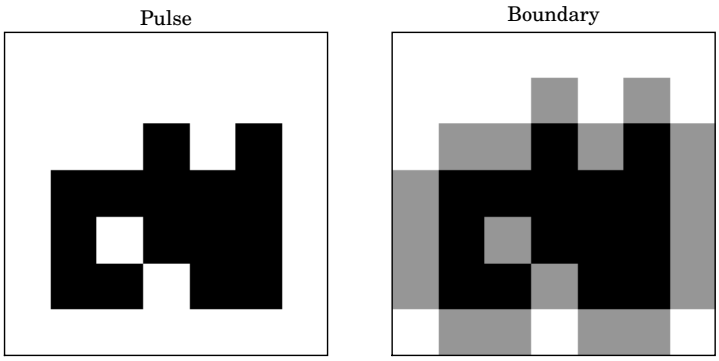


Figure 3.2: *Boundary positions of a pulse.*

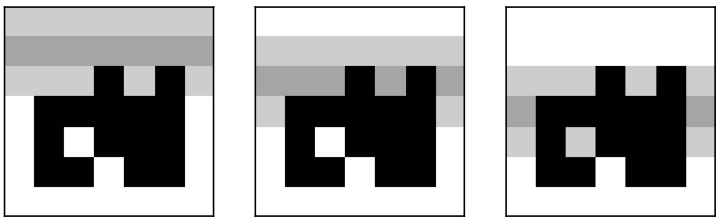


Figure 3.3: *Scanlines used to find boundary positions.*

boundary positions, we follow a scanline approach, with three scanlines moving from the top of the pulse to the bottom (see Fig. 3.3). Here, we describe the operation once the scanlines have entered the pulse (in other words, neglecting top and bottom boundaries, which need to be handled separately):

1. The scanlines are centred around row j and are formed by constructing the pulse at rows $j-1$, j and $j+1$. The scanline is wider by one pixel on each side than the pulse itself.

2. For each element of the central scanline that does *not* belong to the pulse, determine whether any of its neighbours (above, below, left, right or diagonally, in the 8-connected case) belong to the pulse. If they do, then the current element lies on the boundary. Note that only elements covered by the central scanline are analised at any stage.

3. Move the scanlines one row down and repeat (it is only necessary to recalculate the bottom scanline at each step).

Another way to implement this algorithm would have been using interval trees, but the structures and memory allocations involved are more complex, so it is not obvious without a benchmark whether that would be beneficial.

Operations

Merging Two Pulses Later on, when performing the Discrete Pulse Transform, we are required to merge two pulses that touch. This is done on a row-by-row basis. In the trivial case where a row is contained in only one of the two pulses, we simply include that row in the output. Otherwise, we need to sort and join the one-dimensional pulses that comprise the row carefully. Note, however, that these one-dimensional pulses cannot overlap in our problem description. We therefore:

1. Extract the stop-start intervals that form the one-dimensional pulses in row j.
2. Sort the intervals according to their starting position.
3. Step over the intervals and link (join) them if they touch.
4. Save the linked intervals as the representation of row j.
5. Repeat for row $j + 1$.

3.3.2 Algorithm for computing the 2D Discrete Pulse Transform

Each step of the Discrete Pulse Transform is now described in more detail. We use the following terms:

Input image The input image or data – an $M \times N$ matrix of integer values between 0 and 255.

Label image An $M \times N$ array of integer values that indicate the connectivity of pixels in an image. If neighbouring pixels have the same value (i.e. are 4-connected), then they are assigned the same label value.

Intermediate reconstruction An $M \times N$ image can be decomposed into pulses with areas ranging from 1 through MN . When added together, these pulses reconstruct the **input image**. It is also possible to only add pulses with area $> k$. We call this an intermediate reconstruction, as it only approximates the image up to a certain level.

The algorithm consists of three steps:

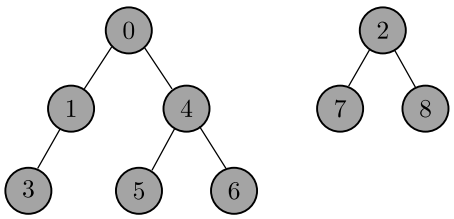


Figure 3.4: *Two trees with labelled nodes.*

1. Find all connected regions / adjacent sets in the image. Among these sets we find all pulses of size 1, which form the first level of the decomposition.
2. Apply the L_1 and U_1 operators, that remove peaks and troughs of size 1, to the adjacent sets. Removed parts of the signal is stored as the level 1 (or area 1) pulses.
3. Repeat this process with L_2 and U_2 to extract pulses of area 2 (the second level of the decomposition), then L_3 and U_3, etc.

Here follows a more detailed overview of each step involved.

Finding Connection Regions

First, we identify all 4 or 8 connected regions in the image (these are the initial pulses that are processed to yield the Discrete Pulse Transform). This is done using the Union-Find connected component algorithm of Fiorio and Gustedt [FG96], with the connectivity tree stored in an array as suggested by Wu *et al.* in [WOS05]. It is further shown in [WOS05] that this algorithm executes in $\mathcal{O}(N)$, and we give a brief overview of its functioning[1]:

Representing a tree using an array One or more trees consisting of N nodes can be stored in an array of length N. Examine the trees shown in Figure 3.4 with nodes labeled $n = 0, \ldots, 8$. These trees can be represented as the array

$$\mathbf{x} = \begin{bmatrix} 0 & 0 & 2 & 1 & 0 & 4 & 4 & 2 & 2 \end{bmatrix}$$

[1] *Note on Graphics Processing Unit (GPU) implementation*: Algorithms are well suited to a GPU if elements of the solution can be calculated in isolation. When finding connected components, each element depends on its neighbours, and possibly all the other pixels in the input, consequently the problem is not easy to parallelise.

In his master's thesis [O'C09], S. O'Connell adapted the connected component algorithm for implementation on a GPU. For the reason given above, the performance on the GPU is less than optimal and roughly similar to that of the CPU (9ms for a 512×512 image – the same timing achieved by our CPU-based implementation).

where $\mathbf{x}_n$ gives the parent of node n. For example, $x_3 = 1$, which tells us that the parent of node three is node one. Similarly, $x_2 = 2$ implies that node two has no parent—it is the root of a tree.

Labelling connected regions as trees The goal of the connected components algorithm is to assign unique labels to each connected region in an $M \times N$ image I. An array, L, of length MN is used to store trees as indicated in the paragraph above.

The image is traversed in raster scan order (i.e. along rows). A region counter, k, is initialised to zero. At each pixel position (r, c):

1. Calculate the offset into the tree array as $t = rN + c$.
2. If the pixel is not connected to (does not have the same value as) the pixel above it or to the left, assign $L_t = t$, effectively creating a new tree.
3. If the pixel is connected to the pixel above, assign $L_t = L_{t-N}$, joining node t to its parent in the previous row.
4. If, in addition, the pixel is connected to the left, assign $L_{t-1} = L_{t-N}$.
5. If the pixel is only connected to the left, assign $L_t = L_{t-1}$.

Appropriate care needs to be taken in the first row and column to prevent indexing errors on the image boundary.

The label vector, L, can also be seen as the flattened version of a *label image* so that $L_{r,c} = L_{rN+c}$. From this image, all connected regions are extracted as pulses and stored in the format discussed in Section 3.3.1.

We then proceed to perform the Discrete Pulse Transform as discussed next.

Identifying Pulses to Merge The Discrete Pulse Transform is performed by alternately executing the L_k (lower) and U_k (upper) operators that extract pulses of area k. If you think of the image as a height-map, then the U_1-operator removes all valleys of area one. Here, a valley is defined as a connected area that is surrounded only by higher values. Similarly, the L_1-operator removes peaks of area one, where peaks are connected areas surrounded only by lower values.

After applying the L_1- and U_1-operators and *storing the removed peaks and valleys* (those form the first level of the DPT), we need to merge pulses that were joined in the process. Note that, at each decomposition level, we have the *intermediate reconstruction* available. It is obtained by setting the image values corresponding to the removed positive (negative) pulses equal to the maximum (minimum) value on the adjacent set.

For each pulse, we calculate its boundary positions using the method described in Section 3.3.1. We then examine the boundary values on the *intermediate reconstruction*, and if any of those values are equal to the pulse value, a merge is required. After examining all boundary positions, a list is drawn up of all coordinates that fall on merge boundaries. At each of those positions, a merge is performed as described in Section 3.3.1, after which the *label image* is updated.

The L_{k+1} and U_{k+1} operators are now applied, repeating the merging process for higher values of k until the image has been entirely decomposed (in other words, until the final MN-sized pulse has been removed). All the removed pulses together form the Discrete Pulse Transform or decomposition.

3.3.3 Benchmark

The decomposition algorithm described was implemented an executed on a Intel Core Duo 3.16GHz processor. Memory utilisation peaked at less that 150MB during decomposition of *Airplane* and roughly 60MB while processing *Chelsea* (including the memory required to store the decomposition itself). Computation times were 3.73s (*Airplane*) and 1.51s (*Chelsea).* Reconstruction executed in a few milliseconds.

Figure 3.6 shows benchmark times for the discrete pulse transform applied to random matrices. Random matrices with a large number of discrete values seem to be the worst case—execution times are much lower for real photographs and for signals limited to, say, 255 discrete values. Both these observations are explained intuitively: a random matrix has many more pulses than a typical photograph and limited discrete values cause merging of pulses that would otherwise remain separated. It would be interesting to investigate whether a link exists between image entropy and the number of pulses generated.

Software

The Discrete Pulse Transform is implemented as `supreme.lib.dpt`. The decomposition is obtained as follows:

```
import supreme.api as sr

image = sr.test_data()

# Perform the decomposition -- this may take
# 5-10 seconds
pulses = sr.lib.dpt.decompose(image.astype(int))

# Reconstruct the original from the pulses
Z = sr.lib.dpt.reconstruct(pulses, image.shape)

# Display the two images side-by-side
sr.show(image, Z)
```

The returned ‘`pulses`‘ is a dictionary indexed by area, i.e., `pulses[2]` contains all pulses of area 2 (those that form the second level of the decomposition).
The reconstruction simply adds all the pulses together, and can also be done manually, as illustrated below. Note the use of the `connected_region_handler` to calculate pulse values and to add pulses to the output array. This is necessary because the pulse itself is stored in a sparse format, as discussed later in this chapter.

```
import numpy as np
from supreme.lib.dpt import \
    connected_region_handler as crh

# Create an output image
out = np.zeros(image.shape, dtype=int)

# Reconstruct output by adding all the pulses
for area in pulses:
    for p in pulses[area]:
        crh.set_array(out, p, crh.get_value(p), 'add')

sr.show(out)
```

Figure 3.5: *Two test images used for benchmarking the 2D DPT. On the left is* Chelsea the Cat *(300 × 351) and on the right is* Airplane *(512 × 512).*

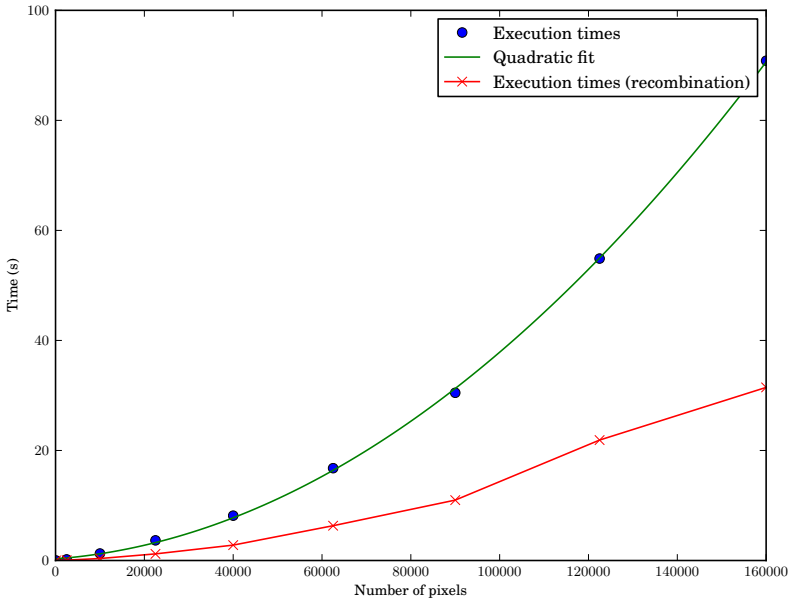


Figure 3.6: *Benchmark of the discrete pulse transform on random images of varying size. Values on the x-axis indicate the total number of pixels, i.e., N^2 for an $N \times N$ matrix. In the bottom curve, labeled "recombination", the number of discrete input values were limited to 255. For large images, this causes the algorithm to execute more quickly than expected due to accidental merges.*

Bibliography

[AFR08] Roumen Anguelov and Inger Fabris-Rotelli. Discrete Pulse Transform of Images. *Lecture Notes In Computer Science*, 5099:1-9, 2008.

[BBC+94] R. Barrett, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition. 1994.

[DGL89] I. S. Duff, Roger G. Grimes, and John G. Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software*, 15(1):1–14, March 1989.

[FG96] Christophe Fiorio and Jens Gustedt. Two linear time Union-Find strategies for image processing. *Theoretical Computer Science*, 154(2), 1996.

[FRV09] I. Fabris-Rotelli and S.J. Van Der Walt. The Discrete Pulse Transform in Two Dimensions. In F. Nicolls, editor, *Proceedings of the twentieth annual symposium of the Pattern Recognition Association of South Africa (PRASA)*, Stellenbosch, South Africa, 2009.

[LR06] D.P. Laurie and C.H. Rohwer. Fast implementation of the Discrete Pulse Transform. In T.E. Simos, G. Psihoyios, and Ch. Tsitouras, editors, *ICNAAM 2006: International Conference on Numerical Analysis and Applied Mathematics*, pages 484–487, Weinheim, 2006. Wiley-VCH.

[O'C09] Sean M. O'Connell. A GPU Implementation of Connected Component Labeling. Master's thesis, Lousiana State University, 2009.

[RL06] C. H. Rohwer and D. P. Laurie. The Discrete Pulse Transform. *SIAM Journal on Mathematical Analysis*, 38(3):1012, 2006.

[Roh05] Carl H. Rohwer. *Nonlinear smoothing and multiresolution analysis*. Birkhäuser Verlag, Basel, Boston, 2005.

[SW09] Philippe Salembier and Michael Wilkinson. Connected operators. *IEEE Signal Processing Magazine*, 26(6):136–157, 2009.

[WOS05] K. Wu, E. Otoo, and A. Shoshani. Optimizing connected component labeling algorithms. Technical report, Lawrence Berkeley National Laboratory, Berkeley, California, 2005.

Chapter 4

Feature detection and matching

4.1 Introduction

Super-resolution imaging consists of three important steps: image acquisition, registration (or alignment) and reconstruction. Under most circumstances, the accuracy of image registration, discussed in detail in Chapter 5, has a profound impact on the quality of the reconstruction. As such, it deserves to be treated with care.

The reconstruction process relies on the registration to indicate corresponding positions in the different low-resolution input frames. Without this information, the problem becomes intractable. In this chapter, we develop a feature detector, used to locate significant features in the image. Of course, the definition of "significant" may change, depending on a specific application. Here, we simply refer to a point that can be consistently detected by an algorithm over multiple images that undergo geometric, photometric or other distortions.

After features have been found, the surrounding pixels are usually analysed to construct a feature descriptor that facilitates matching (finding corresponding features across multiple images). This is a good idea. However, in this chapter we would like to focus only on two fundamental ideas: 1) locating features using the Discrete Pulse Transform and 2) matching those features based on the surrounding pixels (or image patches).

Some excellent feature detection/representation routines, such as SIFT [Low04] and SURF [BTV06], exist, but these are patent encumbered. Maximally Stable Extremal Regions (MSER) [FL07] looks promising. Detectors that work well include FAST [RD06] (included in the accompanying software) and the enhanced Harris detector proposed in [MS05]. While the commonly used Harris corner detector [HS88] is starting to show signs of age, it still has the advantage of being easy to implement. A review of feature detectors is given in [TM07] and of feature descriptors in [MS05].

4.2 Feature detection using the discrete pulse transform

4.2.1 Scale space

The discrete pulse transform decomposes an image into a sum of differently sized pulses (see Chapter 3). Pulses of a fixed size form a single level of the DPT, and the different levels a scale space, similar in concept to the well known Gaussian scale space, on which many feature detectors such as SIFT rely.

Given all pulses $p_{k,i}(x,y) \in D_k$ belonging to the different levels $k = 1 \dots K$, the original signal can be reconstructed using

$$f(x,y) = \sum_k D_k(x,y) = \sum_k \sum_i p_{k,i}(x,y).$$

The range of i varies according to the number of pulses at any specific level. A single pixel, (x,y), belongs to zero or more pulses of different sizes (in other words, it is included in zero or more levels of the decomposition). A partial reconstruction of the signal that suppresses higher levels of detail is given by

$$\hat{f}_L = f - \sum_{k=1}^{L-1} D_k \quad \text{or equivalently} \quad \hat{f}_L(x,y) = f(x,y) - \sum_{k=1}^{L-1} \sum_i p_{k,i}(x,y).$$

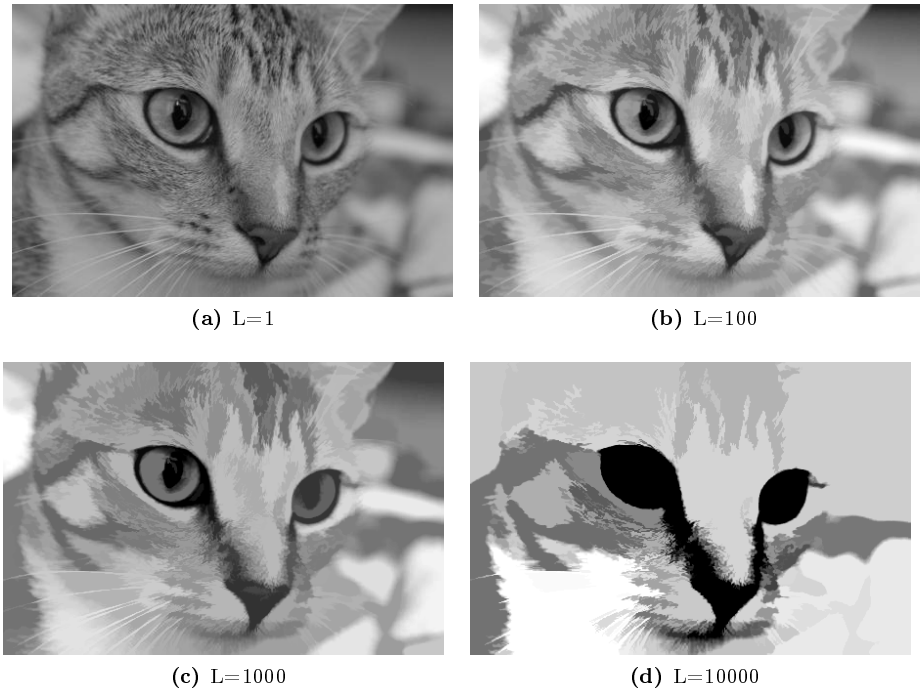
This is the image reconstructions at scale L. Scale 1 represents the original image, while level K represents only the largest pulses. At scale L, only size L and larger pulses are used in the reconstruction. The difference between two layers, L and $L+1$, is $D_L = \sum_i p_L$.

4.2.2 Pulse strength

According to the Cambridge Advanced Learner's Dictionary, a feature is "a typical quality or an important part of something". Indeed, we hope to isolate features that are robust, i.e. they remain recognisable under moderate geometric and photometric transformations.

The DPT provides an easy way to identify prominent features: we simply count the number of pulses in which a pixel occurs. The strength of pixel (x,y) is

$$s(x,y) = \sum_k \sum_i \delta_{x,y}(p_{k,i})$$



(a) L=1 **(b)** L=100

(c) L=1000 **(d)** L=10000

Figure 4.1: *Reconstruction of* Chelsea *at different scales. The changes in brightness are due to automated scaling, done to make pulses (especially in the higher levels) more visible.*

(a) Input Image **(b)** Pulse strength.

Figure 4.2: *The pulse strength of an image, used to identify important segments. In this case, the pulse strength was calculated for all pulses of size less than 100. Darker values indicate stronger pulses.*

where the Dirac-measure,

$$\delta_x(V) = \begin{cases} 1 & \text{if } x \in V \\ 0 & \text{if } x \notin V, \end{cases}$$

indicates whether a pixel x belongs to a given pulse V. We may choose to limit k to small values (say $k < 100$) since we do not expect good features to be larger than roughly 10×10, and even if they are, we are not interested in their pulse attributes until the pulses become fairly small (to be useful, features need to be well localised). As we can see from the examples shown in Fig. 4.2, the pulse strength alone is not enough to isolate features. Rather, the pulse strength highlights areas in which we may expect to find strong features. The question then becomes: what differentiates a good feature from the rest?

Intuitively, we set two criteria:

1. A good feature has a large pulse strength (i.e., occurs in many layers of the scale space).

2. As we traverse the scale space from K down to 1, a good feature rapidly shrinks in size and culminates in a sharp point.

While it is possible to use shape descriptors to analyse larger features (as done in [FL07]), we limit ourselves to sharp peaks for simplicity.

If we visualise a feature as a pyramid rising from the ground, then ideally it should be built with sunken edges as shown in Fig. 4.3. Relative to the pulse size, the amplitude should increase rapidly. We use the following measure of a feature's sharpness (again, we may choose to restrict $k < 100$):

$$t(x, y) = \sum_k \sum_i \frac{\left| p_{k,i}(x, y) \right|}{\sqrt{k}}.$$

Finally, we combine the two criteria,

$$m(x, y) = t(x, y)s(x, y),$$

to form a feature map. Peaks are located by examining the closest surrounding neighbours, and those with the highest values are chosen as features. The result is shown in Fig. 4.4.

At this stage, with the feature positions determined, a feature descriptor can be calculated based on the surrounding pixels. This usually also includes a filtering step, where highly symmetric and other "dangerous" features (that easily mismatch) are also removed (see [HS88] for a technique commonly used).

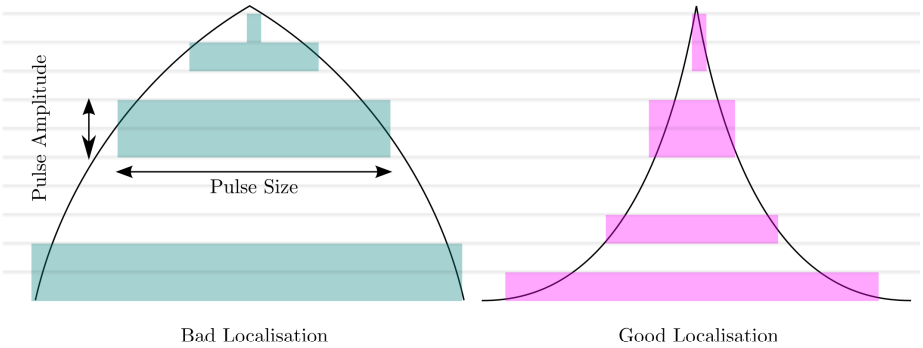


Figure 4.3: *Pulse localisation of features. Features that quickly form a thin, sharp point are more desirable than slower growing ones.*

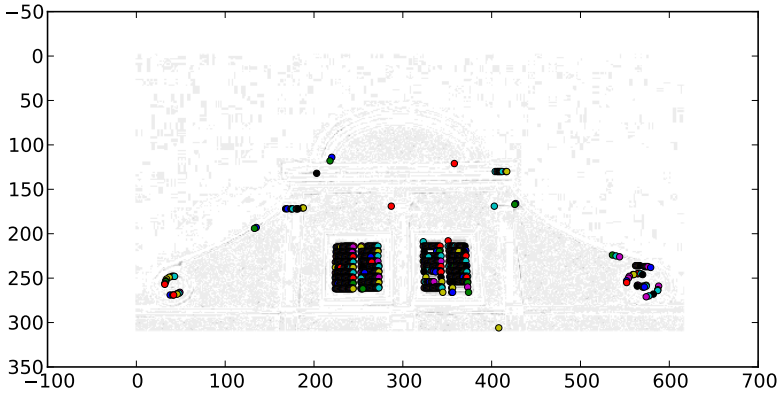


Figure 4.4: *Feature map of* House *with the 1000 best features highlighted. Note how the features are located on corners and other areas of interest.*

In the next section, we discuss matching image patches cut out around the detected features. The DPT decomposition provides a hint as to what the sizes of those patches should be: if a feature is composed mainly of large pulses, then it describes a larger part of the image, and its image patch should, correspondingly, cover a similar area. We calculate the feature area as an average over pulse sizes,

$$A(x,y) = \frac{1}{s(x,y)} \sum_k k\delta_{x,y}(p_{k,i}(x,y)).$$

In the correspondence matching that follows, the window size is set to the average feature area, $\bar{A} = \sum_{x,y} A(x,y)$. In our implementation, we also limit the values of $\bar{A}$ to be no larger than 100 to reduce matching computation time.

Software

A **feature map** based on the discrete pulse transform can be calculated using the `features` function from `supreme.feature.dpt`.

```
from supreme.feature.dpt import features

# Calculate image features, based on pulses from the
# DPT. The parameter 'max_area' indicates the largest
# pulses taken into account when calculating the
# feature characteristics.
fmap, farea = features(pulses, img.shape, max_area=100)
```

4.3 Matching and correspondence

This section presents a statistical method for finding feature correspondences, based on surrounding pixel values (or image patches). This method is well suited to situation where small to moderate geometric changes occur (as is often the case with most super-resolution sequences). For ways of robustly matching features over large geometric distortions, see [TRD09].

4.3.1 Statistical feature comparison

A number of methods have been proposed that rely on classifiers to do feature matching [SD01, RD06]. Other methods rely on statistical moments of the pixels surrounding a feature [She07]. We present a statistical method that is extremely fast and simple to implement. The idea is to compare the rows of two candidate image patches (think of them as matrices of pixels) by calculating matching scores for each corresponding pair. The correspondence likelihood is based on a Quantile-Quantile score, derived from QQ plots.

Quantile-Quantile (QQ) plots

A QQ-plot is a graphical technique used to determine whether two data sequences are drawn from the same distribution. The quantiles (the fraction of points below given probability thresholds) of the first data set are plotted against those of the second. If the data sets come from the same distribution, the result should be a straight line.

When the two data-sequences to be compared, let's say $\mathbf{x}$ and $\mathbf{y}$, are equal in length N, a simple way to form the QQ-plot is by sorting the sequences from small to large and then connecting points (x_i, y_i) from $i = 0$ to $N - 1$. The deviation from the straight line can be measured as

$$\boldsymbol{\Delta} = \mathbf{x} - \mathbf{y}.$$

Since both $\mathbf{x}$ and $\mathbf{y}$ contain only positive elements, we can normalise $\boldsymbol{\Delta}$ using the norms of the components,

$$\Delta_1 = \frac{\|\boldsymbol{\Delta}\|_2}{\max(\|\mathbf{x}\|_2, \|\mathbf{y}\|_2)}.$$

The higher the value of Δ_1, the larger the mismatch in distribution between the two sequences. The algorithm for calculating correspondences, based on this principle, is given as Algorithm 4.1.

To understand why the 2-norm of Q (as referred to in the algorithm) is a good indication of the patch deviation, consider using the 1-norm,

$$\|Q\|_1 = \max_n \sum_m \left|q_{m,n}\right|,$$

the maximum absolute column sum. This is akin to measuring the maximum error over all quantiles. The ∞-norm,

$$\|Q\|_\infty = \max_m \sum_n \left|q_{m,n}\right|,$$

gives the maximum absolute row sum—which measures the maximum deviation across patch rows. The 2-norm,

$$\|Q\|_2 = \max_{x \neq 0} \frac{\|Q\mathbf{x}\|_2}{\|\mathbf{x}\|_2},$$

lies somewhere in between, in the sense that

$$\frac{1}{\sqrt{N}} \|Q\|_1 \leq \|Q\|_2 \leq \sqrt{N} \|Q\|_\infty.$$

Algorithm 4.1 Calculating point-correspondences using Quantile-Quantile comparisons.

Given the features in two images, A and B, calculate likely feature correspondences (i.e., which feature in A is represented by which feature in B).

1. Cut out image patches centered around the features in both A and B. The patches in A are called $P_{A,i}$ and those in B are $P_{B,i}$—they form $N \times N$ matrices. The size of patches can be determined using the discrete pulse transform as described above, or alternatively set to the expected size of the average image feature (roughly between 10×10 and 50×50).

2. For each patch, sort the component of its rows, i.e.

$$P_{A,i} = \begin{bmatrix} 3 & 9 & 2 \\ 1 & 8 & 3 \\ 4 & 4 & 1 \end{bmatrix} \Longrightarrow P_{A,i} = \begin{bmatrix} 2 & 3 & 9 \\ 1 & 3 & 8 \\ 1 & 4 & 4 \end{bmatrix}.$$

3. For each patch $P_{A,i}$, calculate the distribution deviation against all patches from B:

   (a) Construct the matrix

$$Q = \begin{bmatrix} \mathbf{e}_0 \\ \mathbf{e}_1 \\ \vdots \\ \mathbf{e}_{N-1} \end{bmatrix}$$

   where $\mathbf{e}_n = P_{A,i}(n,*) - P_{B,j}(n,*)$ with $P_{A,i}(n,*)$ being the n-th row of $P_{A,i}$.

   (b) Calculate the patch deviation between patch $P_{A,i}$ and $P_{B,j}$ as

$$D_{i,j} = \frac{\|Q\|_2}{\max(\|P_{A,i}\|_2, \|P_{B,j}\|_2)}.$$

4. The best correspondence for patch $P_{A,i}$ in B then is

$$\operatorname*{argmin}_j D_{i,j}.$$

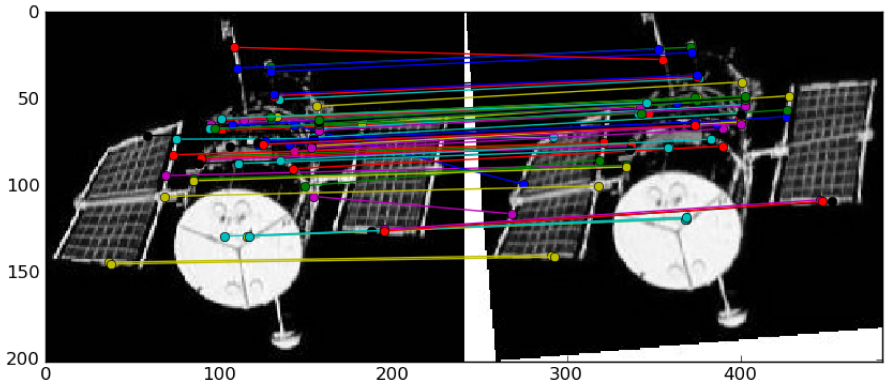


Figure 4.5: *Correspondence matching on a picture of the Hubble Space Telescope (taken from the NASA archives). The second frame is a rotated version of the first. Note that the majority of, but not all, correspondences are correct.*

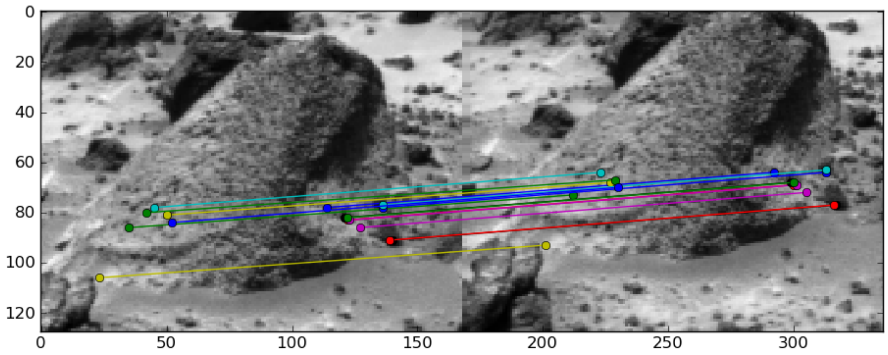


Figure 4.6: *Correspondence matching on real-world data. These photos were taken by NASA during the Pathfinder missions.*

Because the algorithm does not compute pixel intensity differences, but rather compares intensity distributions, it is fairly robust to mild geometric transformations. The output of the algorithm, applied to both an artificial and a real-world data set, is shown in Figures 4.5 and 4.6.

4.3.2 Other matching methods

After image patches have been cut out around detected features, several algorithms are available to find correspondences. We mention a few.

Cross-Correlation

The cross-correlation between two $M \times N$ patches, $A(m,n)$ and $B(m,n)$, is defined as

$$C(i,j) = \sum_m \sum_n A(m,n)B(m+i,n+j).$$

Unfortunately, the cross-correlation function has no well defined range, and is sensitive to changes in patch luminosity.

Normalised Cross-Correlation

The normalised cross-correlation aims to address problems with standard cross-correlation, and is defined as

$$\gamma(m,n) = \frac{1}{MN} \frac{\sum_{x,y} \left[A(x,y) - \bar{A}_{u,v}\right] \left[B(x-u,y-v) - \bar{B}_{u,v}\right]}{\sqrt{\sum_{m,n} \left|A(m,n) - \bar{A}_{u,v}\right|^2 \sum_{m,n} \left|B(m-u,n-v) - \bar{B}_{u,v}\right|^2}}, \tag{4.1}$$

where $\bar{A}_{u,v}$ and $\bar{B}_{u,v}$ are the means of the overlapping parts of A and B and $\gamma \in [-1,1]$. Unfortunately, the order-complexity of the calculation is high at $\mathcal{O}(M^2N^2)$.

Suggestions have been made for speeding up its *direct* calculation [Lew95, TL03] (other methods are available to derive an *estimate*), based on summed area tables (SAT). The SAT of an image $f(x,y)$ is

$$S(m,n) = \sum_{i \le m, j \le n} f(i,j),$$

so that $S(m,n)$ is the sum of all elements to the top and left of the current element (including the element itself). It can be calculated efficiently in a single pass, since

$$S(m,n) = f(m,n) + S(m-1,n) + S(m,n-1) - S(m-1,n-1).$$

The SAT can then be used to sum any block-shaped area in the image, since

$$\sum_{m_0 < m < m_1} \sum_{n0 < n < n1} f(m,n) = S(m_0,n_0) + S(m_1,n_1) - S(m_0,n_1) - S(m_1,n_0).$$

For random variables, we know that

$$\mathsf{E}([X - \bar{X}]^2) = \mathsf{E}(X^2) - \bar{X}^2.$$

In a similar way, we can rewrite (4.1) to become

$$\gamma(u,v) = \frac{1}{MN} \frac{\sum_{x,y} \left[A(x,y) - \bar{A}_{u,v}\right] \left[B(x-u, y-v) - \bar{B}_{u,v}\right]}{\sqrt{\left[\sum_{x,y} A^2(x,y) - \left(\sum_{x,y} A(x,y)\right)^2\right] \left[\sum_{x,y} B^2(x,y) - \left(\sum_{x,y} B(x,y)\right)^2\right]}}.$$

The summed area tables of A and A^2 can then be used to calculate the entire denominator and the means in the numerator. Especially for larger patches, this may lead to significant speed-ups.

When it comes to determining patch correspondence, any form of correlation is problematic in the sense that it does not account for geometric transformations other than shifts in the x and y directions. It may be successful for small patch sizes, but for larger sizes we suggest the statistical approach above, or using it in combination with the log-polar transform, outlined in the next chapter.

Phase correlation

Phase correlation is similar to standard correlation, except that it operates in the phase domain. It can be calculated rapidly using the discrete Fast Fourier Transform as

$$C = \mathcal{F}^{-1} \left\{ \frac{\mathcal{F}\{A\} \circledcirc \mathcal{F}^*\{B\}}{\left|\mathcal{F}\{A\} \circledcirc \mathcal{F}^*\{B\}\right|} \right\},$$

where $\mathcal{F}$ is the Fourier-transform (in two dimensions) and $\circledcirc$ indicates pointwise multiplication. In fact, this calculates the circular convolution (i.e., values that "exit" on one side "enter" on the other). It is based on the observation that two sequences, circularly shifted with respect to one another, have a phase difference in the Fourier domain. In this ideal case, C is the Kronecker delta

$$\delta(x + dx, y + dy)$$

indicating the shift. In the next section, where we discuss the log-polar transform, the circular property is needed. In most other practical cases, the shift is not circular, so that the patches have to be zero-padded to dimensions $(2M - 1) \times (2N - 1)$ before transforming.

Extensions of phase correlation, aimed at registering rotated as well as translated images, have been proposed [DM87, RC96].

Software

The algorithms listed here are available as:

- `supreme.register.ncorr` *(normalised cross-correlation)*
- `supreme.register.phase_corr` *(phase correlation)*

4.4 Conclusion

This chapter details several pixel-based techniques of determining feature correspondence. With the feature correspondences between two images A and B known, we now turn to the next problem: how to estimate the geometric transformation that takes pixels from A to B. It is unlikely that all correspondences are determined correctly, so Chapter 5 also shows how to deal with outlier correspondences robustly, using techniques such as RANdom SAmple Consensus (RANSAC).

Bibliography

[BTV06] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *Lecture notes in computer science*, 3951:404, 2006.

[DM87] E. De Castro and C. Morandi. Registration of Translated and Rotated Images Using Finite Fourier Transforms. *IEEE Trans. Pattern Anal. Mach. Intellig.*, 9(5):700–703, 1987.

[FL07] Per-Erik Forssen and David G. Lowe. Shape Descriptors for Maximally Stable Extremal Regions. *IEEE 11th International Conference on Computer Vision*, October 2007.

[HS88] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[Lew95] J. P. Lewis. Fast Normalized Cross-Correlation. `http://scribblethink.org/Work/nvisionInterface/nip.pdf`, 1995.

[Low04] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[MS05] Krystian Mikolajczyk and Cordelia Schmid. Performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–30, October 2005.

[RC96] B.S. Reddy and B.N. Chatterji. An FFT-based technique for translation, rotation, and scale-invariant image registration. *IEEE Transactions on Image Processing*, 5(8):1266–71, January 1996.

[RD06] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. *Lecture Notes in Computer Science*, 3951:430, 2006.

[SD01] R. Sim and G. Dudek. Learning generative models of scene features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR) 2001*, 1:406–412, 2001.

[She07] Dinggang Shen. Image registration by local histogram matching. *Pattern Recognition*, 40:1161–1172, 2007.

[TL03] Du-Ming Tsai and Chien-Ta Lin. Fast normalized cross correlation for defect detection. *Pattern Recognition Letters*, 24(15), 2003.

[TM07] Tinne Tuytelaars and Krystian Mikolajczyk. Local Invariant Feature Detectors: A Survey. *Foundations and Trends in Computer Graphics and Vision*, 3(3):177–280, 2007.

[TRD09] Simon Taylor, Edward Rosten, and Tom Drummond. Robust feature matching in 2.3μs. In *IEEE CVPR Workshop on Feature Detectors and Descriptors: The State Of The Art and Beyond*, 2009.

Chapter 5

Accurate image registration

5.1 Introduction

Image registration is an important part of super-resolution: it provides an estimate of the geometric relationship between the input frames, thereby informing us as to which pixels from which frames describe a certain part of the scene. It has been shown that a minute amount of misregistration helps to regularise[1] the super-resolution estimate [CB09] , but it is easy to argue that inaccurate registration should be avoided at all cost (adding regularisation is easy, as we shall see in Chapter 6, but fixing misalignment is not).

In the previous chapter, we show how to find corresponding features across two frames, say $f_0(x, y)$ and $f_1(x, y)$. With those tentative correspondences (we call them tentative, because they may not be correct) at our disposal, we are interested in estimating a geometric transformation, T, such that

$$f_0(x, y) = f_1(T(x, y)).$$

The function T can be very complex—the image formation process likely yields images that vary non-linearly in their geometry (radial lens distortion, for example, varies non-linearly as a function of the radius from the centre of the image). As discussed in Chapter 2, though, these effects may be of little consequence for super-resolution since

- images are taken from afar,
- the regions involved are small parts of a larger image,
- neither the camera nor the object moves any significant amount.

Of course, these assumptions are not *required* for super-resolution, but they do make the registration problem more tractable. Given these circumstances, we are able to model the changes in geometry for the majority of super-resolution

[1] As discussed in later chapters, maximum likelihood estimators tend to be overzealous sometimes, yielding oscillating results.

problems as a linear coordinate transformation,

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} = H \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix},$$

where the 3×3 matrix H represents a projection, encompassing rotation, scaling, skew and perspective. Note the use of homogeneous coordinates, for which the equivalence

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \equiv \begin{bmatrix} x_0/z_0 \\ y_0/z_0 \\ 1 \end{bmatrix}$$

holds. Homogeneous coordinate are returned to Euclidean form simply by dividing with the last element.

Estimating the transformation matrix H accurately is the main topic of this chapter. We study its properties, and then explore registration algorithms. These come in two flavours: those that operate in the spatial domain, and those that use a transformed domain, such as the Fourier, wavelet or log-polar domains. Within the spatial domain, methods can be divided further into categories of *dense* and *sparse* registration. Dense methods generally examine images as a whole to determine geometric relationships between pixels, whereas sparse methods rely on features, such as described in the previous chapter.

To conclude, we briefly discuss photometric registration—determining the differences in exposure between frames—used to equalise differences in input frame exposures.

For further reading, we refer to two review articles, [Bro92] and [ZF03]. In [Bro92], one of the first thorough surveys of registration methods, Brown characterises algorithms based on their choice of four components: a feature space, a search space, a search strategy and a similarity metric. More recently, [ZF03] gives a thorough overview of recent developments, including the use of mutual information.

5.2 The transformation matrix

The ability of the transformation matrix, H, to represent a projective transformation is made possible by extending the two-dimensional position vector, $[x, y]^T$, to three components. The value of the last dimension may vary, and

defines an equivalence class so that

$$\begin{bmatrix} zx \\ zy \\ z \end{bmatrix} \equiv \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

Due to this property, these are known as homogeneous coordinates.

The form of the transformation matrix H determines the type of geometric transformation represented. For example,

$$H = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \left[\begin{array}{c|c} R & \mathbf{0} \\ \hline \mathbf{0}^T & 1 \end{array}\right]$$

represents a rotation of angle θ. Making use of the "1" in the homogeneous coordinate, we can add translation

$$H = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & t_x \\ \sin(\theta) & \cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} = \left[\begin{array}{c|c} R & \mathbf{t} \\ \hline \mathbf{0}^T & 1 \end{array}\right],$$

and multiply the rotation matrix by s to get scaling:

$$H = \begin{bmatrix} s\cos(\theta) & -s\sin(\theta) & t_x \\ s\sin(\theta) & s\cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} = \left[\begin{array}{c|c} sR & \mathbf{t} \\ \hline \mathbf{0}^T & 1 \end{array}\right].$$

Skewing can be introduced by multiplying the x and y parameters a and b, e.g.,

$$H = \begin{bmatrix} sa\cos(\theta) & -sb\sin(\theta) & t_x \\ sa\sin(\theta) & sb\cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

while perspective is adjusted in the final row,

$$H = \begin{bmatrix} sa\cos(\theta) & -sb\sin(\theta) & t_x \\ sa\sin(\theta) & sb\cos(\theta) & t_y \\ p_0 & p_1 & 1 \end{bmatrix}.$$

In total, then, there are 8 parameters encoded in the H-matrix. Its elements

are

$$H = \begin{bmatrix} H_{00} & H_{01} & H_{02} \\ H_{10} & H_{11} & H_{12} \\ H_{20} & H_{21} & 1 \end{bmatrix}$$

and, since H operates on homogeneous coordinates, it is homogeneous itself (we can always divide H by a constant without changing its function). Linear transformation matrices can be combined. For example, say we want to rotate an image around its centre at (x_c, y_c). We can express that operation as shifting the image upward and to the left, until its centre lies on the origin, rotating the image and then translating it back to its original position. The transformation matrix for this operation is

$$H = H_S^{-1} H_R H_S$$

where

$$H_S = \begin{bmatrix} 1 & 0 & -x_c \\ 0 & 1 & -y_c \\ 0 & 0 & 1 \end{bmatrix}$$

and

$$H_{R=} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The inverse H_S^{-1} has the same function as H_S, except that it translates in the opposite direction. Note that the order of applying transformations matters; each additional transformation must be pre-multiplied with the existing H.

5.3 Sparse registration: Estimating a homography from correspondences

Feature based registration methods are an excellent way to avoid the local maxima encountered during dense registration [CKK+93] (see Section (5.4)). We discuss two popular homography estimation techniques, and show how outliers (data-points that do not fit the registration model well) are handled.

5.3.1 Least-squares estimation

Direct method

We wish to estimate the 8 unknown parameters of the transformation matrix H, based on known point correspondences. The transformation of a source

coordinate, $\mathbf{x}$, in Euclidean form (i.e., normalised so that $z = 1$) to a target coordinate $\mathbf{x}' = H\mathbf{x}$ yields

$$\begin{aligned} x' &= H_{00}x + H_{01}y + H_{02} \\ y' &= H_{10}x + H_{11}y + H_{12} \\ z' &= H_{20}x + H_{21}y + H_{22}. \end{aligned}$$

By also converting $\mathbf{x}'$ to Euclidean form, we can compare the source and target coordinates,

$$\begin{aligned} \frac{x'}{z'} - \frac{H_{00}x + H_{01}y + H_{02}}{H_{20}x + H_{21}y + H_{22}} &= 0 \\ \frac{y'}{z'} - \frac{H_{10}x + H_{11}y + H_{12}}{H_{20}x + H_{21}y + H_{22}} &= 0. \end{aligned}$$

Multiplying by the denominator yields

$$\begin{aligned} \frac{x'}{z'}(H_{20}x + H_{21}y + H_{22}) - H_{00}x - H_{01}y - H_{02} &= 0 \\ \frac{y'}{z'}(H_{20}x + H_{21}y + H_{22}) - H_{10}x - H_{11}y - H_{12} &= 0 \end{aligned}$$

which can also be written as the system

$$A\mathbf{h} = \begin{bmatrix} -x & -y & -1 & 0 & 0 & 0 & \frac{x'x}{z'} & \frac{x'y}{z'} & \frac{x'}{z'} \\ 0 & 0 & 0 & -x & -y & -1 & \frac{y'x}{z'} & \frac{y'y}{z'} & \frac{y'}{z'} \\ & & & & \vdots & & & & \end{bmatrix} \begin{bmatrix} H_{00} \\ H_{01} \\ H_{02} \\ H_{10} \\ H_{11} \\ H_{12} \\ H_{20} \\ H_{21} \\ H_{22} \end{bmatrix} = \mathbf{0}.$$

Each feature correspondence fills two rows of A, so that n point correspondences yields a $2n \times 9$ matrix.

We now have to solve the homogeneous set of linear equations,

$$A\mathbf{h} = \mathbf{0} \quad \mathbf{h} \neq \mathbf{0}.$$

For 4 point correspondences, the solution is the one-dimensional null-space of

A. For more correspondences, we seek the solution to

$$\arg\min_{\|\mathbf{h}\|=1} \|A\mathbf{h}\| = \arg\min_{\|\mathbf{h}\|=1} \mathbf{h}^T A^T A\mathbf{h} = \lambda_{min} \tag{5.1}$$

where λ_{min} is the smallest eigenvalue of $A^T A$. This is easily shown, given the eigenvalues λ_i and corresponding eigenvectors $\mathbf{q}_i$ of $B = A^T A$. For

$$Q = \begin{bmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \dots & \mathbf{q}_n \end{bmatrix} \quad \text{and} \quad D = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix}$$

it is true that $BQ = QD$ or $B = QDQ^T$. Rewriting (5.1) in terms of this factorisation yields

$$\arg\min_{\|\mathbf{h}\|=1} \mathbf{h}^T QDQ^T\mathbf{h} = \arg\min_{\|\mathbf{y}\|=1} \mathbf{y}^T D^T\mathbf{y} = \arg\min_{\|\mathbf{y}\|=1} \lambda_1 y_1^2 + \lambda_2 y_2^2 + \ldots + \lambda_n y_n^2.$$

With $\lambda_i = \lambda_{min}$, a minimum is achieved when all components of $\mathbf{y}$ are set to zero except for $y_i = 1$. Since $\mathbf{y} = Q^T\mathbf{h}$, we find that $\mathbf{h} = Q\mathbf{y} = \mathbf{q}_{min}$, the eigenvector of B that corresponds to its smallest eigenvalue.

To find this eigenvector, examine the structure of the singular value decomposition (SVD),

$$A = U\Sigma V^T,$$

where the columns of U contain the eigenvectors of AA^T and the columns of V the eigenvectors of $A^T A$ corresponding to the singular values of A on the diagonal of Σ. Recall that we are interested in the eigenvectors of $A^T A$ since the vector $\mathbf{h}$ we are solving for is one such an eigenvector with its eigenvalue closest to zero.

The SVD can be (and usually is) computed so that the singular values appear in decreasing order on the diagonal of Σ. Note that the eigenvalues of the normal matrix $A^T A$ are the squares of the singular values of H. If the system is exactly determined (i.e., with 4 point correspondences), there will be exactly one zero singular value in the last position, which corresponds to the last column of V—our solution. For an over-determined system, we choose the solution as the last column of V, corresponding to the smallest eigenvalue of $A^T A$.

Often, when the skew and perspective distortion is small, we can limit H to

an affine transformation,

$$H = \begin{bmatrix} H_{00} & H_{01} & H_{02} \\ H_{01} & H_{11} & H_{12} \\ 0 & 0 & 1 \end{bmatrix},$$

where we only need to solve for 6 unknown parameters.

Software

The **least-squares solution** to the **projection matrix coefficients** can be calculated using `supreme.register.sparse` with the keyword argument `mode='direct'`.

Iterative method

In the following discussion, denote the Euclidean form of the coordinate $\mathbf{x}$ as $\mathbf{x}_e$, so that

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \text{and} \quad \mathbf{x}_e = \begin{bmatrix} x/z \\ y/z \end{bmatrix}.$$

In the previous section, we set out to find the transformation matrix, H, so that $\mathbf{x}'_e$ and $(H\mathbf{x})_e$ lie close to one another over all feature correspondences, and find the least squares solution that minimises the forward error squared, $\|\mathbf{x}'_e - (H\mathbf{x})_e\|_2^2$ (it is called the forward error because coordinates are transformed "forward", using $H\mathbf{x}$, and then compared to the target coordinates $\mathbf{x}'$). Sometimes, it is useful to minimise the error in both directions, yielding the error function

$$\left\|\mathbf{x}'_e - (H\mathbf{x})_e\right\|_2^2 + \left\|\mathbf{x}_e - (H^{-1}\mathbf{x}')_e\right\|_2^2.$$

This minimisation is achieved using any quadratic minimisation algorithm.

A third method is to minimise the reprojection error, as discussed in [HZ04], whereby the coordinates themselves are included in the parametrisation of the minimisation.

Software

Homography estimation is provided as `supreme.register.sparse`. The open source package "homest" (`http://www.ics.forth.gr/~lourakis/homest/`) provides similar functionality.

```
# x0 is a list of x-coordinates in the first image
# y0 is a list of y-coordinates in the first image
# x1 is a list of x-coordinates in the second image
# y0 is a list of y-coordinates in the second image

# We would like to find the matrix, H, such that
#   [x0]   [x1]
# H [y0] = [y1]
#   [1 ]   [1 ]

import supreme.register as sr

# The mode parameter can be 'direct' or 'iterative'
H = sr.sparse(y0, x0, y1, x1, mode='iterative')
```

5.3.2 Estimation in the presence of outliers

Sadly, no feature correspondence algorithm is perfect, consequently we have to deal with a number of incorrect correspondences amongs those provided. By far the most common approach is Random Sample Consensus, or RANSAC [FB80].

RANdom SAmple Consensus (RANSAC)

In fact, RANSAC is so popular that, in 2006 for the 25th anniversary of the algorithm, a special session was held at the International Conference on Computer Vision and Pattern Recognition to discuss all the variations that had been proposed.

Probably the best known extension is MLESAC [TZ00], but recently Chum [CM05] published a number of improvements including LO-RANSAC and PROSAC. A review of many extensions is available in [CKY97].

Given noisy data to fit to some model (e.g., a straight line, a linear transformation, or anything else), the sample-verify steps that form the RANSAC algorithm are outlined in Algorithm 5.1.

Algorithm 5.1 The RANdom SAmple Consensus (RANSAC) algorithm.

RANSAC aims to find the best parameters, $\mathbf{p}$, for a model M, such that the number of outliers on a data-set X,

$$\sum_i [M(\mathbf{x_i}|\mathbf{p}) \geq K]$$

is minimised (K is a user set threshold). For example, the straight line model has the form

$$M(x, y|m, c) = y - mx - c.$$

Repeat, for a pre-determined number of times (see [FB80]), the following procedure:

1. Randomly draw, without replacement (i.e., without drawing any point twice), N samples from the noisy data-set. Here, N is the minimum number of points required to fit the model, M (e.g., 2 in the case of a straight line).

2. Estimate the model parameters, $\mathbf{p}$, that best fit the N randomly drawn samples.

3. Based on the model parameters, determine the number of data-points (from the entire data-set) that qualify as inliers (i.e., those data points that fit the model well):

$$Q = \sum_i [M(\mathbf{x_i}|\mathbf{p}) < K]$$

   If the number of inliers exceeds those found in previous runs, store the current model parameters.

After termination, the model parameters associated with the largest set of inliers are available.

MSAC: It is noted in [TZ00] that the RANSAC minimises an error of the form

$$C = \sum_i \rho(e_i)$$

where

$$\rho\left(e_i\right) = \begin{cases} 0 & e_i < T \\ 1 & e_i \geq T \end{cases}.$$

Thus, the cost function counts the number of outliers, but does not take the model error of each sample, e_i, into account. We can modify step 3 of RANSAC to read:

Based on the model parameters, evaluate the cost function

$$C = \sum_i \rho\left(e_i\right)$$

where

$$\rho(e_i) = \begin{cases} e_i & e_i < T \\ 1 & e_i \geq T \end{cases}$$

and

$$e_i = M(\mathbf{x}_i|\mathbf{p}).$$

If the value of the error function, C, is lower than in previous runs, store the current model parameters.

This approach is known as m-estimator sample consensus, or MSAC.

LO-RANSAC: In [CM05] it is suggested to execute an inner loop of RANSAC on the inliers every time a new minimum error is obtained. The last sentence of the MSAC description above then becomes

If the value of the error function, C, is lower than in previous runs, proceed to do another RANSAC run, this time only sampling from the best set of inliers, but still verifying against the entire data-set.

In his dissertation, Chum showed that this leads to quicker convergence at a small increase in computational cost.

Early termination Fischler & Bolles [FB80] estimate the number of iterations required to draw an inlier-only set to be

$$k = \epsilon^{-M}$$

where ϵ is the expected probability of choosing an inlier, and M is the number of samples drawn. Since the value of ϵ is usually unknown, we set it conservatively low, increasing the number of iterations and (hopefully) ensuring a good match. The number of iterations is therefore unnecessarily high, but can be updated during each run as we learn more about our data set [Cap05]. For example, given that a run has encountered N inliers in a size L dataset, we set

$$k = \left(\frac{N}{L}\right)^{-M}$$

if the new k is smaller than the existing value but larger than some predefined minimum.

Software

The MSAC variant of **RANSAC**, with the local-optimisation step of LO-RANSAC as well as an early-exit strategy suggested by in [Cap05], is implemented in the module supreme.feature.ransac. The following example estimates the parameters of a straight line, given noisy data-points.

```
from supreme.feature.ransac import RANSAC

class Line:
    ndp = 2 # number of model parameters

    def __init__(self, m, c):
        self.m = m
        self.c = c

    def __call__(self, data, confidence=0.8):
        """Calculate how well the given data matches
        the model.  Return the error for each data-
        point, as well as whether it is an inlier.

        The method used below to determine whether
        a point is an inlier is just a heuristic
        for illustrative purposes.

        """
        # `data` consists of two columns: x, y
        x = data[:, 0]
        y = data[:, 1]
        err = np.abs(y - self.m * x - self.c)
        return err, err < (1 - confidence) * self.m

    def estimate(self, data):
        """Given data, estimate the line parameters.

        """
        x = data[:, 0]
        y = data[:, 1]
        m, c = np.polyfit(x, y, deg=1)
        return m, c

# Assuming we have noisy data, an Mx2 array
m_est, c_est = RANSAC(model=Line(3, 4))(data)
```

Estimating the transformation matrix

The combined MSAC, LO-RANSAC and early-termination variant of RANSAC is used in combination with the direct transformation matrix estimation method discussed earlier. Applied to the putative point-correspondences shown in Fig. 5.1, the transformation shown in Fig. 5.2 is obtained.

Graph Matching

In [TKR08], feature correspondence is formulated as a graph matching problem, which is NP-hard. Several optimisations are suggested to improve execution speed, and, for standard problems, the authors claim to achieve optimal matches within reasonable time limits (minutes, not hours).

5.4 Dense registration methods

In smooth or low-resolution images, it is hard to locate features, so the methods developed in the previous chapter and sections cannot be used. Instead, we opt to match entire images based on their pixel content. This approach is called dense registration.

Next, we discuss common registration procedures, and then mention a popular similarity measure called mutual information.

5.4.1 Error minimisation

Again, we are faced with the problem of solving the unknown geometric transformation T between two images such that

$$f_0(x, y) = T(f_1(x, y)).$$

This time, however, we have no features to assist us. Fortunately, since computers are hard workers, we can solve the problem using brute force by letting the algorithm guess T-functions until it arrives at a good answer. There are two pieces of information missing from this approach: one, how to know when a "good" estimate has been made and, two, which T-function to guess next.

A sufficient measure of the likeness between two images is the mean square error,

$$\|\mathbf{f_0} - T(\mathbf{f_1})\|_2^2 .$$

We say "sufficient" because, as we mentioned in Chapter 7, the L-norms are not ideal in some ways—also the reason why we suggest mutual information in Section 5.4.3.

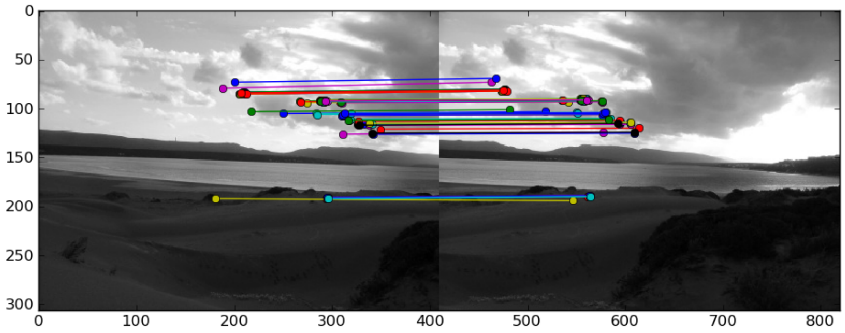


Figure 5.1: *Putative feature matches for the panorama in Fig. 5.2.*

Figure 5.2: *Panoramic stitch of two photos. Features are extracted, using the discrete pulse transform, whereafter RANSAC is used to find a homography that fits inliers. The images are warped accordingly (top, left and right) and blended according to a Laplacian pyramid scheme (bottom) (we use the Enblend package; the algorithm is detailed at* `http://enblend.sourceforge.net/details.htm`*). Photos taken at location 34°23'52.00"S, 20°50'52.09"E.*

As in the previous section, we can model the transformation function as a 3×3 matrix operating on homogeneous coordinates. We need to determine 8 parameters for a full projective transformation, or 6 for an affinity. Given an estimate of those parameters, $\mathbf{h}$, we rewrite the error between our two images as

$$\epsilon(\mathbf{f_0}, \mathbf{f_1}|\mathbf{h}) = \|\mathbf{f0} - T(\mathbf{f_1}|\mathbf{h})\|_2^2 . \tag{5.2}$$

A standard non-linear optimiser, such as the Levenberg-Marquardt algorithm (see [Van05] for a detailed overview), can then be used to estimate the parameters $\mathbf{h}$. Since the derivative of the error function is often hard to derive, Powell's derivative free methods [Pow07, Pow06] may be more suitable.

The interpolation method employed to calculate $T(\mathbf{f_1})$ is important. Lower-order functions, such as bilinear interpolation may over-smooth the image, whereas certain higher order functions could introduce ringing or edge artefacts. In [TU00], the third-order B-spline is suggested as a good compromise. Note that smoothing, in the context of image registration, is especially harmful, since it further flattens the error function, which may not have a prominent minimum to start with.

Software

The Levenberg-Marquardt non-linear minimisation algorithm is available as `scipy.optimize.leastsq`. A modified version of one of Powell's derivative free methods is available as `scipy.optimize.fmin_powell`.

5.4.2 Pyramidal methods

The error function minimised in Section 5.4.1 may (and often does) have several local minima [CKK+93]. Methods employing random jumps in search space, such as simulated annealing, may be able to find a global minimum, but a simpler idea is suggested by Anandan *et al.* [Ana89, BAHH92]

The images to be registered are downsampled by 2^k for $k = 0 \ldots N-1$ to form a pyramidal hierarchy. Starting from the lowest resolution level, the images are aligned using the procedure outlined above. The next level is then registered, using the solution from the previous level as starting parameters. The pyramidal approach avoids local minima, which disappear when the error function is measured over smoothed images.

Alternative formulations include different downsampling factors, q^k for $k = 0, \ldots, N-1$, or including an upsampling step, $k = -1, \ldots, N-1$. The upsampling step aims to facilitate sub-pixel registration accuracy. For this step to be effective, a higher-order interpolation function that preserves detail is required.

5.4.3 Mutual information

Viola first applied the concept of mutual information to image registration [Vio95], whereafter Thevenaz and Unser refined its application [TU96]. The mutual information between two random variables, X and Y, is defined as

$$I(X,Y) = H(Y) - H(Y|X)$$

where $H(Y)$ is its entropy,

$$\begin{aligned} H(X) &= -\mathsf{E}_x \left[\log P(X)\right] \\ &= -\sum_x (\log P(x)) P(x). \end{aligned} \tag{5.3}$$

The mutual information describes the reduction in entropy of Y given X. Already, we can see how it is applicable to image registration: if, when we witness an image X, we suddenly know a lot more about Y, then the two images must be intimately related.

We cannot calculate the quantity $H(Y|X)$ easily, so, following Viola, we rewrite the mutual information as

$$I(X,Y) = H(X) + H(Y) - H(X,Y).$$

The quantity $H(X,Y)$ can be computed from the joint-histogram of X and Y using (5.3). Since the joint-histogram may be sparsely populated, we can convolve it with a discretised Parzen window to get a better approximation of the unpopulated values.

Mutual information is employed instead of the mean square error in the minimisation of (5.2), using a derivative-free method by Powell. Note that the optimisation can be executed more efficiently by employing the mutual information derivatives found in [TU00].

Software

The **joint histogram** (with an adjustable Parzen smoother) is implemented as `supreme.register.joint_hist`. Based on the the result, mutual information can be computed using `supreme.register.mutual_info`. Registration via mutual information is done using `supreme.register.dense_MI`.

5.4.4 Log-polar registration

Log-polar registration [ZW00] is based on the observation that, under certain coordinate transformations, rotations and scalings become translations. Specifically, this property is observed in the transformed image

$$f(\log(R), \theta)$$

where (R, θ) are polar coordinates. Scaling the image by a factor s yields

$$f(\log(R) + \log(s), \theta)$$

whereas rotation by an angle ϕ results in

$$f(\log(R), \theta + \phi).$$

To construct the log-polar transform, we first convert cartesian coordinates to polar coordinates,

$$\begin{aligned} R &= \sqrt{(x - x_c)^2 + (y - y_c)^2} \\ \theta &= \begin{cases} \arctan\left(\frac{y - y_c}{x - x_c}\right) & x - x_c \neq 0 \\ \pi/2 & x - x_c = 0 \end{cases} \end{aligned}$$

where (x_c, y_c) are the centre coordinates. The log axis of the log-polar transform becomes

$$L = \log_b R.$$

Denote the height or width of the input image, whichever is largest, by w. We would like the log-axis of the transformed image to have the same dimension. The maximum value of R occurs at the corner points, where $R = R_{max} = \sqrt{x_c^2 + y_c^2}$. Setting $L = \log_b R = w$, we calculate b as

$$b = e^{\ln(R_{max})/w} = R_{max}^{1/w}.$$

Warping in reverse When warping images, it is not practical to use this "forward" transform. Some cartesian coordinates may, under integer roundoff, map to the same log-polar position, and, worse, some log-polar positions may have no integer cartesian counterparts, leaving empty patches in the transformed image. Instead, we take each coordinate in log-polar space, calculate its (non-integer, floating point) position in cartesian space, and interpolate the image to

obtain its value.

Given L and θ, we therefore want to calculate x and y. First, we compute R as

$$R = e^{\ln(b)L} = e^{L\ln(d)/w}$$

after which x and y is recovered as

$$\begin{aligned} x &= R\cos(\theta) + x_c \\ y &= R\sin(\theta) + y_c. \end{aligned}$$

Registration using the log-polar transform

As shown in Figure 5.3, changes in rotation and scale cause shifts in the log-polar domain. To detect these shifts, we calculate the correlation between two transformed images (either traditional correlation or phase correlation). The position of the correlation peak, $(\Delta x, \Delta y)$, gives the scale and rotation as

$$\begin{aligned} s &= e^{b\Delta x} \\ \phi &= \Delta y, \end{aligned}$$

assuming the transform was performed over 360°. Note, then, that the log polar transform gives 4 of the 6 parameters required for affine registration—the 2 translation parameters must be fixed beforehand.

Extensions

Computing feature correspondence The log-polar transform lies on the boundary of feature-based and dense registration methods. In [VH07], we suggest a modification of the algorithm suggested in [ZW00] to place it firmly in the feature-based category:

- Filter the input image to detect regions of interest.
- In each region, pick a point of interest that responded most strongly to the filter.
- Compare all points of interest in the source image to those in the target image.
- Use the strongest correlation to determine registration parameters.

The log-polar transform has two drawbacks. First, it needs to be performed on fairly large image patches to be of use, and, second, the transform is more expensive to calculate than the statistical quantile-quantile method proposed in

Chapter 4. The adaptive polar transform discussed next claims to address these concerns, although we have not verified that it does so in practice.

The adaptive polar transform (APT) The log polar transform non-uniformly samples the source image, causing oversampling near the fovea and undersampling in the outer regions. The adaptive polar transform [MZE09] aims to address this problem. Due to the uniform sampling employed, a straightforward correlation can no longer be used to register images. Instead, the APT of the each image is projected to two one-dimensional signals—one representing scale, the other rotation angle. By correlating these image projections over different frames, the registration parameters are recovered. The authors of [MZE09] claim that this can be done with fewer operations than the traditional log polar transform. They also describe a feature search strategy, similar in principle to the one given above in "*Computing feature correspondence*".

Software

The **log polar transform** is implemented as `supreme.transform.logpolar`. Registration parameters are calculated using `supreme.register.lp_patch_match`.

5.5 Photometric registration

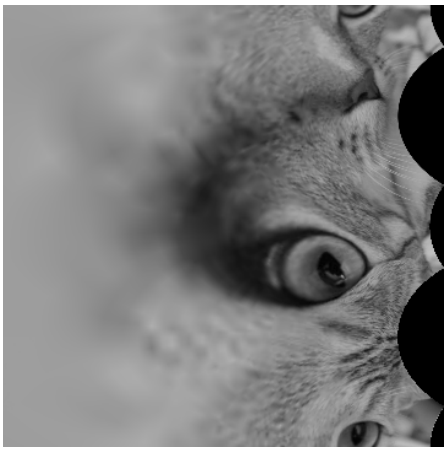
Almost all digital cameras provide functionality to ensure properly exposed photographs. This entails adjusting parameters such as the sensor gain, aperture size and exposure time. Furthermore, scene radiance is modified by the non-linear camera response function, whereby high intensity values are compressed and low intensity values are expanded.

When performing super-resolution imaging, we relate each pixel to its counterpart in other input frames. As such, we prefer a specific point in the scene to have equal pixel intensity values over all frames.

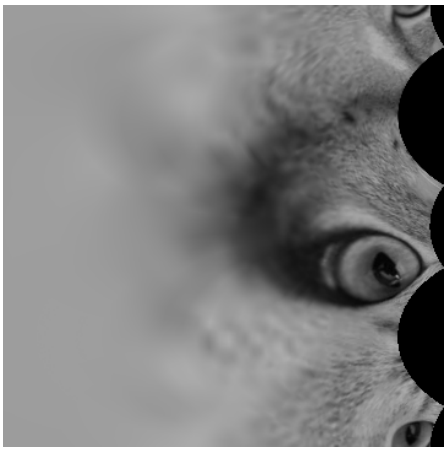
Estimating the parameters of these processes is known as photometric registration. We can loosely distinguish between relative and absolute photometric registration, as used in astronomy [PSF+08]. In absolute registration, the goal of is to determine the true flux incident on the CCD. In relative registration, we merely wish to establish this quantity relative to some known reference (such as the exposure in a chosen frame).

For super-resolution, we aim to establish a crude photometric relationship between input frames, and luckily the accuracies needed are not anything near those required in astronomy. Given a reference and a source input frame, $\mathbf{f_R}$

(a) Input image, *Chelsea the Cat.*



(b) Log-polar transform (LPT) of input image.



(c) LPT after scaling by 2.

(d) LPT after rotating by 30°.

(e) LPT after scaling by 2 and rotating by 30°.

Figure 5.3: *The log-polar transform. Note how changes in scale and rotation cause translation in the transform domain.*

and $\mathbf{f_S}$ respectively, the problem is simply modeled as

$$\mathbf{f_R} = q(\mathbf{f_S}) \tag{5.4}$$

where q is a non-linear function that maps intensities in $\mathbf{f_S}$ to match those in $\mathbf{f_R}$.

The typical camera response function can be modelled as a power-law expression,

$$g(r|\gamma) = r^{\gamma},$$

where r is the scene radiance (a value in $[0, 1]$) and γ is a shape parameter. When the camera shutter remains open for longer, or when the camera gain is adjusted, the function becomes

$$g(r|\gamma, s) = (rs)^{\gamma}$$

where s represents the gain resulting from increased shutter time or gain adjustment. After registration, the measured images are

$$\begin{aligned} \mathbf{f_R} &= g(r|\gamma, 1) = r^{\gamma} \\ \mathbf{f_S} &= g(r|\gamma, s) = r^{\gamma} s^{\gamma}. \end{aligned}$$

Combining the above response functions and adding an offset parameter, b, we rewrite (5.4) as

$$\mathbf{f_R} = q(\mathbf{f_S}) = a\mathbf{f_S} + b \tag{5.5}$$

where a is an intensity multiplier (replacing s^{γ}). In [Cap01], MLESAC (a variant of RANSAC) is used to estimate these parameters, but an easier route is to observe how the mean, μ_S, and variance, σ_S, of $\mathbf{f_S}$ are modified by (5.5) [HW79]. For example, μ_S and μ_R are now related by

$$\mu_R = a\mu_S + b,$$

while σ_R, the standard deviation of $\mathbf{f_R}$, becomes

$$\sigma_R = a\sigma_S.$$

The parameters are now given by

$$\begin{aligned} a &= \sigma_R/\sigma_S \\ b &= (\mu_R\sigma_S - \mu_S\sigma_R)/\sigma_S. \end{aligned}$$

The means and variances are calculated using those entries where both images have intensity values in $[10, 200]$ (for 255 level greyscale images). This is done to avoid clipping at high intensities, noise at low intensities and zeros introduced by out-of-boundary values during warping.

Software

Photometric registration is implemented as `supreme.photometry.photometric_adjust`. To modify an image, `source`, to look like the image `target`, use:

```
from supreme.photometry import photometric_adjust

a, b = photometric_adjust(source, target)
source = source * a + b
```

Histogram Adjustment The above procedure relies on the camera response function being r^γ. If we are unsure of this modelling, we may instead estimate p using histogram matching. One of the first uses of histogram matching on multi-sensor data is that of Horn and Woodham in their 1979 paper on removing stripes from LANDSAT imagery [HW79].

We assume that the radiance distribution of two registered images is the same, at least in the overlapping region. The function q is known to be monotone increasing (i.e., $q(x_0) < q(x_1)$ if $x_0 < x_1$), so that the standard method for transformation of a random variable can be applied.

Given values $x \in \mathbf{f_S}$ and corresponding $y \in \mathbf{f_R}$, we seek

$$y = q(x).$$

This introduces a relationship between the distribution functions (the cumulative probability density functions),

$$P_X(x) = P_Y(y).$$

The function $q(x)$ is determined as

$$y = q(x) = P_Y^{-1}(P_X(x))$$

where P_Y^{-1} is the inverse of P_Y. Since both the distribution functions, P_X and P_Y, can be measured from $\mathbf{f_S}$ and $\mathbf{f_R}$, calculating $q(x)$ should pose no problem. The only hurdle is that, instead of continuous distributions, P_X and P_Y are

discrete, so that

$$P_Y(y) = \sum_{\forall t: t \leq y} p(t)$$

where p is the histogram of $y \in \mathbf{f_R}$. The inverse, $P_Y^{-1}(z)$, can then be formulated as follows:

> *Calculate the value y for which the discrete distribution function $P_Y(y)$ is closest to z.*

Since y can only be one of several discrete values, the function maps a continuous z to a discrete y. For our specific application, the calculation of the histogram neglects zero-values, since those are introduced during warping when out-of-boundary coordinates occur. Results are shown in Figure 5.4.

Software

Histogram matching is implemented as `supreme.photometry.histogram_adjust`. To modify an image, `source`, to look like the image `target`, use:

```
from supreme.photometry import histogram_adjust

q = histogram_adjust(source, target)
source = q(source)
```

Photometric registration for super-resolution

Super-resolution input frames are often small, cropped areas of a larger scene. As such, they may contain few grey-levels, resulting in a poor density function estimate when applying the histogram method. The discretisation of function $q(x)$ furthermore causes unacceptable quantisation errors. It may be possible to approximate the distribution functions locally and to calculate a more exact inverse, but we have not investigated this possibility further. Until we do, we recommend the affine model, $a\mathbf{x} + b$, which is linear, robust, fast and parametrised by only two variables. Other interesting research include estimation of the camera response function from a single image [NCT07].

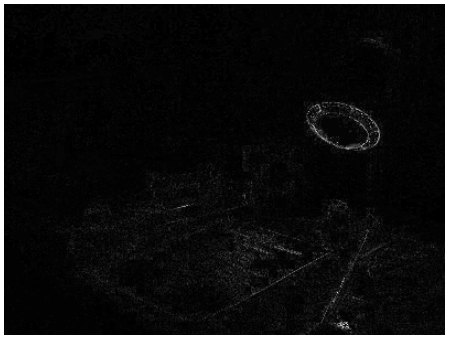
(a) Reference frame.

(b) Source frame.

(c) Source frame, photometrically adjusted to fit reference using an affine intensity transformation.

(d) Absolute difference between reference frame and affine adjusted source frame. Note that over-exposed areas could not be corrected.

(e) Source frame, photometrically adjusted to fit reference using histogram matching.



(f) Difference between reference frame and histogram adjusted source frame.

Figure 5.4: *Minimising exposure difference using histogram matching. These photographs are from [vWNA06].*

Bibliography

[Ana89] P. Anandan. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2(3):283–310, January 1989.

[BAHH92] James R. Bergen, P. Anandan, Keith J. Hanna, and Rajesh Hingorani. Hierarchical Model-Based Motion Estimation. *Lecture Notes In Computer Science; Vol. 588*, 1992.

[Bro92] L.G. Brown. A Survey of Image Registration Techniques. *ACM computing surveys (CSUR)*, 24(4):376, 1992.

[Cap01] David Peter Capel. *Image Mosaicing and Super-resolution*. Ph.D. dissertation, University of Oxford, 2001.

[Cap05] D. Capel. An effective bail-out test for RANSAC consensus scoring. In *Proc. BMVC*, pages 629–638, 2005.

[CB09] Guilherme Holsbach Costa and José Carlos M. Bermudez. Registration Errors: Are They Always Bad for Super-Resolution? *IEEE Transactions on Signal Processing*, 57(10):3815–3826, October 2009.

[CKK+93] P. Cheeseman, B. Kanefsky, R. Kraft, J. Stutz, and R. Hanson. Super-resolved surface reconstruction from multiple images. In G. R. Heidbreder, editor, *Proceedings of the Thirteenth International Workshop on Maximum Entropy and Bayesian Methods*. Kluwer Academic, 1993.

[CKY97] S. Choi, T. Kim, and W. Yu. Performance Evaluation of RANSAC Family. *Journal of Computer Vision*, 24(3):271–300, 1997.

[CM05] O. Chum and J. Matas. Matching with PROSAC — Progressive Sample Consensus. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, (I):220–226, 2005.

[FB80] 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), 1980.

[HW79] B. Horn and R. Woodham. Destriping LANDSAT MSS images by histogram modification. *Computer Graphics and Image Processing*, 10(1):69–83, May 1979.

[HZ04] R.I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[MZE09] Rittavee Matungka, Yuan F. Zheng, and Robert L. Ewing. Image registration using adaptive polar transform. *IEEE Transactions on Image Processing*, 18(10):2340–54, October 2009.

[NCT07] T.T. Ng, S.F. Chang, and M.P. Tsui. Using geometry invariants for camera response function estimation. In *IEEE CVPR*, June 2007.

[Pow06] M.J.D. Powell. The NEWUOA software for unconstrained optimization without derivatives. *Nonconvex Optimization and its Applications*, 83:255, 2006.

[Pow07] M.J.D. Powell. A view of algorithms for optimization without derivatives. *Mathematics Today-Bulletin of the Institute of Mathematics and its Applications*, 43(5):170–174, 2007.

[PSF+08] N. Padmanabhan, D.J. Schlegel, D.P. Finkbeiner, J.C. Barentine, M.R. Blanton, H.J. Brewington, J.E. Gunn, M. Harvanek, D.W. Hogg, Ž. Ivezić, et al. An Improved Photometric Calibration of the Sloan Digital Sky Survey Imaging Data. *The Astrophysical Journal*, 674:1217–1233, 2008.

[TKR08] Lorenzo Torresani, Vladimir Kolmogorov, and Carsten Rother. Feature Correspondence via Graph Matching: Models and Global Optimization. In *Computer Vision - ECCV 2008*, volume 5303 of *Lecture Notes in Computer Science*, pages 596–609. Springer, Berlin, Heidelberg, 2008.

[TU96] P. Thevenaz and M. Unser. A pyramid approach to sub-pixel image fusion based on mutual information. *Proceedings of 3rd IEEE International Conference on Image Processing*, I:265–268, 1996.

[TU00] P. Thevenaz and M. Unser. Optimization of mutual information for multiresolution image registration. *IEEE Transactions on Image Processing*, 9(12):2083–99, 2000.

[TZ00] P.H.S. Torr and A. Zisserman. MLESAC: A New Robust Estimator with Application to Estimating Image Geometry. *Computer Vision and Image Understanding*, 78(1):138–156, April 2000.

[Van05] S.J. Van Der Walt. Automated stratigraphic classification and feature detection from images of borehole cores. Msc, Stellenbosch University, 2005.

[VH07] S.J. Van Der Walt and B.M. Herbst. Methods Used in Increased Resolution Processing: Polygon based interpolation and robust log-polar based registration. In *Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP) 2007*, Barcelona, Spain, 2007.

[Vio95] P. Viola. *Alignment by Maximization of Mutual Information*. Ph.D. dissertation, Massachusetts Institute of Technology, 1995.

[vWNA06] M. Čadík, M. Wimmer, L. Neumann, and A. Artusi. Image attributes and quality for evaluation of tone mapping operators. *National Taiwan University*, pages 35–44, 2006.

[ZF03] B. Zitova and J. Flusser. Image registration methods: a survey. *Image and Vision Computing*, 21(11):977–1000, 2003.

[ZW00] Siavash Zokai and George Wolberg. Robust image registration using log-polar transform. In *Proceedings of the IEEE International Conference on Image Processing, Canada*, 2000.

Chapter 6

Super-resolution image processing

6.1 Introduction

The general super-resolution problem can be stated as follows: *Given a number of digital photos of an object, can we combine these images to form a new image with increased detail?* To answer this question, we need additional information:

- Were the images degraded by factors such as motion blur, excessive sensor noise or lighting changes? (Pertains to *image formation)*
- What is the relative position of the camera and the object? (Pertains to *image formation* and *registration*)
- Did the camera or the object move, and how? (Pertains to *registration*)
- How many photos of what sizes are available? (Pertains to *registration* and *reconstruction*)

Image formation (see Chapter 2) is the process whereby light, reflected from a scene, travels through an optical system and causes accumulation of charge in a photosensitive sensor element. The charge values are read out, amplified, discretised and possibly processed before being stored as image intensity values.

Super-resolution relies on slight shifts in camera (or object) positions between several input frames to provide high-frequency information lost during sampling. Before reconstruction can take place, images must be aligned or **registered** (i.e., the effect of camera motion must be negated), preferably to sub-pixel accuracy (see Chapter 5).

Once images are accurately aligned, one of several **reconstruction** processes can be applied to restore high-frequency detail.

Image acquisition model

The image acquisition process is often represented as the simplified model

$$\mathbf{g} = S\downarrow(h(\mathcal{T}(\mathbf{f}))) + \boldsymbol{\eta}$$

where

- $\mathbf{g}$ is the resulting low-resolution (LR) digital image (also referred to as the frame or photograph),
- $\mathbf{f}$ is a high-resolution (HR) representation of the scene,
- $\mathcal{T}$ is a geometric transformation dependent on camera position,
- h is the camera point-spread function,
- $S\downarrow$ is the downsampling operator and
- $\boldsymbol{\eta}$ is normally distributed noise.

This model makes the following assumptions, among others:

- The LR frame can be reproduced from the HR frame. In reality, the camera generates images based on the scene radiance instead, so the assumption holds only if the HR image is a fairly good representation of the true scene radiance.
- No non-linear noise sources are present, and the additive noise is Gaussian. This assumption is explored in Chapter 2.

Estimating the HR image from a set of LR images only becomes a tractable problem once further assumptions are made. For example, the model is often linearised as

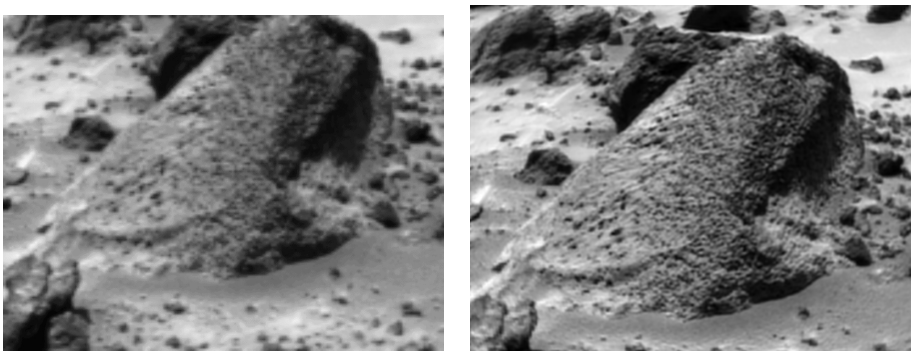
$$\mathbf{g} = A\mathbf{f} + \boldsymbol{\eta},$$

which yields the solutions presented in Chapter 7.

This chapter presents the original super-resolution paper [CKK+93], where super-resolution is posed as a maximum a posteriori (MAP) estimate of the high-resolution image, whereafter novel heuristic methods are listed.

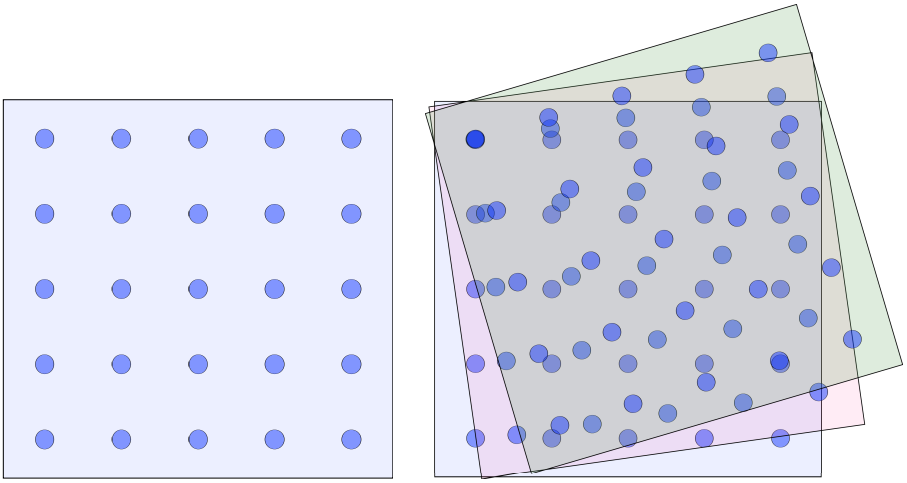
6.2 The dawn of super-resolution

Super-resolution as we know it was introduced in a 1993 paper by NASA Ames's Cheeseman, Kanefsky, Hanson and Stutz [CKK+93]. One of their results, based on data from the Mars Pathfinder mission, is shown in Figure 6.1. Even today,



(a) Enlarged input image, one of 25. **(b)** Super-resolved image.

Figure 6.1: *A high-resolution image of a rock named "Wedge", calculated using the first super-resolution algorithm. The technique, developed by NASA Ames's Cheeseman, Kanefsky, Hanson, and Stutz [CKK+93], here combines 25 images taken as part of NASA's Pathfinder mission.*



(a) An image, obtained by sampling on a grid (the blue dots). **(b)** An overlay of multiple, rotated images, each sampled on a grid.

Figure 6.2: *Sampling a signal multiple times may lead to an increase in sampling rate. In this case, notice how a denser sampling is obtained in the overlapping image regions.*

the best algorithms still seek a maximum a posteriori estimate much like they did. One of the first papers on MAP image restoration was [Bes86], while another MAP approach to super-resolution was published in [HC90]. What follows is an overview of [CKK+93].

6.2.1 Why is super-resolution possible?

Imagine having a device that samples a 1Hz signal once every second. Because the sampling rate is slower than the Nyquist rate, a perfect reconstruction is not possible. Making use of another two identical devices can be beneficial: each is activated 0.3 seconds after the previous, and, by combining the resulting data, a sampling rate of 3Hz is achieved.

We use a similar experimental setup for super-resolution imaging; the sampling device (a camera), while moving slightly, samples the scene radiance[1] by taking several photos. Unlike the first experiment, we do not have control over the sampling "delay" (the relative movement of the camera), resulting in a dense but irregular sampling (see Fig. 6.2).

During super-resolution post-processing, the photographs taken (called low-resolution or LR frames) are combined to form an estimate of the scene radiance (called the high-resolution or HR frame). The high-resolution frame itself is a sampled version of the scene radiance, albeit at higher resolution.

6.2.2 Maximum a posteriori estimate

Given a number of low-resolution frames, concatenated to form the vector $\mathbf{b}$, and the accompanying camera parameters, $\mathbf{c}$, the super-resolution problem is the estimation of a high-resolution image, $\mathbf{x}$, such that the posterior

$$P(\mathbf{x}|\mathbf{b},\mathbf{c})$$

is maximised. Finding a high-resolution image from a number of low-resolution image is no easy task; we prefer to rewrite the maximisation, making use of Bayes's theorem, as

$$P(\mathbf{x}|\mathbf{b},\mathbf{c}) = \frac{P(\mathbf{b}|\mathbf{x},\mathbf{c})P(\mathbf{x}|\mathbf{c})}{P(\mathbf{b}|\mathbf{c})}. \tag{6.1}$$

The likelihood $P(\mathbf{b}|\mathbf{x},\mathbf{c})$ is easier to maximise, since it inverts the problem and asks the question: "Given a high-resolution image and camera parameters, how would the low-resolution images look?". The denominator is not important in the maximisation, since it is independent of $\mathbf{x}$.

If the prior $P(\mathbf{x}|\mathbf{c})$ is set to a constant, the problem reduces to the maximum likelihood estimator. We can think of the maximum a posteriori estimator as the maximum-likelihood (ML) estimator, regularised by the term $P(\mathbf{x}|\mathbf{c})$.

[1] In reality, we sample image irradiance—the visible-light energy incident on the image sensor. Under some mild assumptions, and especially in narrow-field imaging systems, scene radiance is proportional to image irradiance [AMK07].

6.2.3 Pixel probabilities

Experiments show [Cap01, CKK+93] that the per-pixel probability, $P(b|\mathbf{x},\mathbf{c})$ with $b \in \mathbf{b}$, can be approximated as a Gaussian distribution,

$$P(b|\mathbf{x},\mathbf{c}) = \mathcal{N}(\bar{b},\sigma_b) = \frac{1}{\sqrt{2\pi\sigma_b^2}} e^{-(b-\bar{b})^2/(2\sigma_b^2)}.$$

The $\mathbf{x}$-dependence is through $\mathbf{b} = A\mathbf{x}$, as explained below. The probability has a maximum when $b = \bar{b}$, the "true" pixel value as derived from the scene radiance. If the high resolution image is a good representation of the scene radiance, it can be used to determine $\bar{b}$ accurately. The low-resolution pixel becomes a weighted sum of high-resolution pixels in the same vicinity; the weights are determined by the camera point-spread function while the necessary geometric transformations are described by some of the camera parameters in $\mathbf{c}$.

We assume that pixel perturbations across different low resolution frames are independent—a reasonable assumption, given that there are multiple factors at play (e.g., noise and registration errors), uncorrelated over frames. Furthermore, we assume independence over neighbouring pixels. The assumption is reasonable, because the neighbourhood influence is highly localised. The above distribution can then be vectorised as

$$P(\mathbf{b}|\mathbf{x},\mathbf{c}) = \frac{1}{|2\pi\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{b}-\bar{\mathbf{b}})^T\Sigma^{-1}(\mathbf{b}-\bar{\mathbf{b}})\right),$$

with $\Sigma = \sigma^2 I$ a spherical covariance matrix, k the dimensionality of $\mathbf{b}$ and $\bar{\mathbf{b}}$ the values of the low-resolution pixels, estimated from the high-resolution image $\mathbf{x}$. Since the values in $\bar{\mathbf{b}}$ are modelled to be a linear combinations of those in $\mathbf{x}$, we can write

$$\bar{\mathbf{b}} = A\mathbf{x}.$$

The properties of the matrix A are studied in the next chapter.

It follows that

$$\arg\max_{\mathbf{x}} P(\mathbf{b}|\mathbf{x},\mathbf{c}) = \arg\max_{\mathbf{x}} \left\{-\|\mathbf{b}-A\mathbf{x}\|^2\right\}. \tag{6.2}$$

When the prior is included in (6.1), we have

$$\begin{aligned}\arg\max_{\mathbf{x}} P(\mathbf{x}|\mathbf{b},\mathbf{c}) &= \arg\max_{\mathbf{x}} \frac{P(\mathbf{b}|\mathbf{x},\mathbf{c})P(\mathbf{x}|\mathbf{c})}{P(\mathbf{b}|\mathbf{c})} \\ &= \arg\max_{\mathbf{x}} P(\mathbf{b}|\mathbf{x},\mathbf{c})P(\mathbf{x}|\mathbf{c}).\end{aligned}$$

Taking the log and combining with (6.2) while assuming a spherical covariance

yields

$$\begin{aligned}\arg\max_{\mathbf{x}} P(\mathbf{x}|\mathbf{b},\mathbf{c}) &= \arg\max_{\mathbf{x}} \left[\log P(\mathbf{b}|\mathbf{x},\mathbf{c}) + \log P(\mathbf{x}|\mathbf{c})\right] \\ &= \arg\max_{\mathbf{x}} \left[-\|\mathbf{b}-A\mathbf{x}\|^2 + \lambda \log P(\mathbf{x}|\mathbf{c})\right],\end{aligned}$$

where various constants are absorbed by λ.

When choosing the prior as Gaussian with zero mean and spherical covariance we have

$$\begin{aligned}\arg\max_{\mathbf{x}} P(\mathbf{b}|\mathbf{x},\mathbf{c}) &= \arg\max_{\mathbf{x}} \left[-\|\mathbf{b}-A\mathbf{x}\|^2 - \lambda \mathbf{x}^{\mathbf{T}}\mathbf{x}\right] \\ &= \arg\min_{\mathbf{x}} \left[\|\mathbf{b}-A\mathbf{x}\|^2 + \lambda \mathbf{x}^{\mathbf{T}}\mathbf{x}\right]. \qquad (6.3)\end{aligned}$$

Knowing that our pixel values are not centred around zero, the zero-mean Gaussian distribution does not seem appropriate. In the next chapter, we show how to manipulate our input data to fit this model.

The form of (6.3) is well known as the regularised solution to the least-squares problem $A\mathbf{x} = \mathbf{b}$.

6.2.4 Camera parameters

Our camera parameters, $\mathbf{c}$, are defined as all the known information about the image formation process. This therefore includes registration information, parameters of the point-spread function, and so forth. Usually, the number of registration parameters is very small compared to the data-set; this forms the basis for the following argument.

Is it satisfactory to pre-calculate the camera parameters during registration, or should they be included in the MAP estimation? For example, we could determine the parameters that maximise

$$\arg\max_{\mathbf{x},\mathbf{c}} P(\mathbf{x},\mathbf{c}|\mathbf{b}).$$

This approach, however, involves the difficult process of modelling the joint distribution, as done in [PCRZ07b]. Cheeseman *et al.* motivates why this is unnecessary [CKK+93]: the small number of registration parameters are obtained from a large number of data-points (all image pixels); as such, $\mathbf{c}$ is highly over-determined and can be pre-computed accurately without modelling its distribution.

6.2.5 Boundary effects

In the description above, we model the pixels in the low-resolution vector $\mathbf{b}$ as weighted combinations of those in $\mathbf{x}$, written as

$$\mathbf{b} = A\mathbf{x}.$$

For a single value of $b \in \mathbf{b}$ we write

$$b = \sum_k x_k$$

where k represents all high-resolution pixels in the vicinity of b. However, close to the image boundaries, these pixels could lie *outside* $\mathbf{x}$. To handle the problem, four approaches have been suggested:

1. **Treat values outside the boundary as zero.** This is the approach used in our implementation and in the next chapter. Due to regularisation and the smoothing effect of either bilinear or polygon interpolation (see Chapter 7), boundary effects have little impact on the reconstruction, especially in areas where numerous low-resolution frames overlap.

2. **Perform super-resolution on subimages**, cut from the low-resolution input frames [CKK+93]. This way, boundary values are always available.

3. **Stack all input frames**, and use this as an estimate of pixels outside the boundary [Cap01].

4. **Use traditional boundary extension**, such as mirroring or periodic extension.

6.2.6 The point-spread function

Another important consideration is the weights in A. Usually, A represents two operations: sampling—parameterised by the camera point-spread function (PSF)—followed by downsampling. (Note that, in the next chapter, we describe a parameter free construction of A.)

The camera point-spread function can be measured in a laboratory [CKK+93] or determined experimentally [Cap01].

We find that it can even be estimated during reconstruction, since unsuitable PSFs cause oscillatory behaviour. For example, the PSF is often modelled as a Gaussian kernel. If the kernel is too wide, it oversmooths the solution; to compensate, high-amplitude components are added to the solution during reconstruction in an attempt to reduce $\|A\mathbf{x} - \mathbf{b}\|^2$. If, on the other hand, the

kernel is too narrow, it acts as a high-pass filter, emphasising noise. Minimising the absolute sum of gradients of the solution often yields a good point-spread parameter.

6.2.7 Solving the linear system

It is not always necessary to write super-resolution as a linear problem, but when we do a whole arsenal of tried-and-tested algorithms become available. In [CKK+93], the system is constructed to be square (done by choosing the correct increase in resolution) and solved using Jacobi iteration. It is often more practical to solve the least squares problem, as shown in Chapter 7, where we solve a large, sparse and over-determined linear system using either steepest descent, conjugate-gradients or damped LSQR.

6.3 Other approaches

The techniques that follow are commonly used to improve image quality.

6.3.1 Averaging

Given measurements of the form

$$x_i = \bar{x} + \eta$$

where the actual value, $\bar{x}$, is corrupted by normally distributed noise, $\eta \sim \mathcal{N}(0, \sigma^2)$, we can show that the summed variable

$$y = \frac{1}{N} \sum_{i=0 \ldots N-1} x_i$$

is also distributed normally as

$$y \sim \mathcal{N}\left(\bar{x}, \frac{\sigma^2}{N}\right).$$

This result explains why *image stacking* is commonplace in astronomy: it reduces zero-mean noise significantly. Astronomers are also plagued by "seeing", the effect of the Earth's turbulent atmosphere on the path of light; one way to avoid it is to take a number of short exposures, and to stack only the best frames, a process known as "lucky imaging".

Stein and James [JS61, Ste81] show that, in terms of the overall mean-squared error, a better estimate is given by their biased estimator.

(a) Source image (1 of 60), shape 38×34.

(b) Average of 5 times enlarged input images.

(c) A "super-resolution" result from literature, showing no improvement over the average image.

Figure 6.3: *Zero-mean noise reduction by averaging.*

In the next chapter, we use the average image to regularise our super-resolution estimate.

6.3.2 Map and deblur

This method, published in 1999 [GR99], is the first step in Bannore's recently published "iterative-interpolation super-resolution" [Ban09]. All low-resolution pixels are mapped to a high-resolution reference frame to form a sparse reconstruction (some pixels are not assigned values). The remaining "holes" are filled using interpolation or weighted averaging. Finally, a de-blurring operator is applied to sharpen the high-resolution estimate.

Capel [Cap01] explains the reasoning behind this method, given that a low resolution frame can be represented as

$$g_n = s \downarrow (h * \mathcal{T}_n(f))$$

where f is the high-resolution image, $\mathcal{T}_n$ is a Euclidean geometric transformation, h is an isotropic point spread function, and $s \downarrow$ is a downsampling operator. The order of the the convolution and the transformation operators can be reversed to obtain

$$g_n = s \downarrow (\mathcal{T}_n(h * f)).$$

This process is then inverted by upsampling, removing the transformation and deblurring.

6.3.3 Pan-sharpening

In satellite imagery, the situation arises where two images of a scene are available: a high-resolution, monochromatic (grey-scale) version, and a low-resolution, multispectral (colour) version. The process of "colouring in" the high-resolution, monochromatic image is known as pan-sharpening, a form of image fusion. It reminds strongly of a well-initialised version of the colorisation process suggested in [LWL04]. A review of pan-sharpening techniques is given in [GNA+].

6.3.4 Compressive sampling / compressed sensing

When a camera is under our control, compressive sampling can be applied to reconstruct a dense representation of a scene, based on sparse samples [MW08]. In practice, super-resolution is applied in situations where the camera cannot be influenced (otherwise, simply changing lenses would provide the resolution improvement required).

In [WH08], a single-frame super-resolution algorithm is developed from the perspective of compressed sensing.

Bibliography

[AMK07] A.V. Arecchi, T. Messadi, and R.J. Koshel. *Field Guide to Illumination*. SPIE Press, 2007.

[Ban09] V. Bannore. *Iterative-Interpolation Super-Resolution Image Reconstruction: A Computationally Efficient Technique*. Springer, 2009.

[Bes86] J. Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society. Series B (Methodological)*, 48(3):259–302, 1986.

[Cap01] David Peter Capel. *Image Mosaicing and Super-resolution*. Ph.D. dissertation, University of Oxford, 2001.

[CKK+93] P. Cheeseman, B. Kanefsky, R. Kraft, J. Stutz, and R. Hanson. Super-resolved surface reconstruction from multiple images. In G. R. Heidbreder, editor, *Proceedings of the Thirteenth International Workshop on Maximum Entropy and Bayesian Methods*. Kluwer Academic, 1993.

[GNA+] A. Garzelli, F. Nencini, L. Alparone, B. Aiazzi, and S. Baronti. Pan-sharpening of multispectral images: a critical review and comparison. *IEEE International Geoscience and Remote Sensing Symposium, 2004. IGARSS '04. Proceedings. 2004*, pages 81–84.

[GR99] F. Guichard and L. Rudin. Velocity estimation from images sequence and application to super-resolution. In *Proceedings of the 1999 International Conference on Image Processing*, pages 527–531. IEEE, 1999.

[HC90] Yi-Ping Hung and David B. Cooper. Maximum a-posteriori probability 3-D surface reconstruction using multiple intensity images directly. In Bernd Girod, editor, *Sensing and Reconstruction of Three-Dimensional Objects and Scenes*, volume 1260, pages 36–48, Santa Clara, CA, USA, January 1990. SPIE.

[JS61] W. James and Charles Stein. Estimation with Quadratic Loss. In Jerzy Neyman, editor, *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability*, pages 361–379, Berkeley, California, 1961. University of California Press.

[LWL04] Anat Levin, Yair Weiss, and Dani Lischinski. Colorization using optimization. *ACM Transactions on Graphics*, 23(3):689, 2004.

[MW08] R.F. Marcia and R.M. Willett. Compressive coded aperture superresolution image reconstruction. In *Int. Conf. on Acoustics, Speech and Sig. Proc., ICASSP*, pages 833–836, 2008.

[PCRZ07b] Lyndsey C. Pickup, David P. Capel, Stephen J. Roberts, and Andrew Zisserman. Overcoming Registration Uncertainty in Image Super-Resolution: Maximize or Marginalize? *EURASIP Journal on Advances in Signal Processing*, 2007:1–15, 2007.

[Ste81] C.M. Stein. Estimation of the Mean of a Multivariate Normal Distribution. *The Annals of Statistics*, 9(6):1135–1151, 1981.

[WH08] John Wright and Thomas Huang. Image super-resolution as sparse representation of raw image patches. *2008 IEEE Conference on Computer Vision and Pattern Recognition*, June 2008.

Chapter 7

Super-resolution as a sparse linear problem

7.1 Introduction

In the previous chapter we state that the super-resolution problem is often modelled as

$$A\mathbf{x} = \mathbf{b} + \boldsymbol{\eta}$$

where $\mathbf{x}$ represents a high-resolution (HR) discretisation of the scene radiance, A models the camera process, $\boldsymbol{\eta}$ is zero-mean Gaussian noise and $\mathbf{b}$ is a vector of all resulting low-resolution (LR) images concatenated. Using Bayes's theorem, we find a maximum a-posteriori (MAP) estimate of the solution $\mathbf{x}$ by minimising the error

$$\|\mathbf{b} - A\mathbf{x}\|^2 + \lambda \mathbf{x}^\mathbf{T}\mathbf{x}.$$

This corresponds to the damped solution of the linear system $A\mathbf{x} = \mathbf{b}$.

It is useful to familiarise ourselves with the dimensions of the vectors and matrices involved. First, examine the noiseless model for a single frame,

$$A^{(i)}\mathbf{x} = \mathbf{b}^{(i)}$$

where i is the frame index. The first LR output image, $\mathbf{b}^{(0)}$, is the $P \times Q$ output image unpacked in lexicographic order. We assign $M = PQ$ as the dimensionality of $\mathbf{b}^{(0)}$. The vector $\mathbf{x}$ is the high-resolution image of dimensionality $zP \times zQ$, again unpacked in lexicographic order. The zoom factor, z with $z > 1$, represents the increase in resolution; e.g., if $z = 2$ then the high-resolution image has twice as many pixels as the low-resolution image along each axis. The dimensionality of $\mathbf{x}$ is $N = z^2M$. Given the dimensions of $\mathbf{b}^{(0)}$ and $\mathbf{x}$, the shape of the matrix A has to be $M \times N = M \times z^2M$.

Since $M < N$, the system $A^{(0)}\mathbf{x} = \mathbf{b}^{(0)}$ is underdetermined, but when we

combine all k camera matrices and input images to form

$$A = \begin{bmatrix} A^{(0)} \\ A^{(1)} \\ \vdots \\ A^{(k)} \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} \mathbf{b}^{(0)} \\ \mathbf{b}^{(0)} \\ \vdots \\ \mathbf{b}^{(k)} \end{bmatrix}$$

the resulting system $A\mathbf{x} = \mathbf{b}$ is overdetermined if $k > z^2$. Note that, even when combining a large number of frames, there is no guarantee that each additional frame provides independent information (imagine, for example, the case where k identical frames are combined). In practice, the values of $\mathbf{x}$ can only be determined accurately in positions where multiple frames overlap.

The goal of this chapter is to explore the structure and formation of the matrix A, and to study different methods of solving $\mathbf{x}$ in the least-squares problem where $A\mathbf{x} = \mathbf{b}$ is overdetermined.

7.2 The camera matrix, A

The camera matrix A, which represents the image formation process, is the only customisable parameter in the linear problem $A\mathbf{x} = \mathbf{b}$, and has to be chosen with care. In the previous chapter we discuss a simplified expression for image formation,

$$\mathbf{b}^{(i)} = S \downarrow (h(\mathcal{T}^{(i)}(\mathbf{x}))) + \boldsymbol{\eta}^{(i)},$$

approximated as

$$\mathbf{b}^{(i)} = A^{(i)}\mathbf{x} + \boldsymbol{\eta}^{(i)}. \tag{7.1}$$

Note that here we examine the formation of a single image, $\mathbf{b}^{(i)}$, while the camera matrix that produces all low-resolutions frames is simply

$$A = \begin{bmatrix} A^{(0)} \\ A^{(1)} \\ \vdots \\ A^{(i)} \end{bmatrix}$$

as mentioned above. From (7.1) we note that the camera matrix encapsulates three processes: geometrical transformation, the effect of the point-spread function and down-sampling. How, then, should $A^{(i)}$ be calculated?

Each row of $A^{(i)}$ represents weights applied to values in $\mathbf{x}$ to form a single

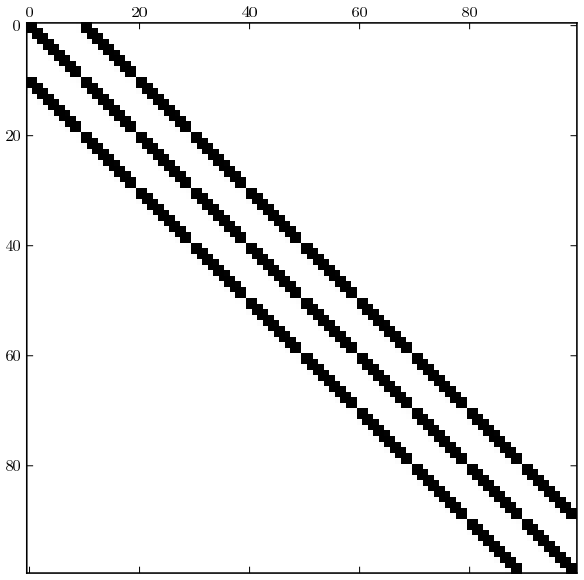


Figure 7.1: *The sparse matrix structure of a convolution operator. In this example, the convolution mask is* 3×3 *and the target image is* 10×10.

pixel of $\mathbf{b}^{(i)}$, the m-th pixel being

$$b_m^{(i)} = \sum_n A_{m,n}^{(i)} x_n. \tag{7.2}$$

If we combine the effect of transformation and down-sampling, $A^{(i)}$ can be approximated as

$$A^{(i)} = T^{(i)}C \quad \text{or} \quad A^{(i)} = CT^{(i)}$$

where C represents the effect of the point-spread function as a convolution, while $T^{(i)}$ represents geometric transformation accompanied by interpolation. The sparse nature of these operators are illustrated in Figures 7.1 and 7.2. While not visible in the above description, note that the geometric transformation itself is expressed as a linear transformation, $\mathbf{p}' = H\mathbf{p}$ where $\mathbf{p}$ is a homogeneous coordinate, as discussed in Chapter 5. The order of the operators is not arbitrary, and either choice presents certain difficulties. Importantly, the operator $T^{(i)}$ transforms a high-resolution image to a low-resolution image. Therefore, when convolution is applied first (to the high-resolution image), foreshortening due to the geometric transformation may lead to certain areas being more densely sampled than others. If the geometric transformation is applied first, with the convolution operator acting on the resulting low-resolution image,

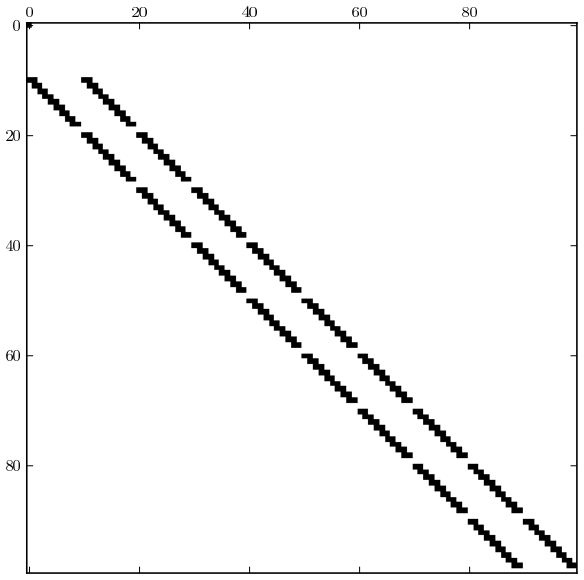


Figure 7.2: *The sparse matrix structure of a geometric transformation operator, employing* ***bilinear interpolation****. In this example, the transformation is a clockwise rotation by* $5°$*. Gaps appear when boundary pixels are met, and the interpolator returns zero (by design).*

some samples in the high-resolution image may not be taken into consideration at all. In [Cap01, p. 126], this problem is addressed by designing the camera matrix as follows:

1. Construct a convolution kernel (representing the camera point-spread function) that operates on the low-resolution image.

2. Use the known geometric transformation, H, to modify the kernel for operating on high-dimensional images. For example, each kernel coordinate will change from

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{to} \quad H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix},$$

   thereby also altering the kernel shape and the convolution path. Conversely, we can think of it as "fetching" all high-resolution pixels that contribute to a specific low-resolution pixel.

The approach works well, but introduces some challenges of its own:

- What should the shape and size of the convolution kernel be? The camera

response function is well modelled as a Gaussian kernel, but even so the optimal variance, σ^2, is unknown.

- How should the transformed kernel be represented and applied? Capel models the kernel as a piecewise bilinear surface, allowing easy transformation and integration.

As mentioned in the previous chapter, the kernel variance parameter, σ^2, can be established experimentally. Reconstructions are made while varying σ^2 until the result shows little oscillatory behaviour. Still, we prefer not to have the free parameter at all.

In the next section, we examine the simplified operator, $A^{(i)} = T^{(i)}$, which no longer requires such a parameter.

7.3 Linear interpolation operators

Note that, from here onwards, we neglect the frame number to simplify notation, i.e., $A = A^{(i)}$ and $T = T^{(i)}$.

The camera matrix can be approximated simply by bilinear interpolation,

$$A = T,$$

introducing the obvious flaw that only 4 high-resolution pixels are used to calculate the value of any low-resolution pixel. In reality, a low-resolution pixel may (and probably will) depend on more high-resolution pixels; the exact number being determined by the resolution increase, z, and the severity of the transformation, H. If, however, the zoom ratio is chosen conservatively, the approximation may be a good one, as illustrated in Figure 7.3.

Next, we examine the *bilinear interpolation* operator, whereafter *polygon-based interpolation* is introduced allowing any appropriate number of high-resolution pixels—irrespective of the zoom factor or the severity of the transformation—to contribute to the low-resolution pixel.

7.3.1 Bilinear interpolation

The bi-linear transformation/interpolation operator, $A = T$, is near-Toeplitz with interpolation coefficients appearing on the diagonals, as shown in Figure 7.2. The coefficients in Equation (7.2) are derived from bilinear interpolation as follows:

Suppose a function is known at four grid coordinates, namely $f_{00} = f(0,0)$, $f_{01} = f(0,1)$, $f_{10} = f(1,0)$, and $f_{11} = f(1,1)$. We need to calculate $f(x,y)$

(a) Input image (one of ten), upscaled.

(b) Reconstruction at 5× zoom with bi-linear interpolation. A high-resolution reconstruction is made, but the result is oscillatory due to the bilinear operator's small footprint.

(c) Reconstruction at 1.8× zoom with bi-linear interpolation. Note that, while the resolution of this reconstruction is low, the detail is at least as good as the 5× reconstruction above, but without oscillations. The small footprint of the bi-linear super-resolution operator is adequate for such low zoom factors.

(d) Reconstruction at 5× zoom with polygon interpolation. This operator has a variable size footprint, and is therefore capable of handling any size zoom factor.

Figure 7.3: *The effect of the zoom factor on bilinear and polygon super-resolution operators.*

with $0 \leq x \leq 1$, $0 \leq y \leq 1$. An exact answer is impossible to find, but linear interpolation gives the approximation

$$f(x,y) \approx \begin{bmatrix} 1-x & x \end{bmatrix} \begin{bmatrix} f_{00} & f_{01} \\ f_{10} & f_{11} \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix}.$$

This method is known as bi-linear interpolation (even though the successive combination of two linear operators is no longer linear). In the more general case, where (x,y) falls inside an arbitrary cell (again with surrounding function values f_{00}, f_{01}, f_{10}, and f_{11} known), we define two location variables,

$$u \;=\; x - \lfloor x \rfloor \quad \text{and} \quad t = y - \lfloor y \rfloor\,,$$

so that

$$\begin{aligned} f(x,y) &\approx \begin{bmatrix} 1-u & u \end{bmatrix} \begin{bmatrix} f_{00} & f_{01} \\ f_{10} & f_{11} \end{bmatrix} \begin{bmatrix} 1-t \\ t \end{bmatrix} \\ &= f_{00}(1-u)(1-t) + f_{01}(t)(1-u) + f_{10}(u)(1-t) + f_{11}(u)(t). \end{aligned} \tag{7.3}$$

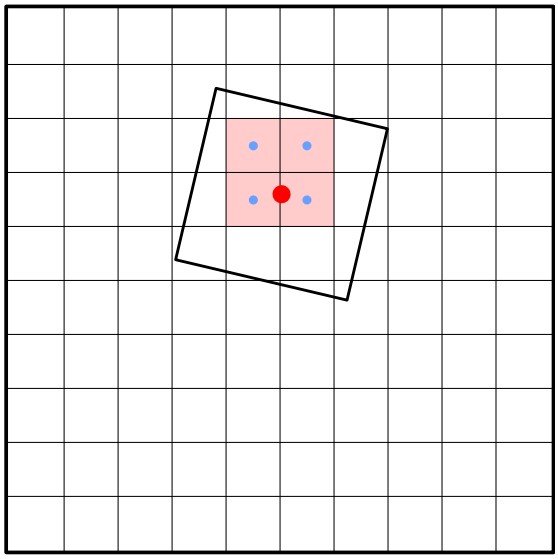


Figure 7.4: *A low-resolution pixel, transformed to the high-resolution image. The centre of the transformed pixel (big dot) is used to find the nearest surrounding pixel centres (small dots), indicating which pixels in the high-resolution image are interpolated to inform the value of the low-resolution pixel.*

If all known grid-values of $f(x, y)$ are placed in a vector, $\mathbf{x}$, then

$$f(x, y) = \mathbf{a}^T \mathbf{x}, \tag{7.4}$$

where $\mathbf{a}$ is a sparse vector of interpolation coefficients derived from (7.3) ($\mathbf{a}$ has mostly zero entries, except where elements correspond to f_{00}, f_{01}, f_{10}, or f_{11} in $\mathbf{x}$). When computing $f(x, y)$ for several coordinate pairs, (x_i, y_i), (7.4) becomes

$$\mathbf{b} = A\mathbf{x}, \quad \mathbf{b} = \begin{bmatrix} f(x_0, y_0) \\ f(x_1, y_1) \\ \vdots \\ f(x_{N-1}, y_{N-1}) \end{bmatrix}, \tag{7.5}$$

analogous to (7.2).

For super-resolution reconstruction, we need to warp (and down-scale) the high-resolution image to produce a given low-resolution image. The transformation matrix that warps the high-resolution frame to the i-th low-resolution frame is

$$M = \left(H_{i,0}\right)^{-1} S \quad \text{with} \quad S = \begin{bmatrix} 1/z & 0 & 0 \\ 0 & 1/z & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(a) The high-resolution input image. A vector representation, $\mathbf{x}$, is obtained by unpacking the values in lexicographic order.

(b) The matrix-vector product, $A\mathbf{x}$, reshaped to form an image.

Figure 7.5: *Effect of the transformation and bilinear interpolation operator A on $\mathbf{x}$. Here, A was constructed to rotate by 5° and to downsample by 2.*

and $H_{i,0}$ being the transformation matrix for warping the i-th frame to the reference frame (see Figure. 7.4), as derived in Chapter 5. We can now rewrite (7.5) as

$$\mathbf{b} = A\mathbf{x}, \quad \mathbf{b} = \begin{bmatrix} f(M^{-1}\mathbf{c}_0) \\ f(M^{-1}\mathbf{c}_1) \\ \vdots \\ f(M^{-1}\mathbf{c}_{N-1}) \end{bmatrix}$$

where $\mathbf{c}_i$, $i = 0, \ldots, N-1$ represent all coordinates in the low-resolution frame. With knowledge of these coordinates, the matrix A can be constructed as in (7.3). Figure 7.5 shows the matrix-vector product $A\mathbf{x}$ where $\mathbf{x}$ is a high-resolution image and A performs bilinear interpolation after rotating by 5°.

7.3.2 Polygon-based interpolation

In [VH07], a polygon intersection scheme is presented as a *linear interpolator*. Subsequently, a member of the Space Telescope and Science Institute brought to our attention a related algorithm called Drizzle [FH02], used by NASA to fuse Hubble Space Telescope photographs (see `http://stsdas.stsci.edu/multidrizzle`). However, while both methods rely on intersecting quadrilaterals (four-cornered polygons that represent pixels) to determine pixel weights, formulating polygon intersection as a linear operator proves to be fundamental in its application to super-resolution reconstruction.

As in the previous section, we want to express a low-resolution output image,

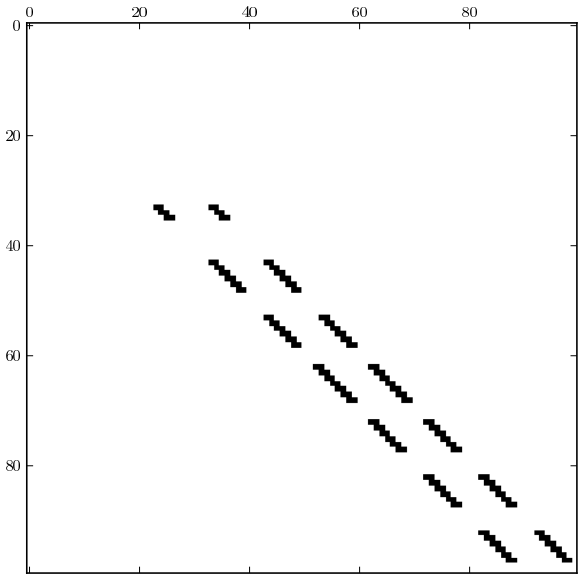


Figure 7.6: *The sparse matrix structure of a geometric transformation operator, employing* ***polygon interpolation****. In this example, the transformation is a clockwise rotation by 5°. The structure is very similar to that of the bilinear interpolator, except that the polygon interpolator has a wider "footprint", resulting in more rejected boundary pixels.*

b, as

$$\mathbf{b} = A\mathbf{x}$$

or, equivalently, each pixel in **b** as

$$b_m = \sum_n A_{m,n} x_n.$$

Each pixel value, b_m, depends on a number of pixels from the high-resolution image, **x**, weighted by the coefficients in row m of the operator A. The motivation for the polygon interpolation operator is as follows:

A camera sensor is a grid of photo-sensitive cells (think of them as photon buckets, each representing a pixel). Due to micro-lenses, the gaps between the cells are negligible. During imaging, the sensor irradiance is integrated over each cell for the duration of exposure, after which the values are read out as a matrix. Now, imagine two sensors, one with large cells (low-resolution) and the other with small cells (high-resolution), rotated relative to one another. How are the cell values for the different sensors related? Our proposed solution is to

measure the overlap between the larger and smaller cells, as shown in Figure 7.7. The value of a (large) low-resolution cell is set to a weighted sum of all (small) high-resolution cells; the weights depend on their overlap.

Algorithm 7.1 outlines the calculation of the coefficients in A.

Algorithm 7.1 Calculating the coefficients of the polygon interpolation operator A. Also see Figure 7.7.

For each low-resolution pixel, b_m:

1. Create a quadrilateral (four-node polygon) from the corner-points of b_m. For example, the pixel at $(0, 0)$ would correspond to the polygon with nodes

$$\begin{aligned} \mathbf{x}_L^m &= (-0.5, 0.5, 0.5, -0.5) \\ \mathbf{y}_L^m &= (-0.5, -0.5, 0.5, 0.5). \end{aligned}$$

   The subscript L indicates "low-resolution" and the super-script is the pixel number.

2. Transform the polygon to the high-resolution frame, using the transformation matrix M^{-1} given in the previous section. The new corner coordinates are $\hat{\mathbf{x}}_L^m$, $\hat{\mathbf{y}}_L^m$. If any of the coordinates fall outside the high-resolution image, break this loop and continue to the next low resolution pixel (there may be other ways to handle boundary problems, but this is simple and works well).

3. Determine the bounding box of the newly formed polygon:

$$\begin{aligned} \mathbf{x}_{BB} &= \left(\lfloor \min \mathbf{x}'_L \rfloor, \lceil \max \mathbf{x}'_L \rceil, \lceil \max \mathbf{x}'_L \rceil, \lfloor \min \mathbf{x}'_L \rfloor\right) \\ \mathbf{y}_{BB} &= \left(\lfloor \min \mathbf{y}'_L \rfloor, \lfloor \min \mathbf{y}'_L \rfloor, \lceil \max \mathbf{y}'_L \rceil, \lceil \max \mathbf{y}'_L \rceil\right) \end{aligned}$$

4. For each high-resolution pixel inside the bounding box:

   (a) Assign the pixel number $n = iN + j$ where (i, j) is the grid position of the high-resolution pixel and N is the total number of columns in the high-resolution frame.

   (b) Create a quadrilateral from the corner-points of the high-resolution pixel with vertices $\mathbf{x}_H^n$ and $\mathbf{y}_H^n$.

   (c) Measure the area of overlap between the polygons $(\mathbf{x}_L^m, \mathbf{y}_L^m)$ and $(\mathbf{x}_H^n, \mathbf{y}_H^n)$, and assign the value to $A_{m,n}$.

5. Divide each row $A_{m,*}$ by its sum so that the weights add to one.

This operator has the advantage that it is parameter free, and has a variable size footprint that covers all the necessary high-resolution pixels. Furthermore, it has less of a smoothing effect than the bi-linear interpolator.

Computing the coefficients of A is more expensive for polygon interpolation than for bi-linear interpolation, due to the many polygon area intersection calculations (called "clipping" operations) involved. However, two conditions improve execution time when clipping:

1. One of the polygons is aligned with the grid.

2. Both polygons are convex.

The first observation is particularly important, since it allows the use of algorithms that clip polygons to a "viewport" (this is typically used to determine which part of a polygon falls inside the screen). The second means that a simpler class of algorithm can be employed. We use the Liang-Barsky algorithm [LB83], which is optimised for rapidly clipping convex polygons against a viewport. Another approach is that by Maillot [Mai92].

The area of the resulting non-self-intersecting clipped polygon is easily determined as given by [Bou];

$$a = \frac{1}{2} \sum_{i=0}^{N-1} (x_i y_{i+1} - x_{i+1} y_i),$$

where $\mathbf{x}$ and $\mathbf{y}$ are the polygon vertices in clock-wise (or anti-clock-wise) order. The clipping of the entire collection of pixels can easily be parallellised. If only the result of the operator, $A\mathbf{x}$, is required, it can be rapidly rendered via the graphical processing unit without explicitly calculating any polygon intersections (pixels are simply warped and added, while the GPU takes care of any clipping in its fixed pipeline).

Figure 7.8 illustrates the effect of A on a vector $\mathbf{x}$.

Note that the polygon interpolation operator has several advantages over bilinear interpolation: it accurately models the underlying sensor physics, it is easy to compute and it has a variable footprint that automatically adjusts to the underlying transformation.

It is important to realise that this interpolation model is only accurate *before* Bayer demosaicking takes place (a process which destroys much of the super-resolution information in any case).

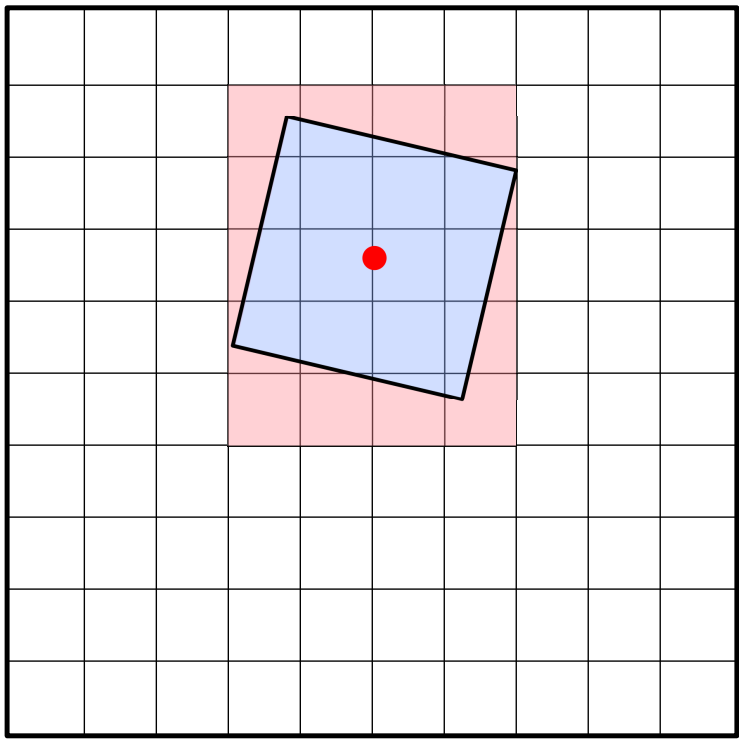


Figure 7.7: *Polygon interpolation "footprint". The dot indicates the centre of a low-resolution pixel, transformed to the high-resolution pixel grid. All high-resolution pixels touching the transformed pixel are used to interpolate the value of the low-resolution pixel.*

(a) The high-resolution input image. A vector representation, $\mathbf{x}$, is obtained by unpacking the values in lexicographic order.

(b) The matrix-vector product, $A\mathbf{x}$, reshaped to form an image.

Figure 7.8: *Effect of the transformation and polygon interpolation operator A on $\mathbf{x}$. Here, A was constructed to rotate by $5°$ and to downsample by 2.*

Software

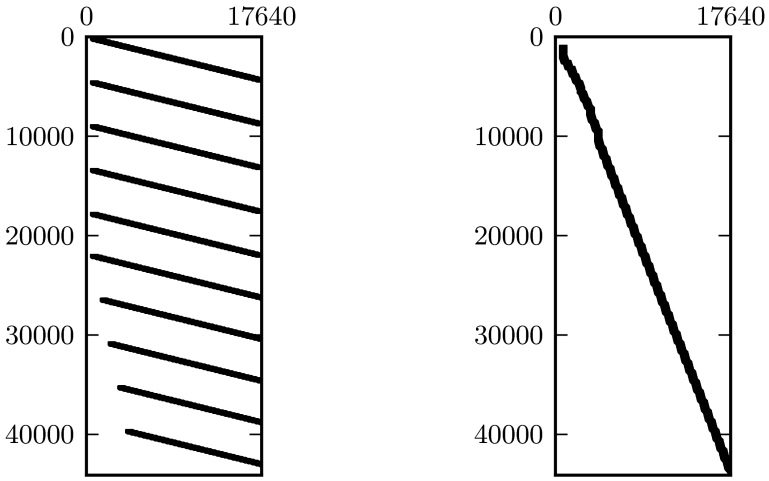
Polygon Clipping and Geometry

The following geometric functions are available in `supreme.ext`:

- `line_intersect`: Calculate the point of intersection between two lines.
- `npnpoly`: Given a collection of points, determine which fall inside a given polygon.
- `poly_clip`: Clip a convex polygon to a given viewport (a rectangular polygon).
- `poly_interp_op`: Construct a linear interpolation operator as descibed in this section.

The following example generates the product $A\mathbf{x}$ shown in Figure 7.8.

```
from supreme.ext import poly_interp_op
from supreme.io import imread
from supreme.api import show

# Zoom factor
z = 1/2.

# Construct the transformation matrix
C = z * np.cos(5 / 180 .* np.pi)
S = z * np.sin(5 / 180 .* np.pi)
H = np.array([[C, -S, 0], [S, C, 0], [0, 0, 1]])

# Read the input image and convert to grey-scale
x = imread('chelsea.jpg', flatten=True)

# Construct the operator, A
M, N = x.shape
B = poly_interp_op(M, N, H, M//2, N//2)

# Apply the operator and reshape to an image
b = (B * x.flat).reshape((M/2, N/2))

show(x, b)
```



(a) Multi-frame polygon interpolation operator.

(b) Multi-frame polygon interpolation operator in standard form.

Figure 7.9: *Sparsity pattern of interpolation operators. When the operator A is applied to the high-resolution image vector $\mathbf{x}$ a number of low-resolution images are produced (10 in this case).*

7.4 Solving the large, sparse least-squares problem $A\mathbf{x} = \mathbf{b}$

The previous chapter gives the solution to the least squares problem as

$$\arg\min_x f(\mathbf{x}) \quad \text{with} \quad f(\mathbf{x}) = \|A\mathbf{x} - b\|_2^2 \tag{7.6}$$

or, with damping,

$$\arg\min_x f(\mathbf{x}) \quad \text{with} \quad f(\mathbf{x}) = \|A\mathbf{x} - b\|_2^2 + \lambda \mathbf{x}^T\mathbf{x}. \tag{7.7}$$

The operator A produces all low-resolution frames from a given high-resolution frame; its structure is shown in Figure. 7.9.

Like many other inverse problems, our problem is often ill-posed, especially when data is lacking. Given enough image frames with different geometric transformations, the solution is sufficiently constrained. Otherwise, without some form of regularisation, no usable result can be computed.

As a thought experiment, imagine reconstructing a high-resolution image from a single low-resolution image. In the low-resolution image, all high-frequency information has been removed. It is therefore impossible to determine a one-to-one correspondence between the two frames (i.e., many different high-resolution

images may map to the same low-resolution image). The more low-resolution frames we add, the better we are able to estimate the high-frequency information. Our ability to recover such information is limited by the accuracy of the parameters provided and the nature of our data set.

Robust approaches allow some form of damping that prevents the solution from moving too far away from a pre-specified $\mathbf{x}_0$. In the case of (7.7), $\mathbf{x}_0 = \mathbf{0}$.

Regularising toward an arbitrary vector

We solve for $\mathbf{x}$ in

$$\arg\min_x \left\{ \|A\mathbf{x} - \mathbf{b}\|_2^2 + \lambda \mathbf{x}^T \mathbf{x} \right\}, \tag{7.8}$$

or its algebraic equivalent,

$$(A^T A + \lambda I)\mathbf{x} = A^T \mathbf{b}. \tag{7.9}$$

We can easily modify the problem so that $\mathbf{x}$ is constrained not to $\mathbf{0}$ but to an arbitrary given vector $\mathbf{x}_0$. We first transform the right-hand side to

$$\hat{\boldsymbol{b}} = \boldsymbol{b} - A\boldsymbol{x}_0.$$

If $\mathbf{x}_0$ is a good estimate of $\mathbf{x}$, then we expect the solution of

$$\arg\min_{\boldsymbol{\delta x}} \left\{ \left\| A\boldsymbol{\delta x} - \hat{\mathbf{b}} \right\|_2^2 + \lambda (\boldsymbol{\delta x})^T (\boldsymbol{\delta x}) \right\}$$

to lie close to $\mathbf{0}$. We find our final solution by adding back $\mathbf{x}_0$ to obtain

$$\boldsymbol{x} = \boldsymbol{\delta x} + \boldsymbol{x}_0.$$

Conditioning

The regularisation parameter in (7.9) improves the conditioning of the linear system. Given the eigenvalues λ_i for $M = A^T A$, the condition number is defined as

$$\kappa(M) = \frac{\max(\lambda_i)}{\min(\lambda_i)} = \frac{\lambda_{max}}{\lambda_{min}}.$$

For $M = (A^T A + \alpha I)$ this becomes

$$\kappa(M) = \frac{\lambda_{max} + \alpha}{\lambda_{min} + \alpha}.$$

The improved conditioning comes at a cost: we no longer solve the same linear system as before.

Solving

Equation (7.9) can be solved with a direct sparse solver. A common alternative is to approach the optimisation problem in (7.8) using any standard gradient-based iterative minimisation algorithm. In contrast to direct methods, such an algorithm may be terminated early as soon as a satisfactory solution is found.

The next section lists a number of iterative minimisation methods: steepest descent (solves Equation 7.6), conjugate gradients (solves a variant of Equation 7.7), L-BFGS and LSQR (solves Equation 7.7).

For an in depth overview of iterative least-squares methods, we refer to [Bjo96].

Choice of norm

The 2-norm is not a very effective way of measuring the error between two vectors and the 1-norm, often suggested as an alternative, suffer from many of the same deficiencies. However, while the 2-norm severely penalises individual outliers, the 1-norm is more tolerant, and may lead to a noisier but slightly more detailed reconstructions. Since the 1-norm is discontinuous at zero, it has been suggested in [BR96] that it be replaced by the 2-norm around the origin.

In [KKL+07] an interior point method is proposed for solving the least squares problem (2-norm) with L1-regularisation.

Later in this chapter (see Section 7.5) we discuss recently developed alternative norms.

7.4.1 Iterative optimisation methods

Gradient descent

Gradient descent (also known as "steepest descent") minimises a function by taking steps down its gradient. Following an initial estimate, $\mathbf{x_0}$, the solution is estimate by iteratively improving $\mathbf{x}$ to

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \boldsymbol{\nabla} f(\mathbf{x}_k)$$

where

$$f(\mathbf{x}) = \|A\mathbf{x} - \mathbf{b}\|_2^2 + \lambda \mathbf{x}^T \mathbf{x}$$

and

$$\boldsymbol{\nabla} f(\mathbf{x}_k) = 2(A\mathbf{x}_k - \mathbf{b})^T A + 2\lambda \mathbf{x}^T.$$

The step size is adjusted using α. It is possible to include a prior term, although in practice we found this to be unnecessary, given a small enough α. Gradient

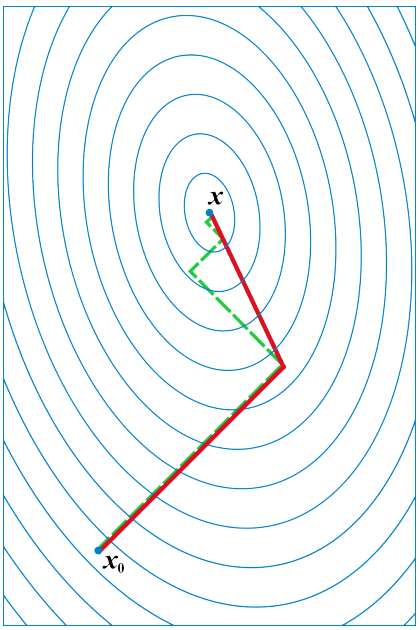


Figure 7.10: *Steps taken during function minimisation. The elliptical lines indicate contours of the target function, while the steps taken using gradient descent and conjugate gradients are shown as thich dashed and solid lines respectively. This figure is derived from an illustration released into the public domain by Oleg Alexandrov (http://commons.wikimedia.org/wiki/File:Conjugate_gradient_illustration.svg).*

descent may take long to converge if the function minimum lies at the bottom of a long, narrow trough.

Conjugate gradients

The conjugate gradient method is often used to solve large systems of linear equations where the matrix is positive semi-definite. This is the case for the normal least squares equation, where

$$\mathbf{x}^T(A^TA)\mathbf{x} = (A\mathbf{x})^T(A\mathbf{x}) \geq 0.$$

The conjugate gradient method improves upon gradient descent since each step is taken in a direction conjugate to those taken before (see Figure 7.10). To save memory and to improve conditioning, forming the entire normal matrix A^TA can be avoided; we simply require the matrix-vector products $A\mathbf{v}$ and $A^T\mathbf{v}$ to be computed. The Preconditioned Conjugate Gradient method is often recommended to improve convergence. It solves the system

$$M^{-1}A^TA\mathbf{x} = M^{-1}\mathbf{b}$$

where M^{-1} is some approximation of the inverse of A^TA. A common choice is to take M as the diagonal of A^TA (this is called the Jacobi preconditioner).

Bayesian prior An alternative approach is to use the conjugate gradient method simply as a nonlinear minimiser, armed with a cost function $\|A\mathbf{x} - \mathbf{b}\|_2$ and its gradient, $2(A\mathbf{x}_k - \mathbf{b})^TA$.

In our software, we make use of an open-source implementation (`scipy.optimize.fmin_cg`) of a non-linear conjugate gradient minimiser formulated by Polak & Ribiere and described in [NW00]. In the linear case, this approach reduces to the standard conjugate gradient method without incurring additional cost.

As mentioned above, we can ensure that a good solution is found by adding a penalisation term to the cost function:

$$f(\mathbf{x}) = \|A\mathbf{x} - b\|_2^2 + \lambda \|\mathbf{x} - \mathbf{x}_0\|_2^2 . \tag{7.10}$$

This ensures that the solution never deviates far from $\mathbf{x}_0$. A good initial estimate for $\mathbf{x}_0$ is obtained by upscaling and stacking (averaging) all low-resolution frames.

LSQR (least squares)

LSQR [PS82b, PS82a] is a method similar to conjugate gradients for solving large sparse least squares problems. According to the authors, it is algebraically equivalent to the symmetric conjugate gradient method applied to

$$(A^TA + \lambda I)\mathbf{x} = A^T\mathbf{b},$$

but has better numerical properties. Since our A matrix is often ill-conditioned, this improves conditioning. Like the conjugate gradient method, LSQR never needs to form A^TA explicitly, as long as the matrix-vector products $A\mathbf{v}$ and $A^T\mathbf{v}$ are available.

It is worth noting that sparse matrix-vector products can be computed very efficiently on modern Graphics Processing Units, as shown by Bell and Garland [BG09].

L-BFGS (Memory-limited Broyden-Fletcher-Goldfarb-Shanno)

This quasi-Newton algorithm takes steps based on an estimated Hessian, and allows specification of bounds on each variable. While the authors warn that convergence may be slow [ZBLN97], we've seen extremely rapid convergence (typically 20 to 40 function calls) specifically when minimising the square of the 2-norm.

7.4.2 Iterative-interpolation super-resolution

In [Ban09], a method named "iterative-interpolation super-resolution" (IISR) is decribed. We show that this method is equivalent to the standard formulation, given in [Cap01].

The algorithm produces iterative updates to a trial solution, $\mathbf{x}_k$, so that

$$\mathbf{x}_{k+1} = \mathbf{x}_k + R_0(\mathbf{b} - A\mathbf{x}_k). \tag{7.11}$$

The matrix R_0 is "the superposition of three consecutive operations, that is, translation and up-sampling of the LR frames, followed by interpolation of the HR image" [Ban09, p. 30]. In other words, R_0 is a close approximation of A^T. Substituting back into (7.11), with A^Treplacing R_0, the update equation becomes

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + A^T(\mathbf{b} - A\mathbf{x}_k) \\ &= \mathbf{x}_k + A^T\mathbf{b} - A^TA\mathbf{x}_k. \end{aligned}$$

Close to convergence, $\mathbf{x}_{k+1} = \mathbf{x}_k$, so that we have

$$A^TA\mathbf{x} = A^T\mathbf{b}$$

which is the normal equation for the least squares problem

$$\underset{\mathbf{x}}{\operatorname{argmin}} \|A\mathbf{x} - \mathbf{b}\|_2 .$$

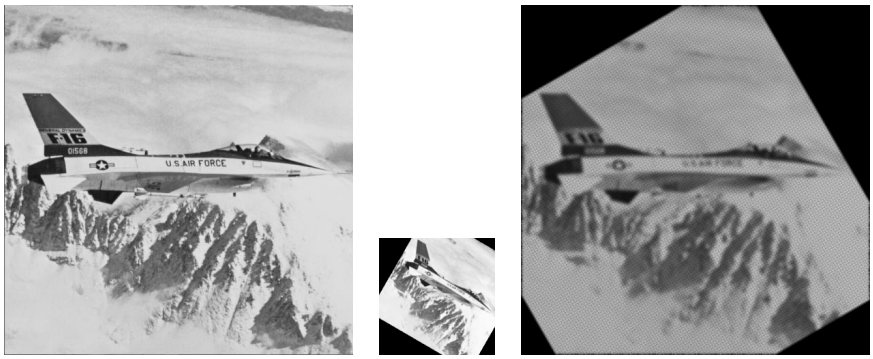
7.5 Structural metrics

In the above examples, we minimised the mean squared error (the two-norm squared) between two vectors. Interestingly, the behaviour of norms change as dimensionality increases, as explored in [Sco92]. Following [LV07], we see that the volume of a sphere in D dimensions is [Weg90]

$$V(r) = \frac{\pi^{\frac{D}{2}} r^D}{\Gamma\left(1 + \frac{D}{2}\right)}.$$

Therefore, the relative volume contained in a spherical shell of thickness ϵ is

$$\frac{V(r) - V(r(1-\epsilon))}{V(r)} = \frac{1^D - (1-\epsilon)^D}{1^D}$$



(a) $\mathbf{x}$, 512×512 **(b)** $A\mathbf{x}$, 170×170 **(c)** $A^T A\mathbf{x}$, 512×512

Figure 7.11: *The effect of A and A^T on images. In this example, the operator A represents convolving the image by a Gaussian window with standard deviation 2, followed by rotation through $30°$, translation, and a factor 3 downsampling. The input image, (a), is multiplied by A to give (b), which is multiplied by A^T to form (c). Note how A^T warps loe-resolution pixels back to the high-resolution grid, and fills out regions in between high-resolution pixels.*

which becomes one as D strives to infinity. In other words, almost the entire volume of a high-dimensional hypersphere is located in close proximity to its outer shell. This result highlights a weakness of the norms in high dimensions. All error vectors $\mathbf{v}_i$, relative to an image $\mathbf{f}$, of the form $\mathbf{v}_i = (\mathbf{f} - \mathbf{g}_i)$ lie close to the outer surface of the hypersphere centred around $\mathbf{f}$—their p-norms, $\|\mathbf{v}\|_p$, are therefore all very similar. This attribute is known as the concentration phenomenon. In his dissertation [Dem94], Demartines shows that, as dimensionality increases, the mean norm of a random vector grows proportionally to $\sqrt{D}$, while the variance stays approximately the same.

Despite this glaring deficiency, the family of p-norms is a popular choice due to their simplicity, and the ease of deriving their gradients—especially important in light of our minimisation process.

In [WB09], the authors introduce a new similarity index called the Structured Similarity Index Metric, or SSIM. The SSIM takes the structural content of the image into account when calculating the difference between two images. This method is also usable in optimisations, since the gradient can be calculated as shown in [WS08]. A wavelet-based extension of SSIM is also available [SWG+09].

We have not yet implemented this metric in the accompanying software.

7.6 Sensitivity to photometric registration

As shown in Figure 7.12, photometric registration is not of great importance in areas where most input frames overlap. However, neglecting photometric registration introduces prominent artefacts around frame edges. Under severe illumination changes (which are uncommon in super-resolution data-sets), the simple method proposed in Section 5.5 may not be adequate; this situation is explored in [GG07].

7.7 Recursive implementation

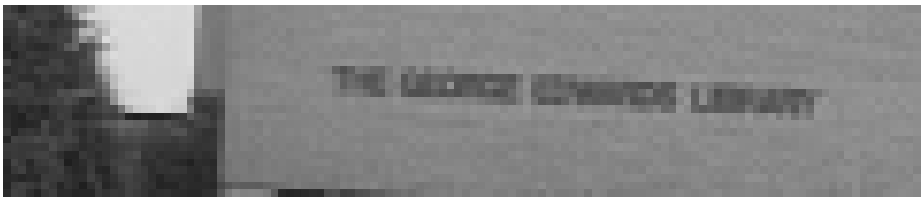
Above, we seek an $\mathbf{x}$ that minimises $\|A\mathbf{x} - \mathbf{b}\|_2^2 + \lambda \|\mathbf{x} - \mathbf{x}_0\|_2^2$, where $\mathbf{b}$ represents all low resolution images and $\mathbf{x}_0$ is our prior estimate of the response (usually, simply the upscaled average of all low-resolution frames). This is a convenient formulation when all frames are available, and when we have enough computer memory at our disposal.

If that is not the case, we need a procedure that processes a single frame at a time. This saves memory and requires only one image per iteration. The recursive implementation uses the output of a single run as the prior to the next:

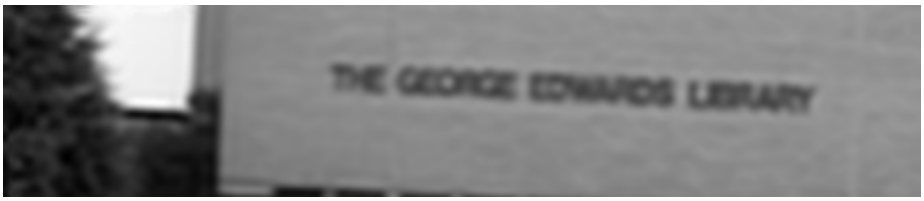
$$\begin{aligned}
\mathbf{x}_1 &= \arg\min_{\mathbf{x}} \|A_0\mathbf{x} - \mathbf{b}_0\|_2^2 + \lambda \|\mathbf{x} - \mathbf{x}_0\|_2^2 \\
\mathbf{x}_2 &= \arg\min_{\mathbf{x}} \|A_1\mathbf{x} - \mathbf{b}_1\|_2^2 + \lambda \|\mathbf{x} - \mathbf{x}_1\|_2^2 \\
&\vdots \\
\mathbf{x}_k &= \arg\min_{\mathbf{x}} \|A_{k-1}\mathbf{x} - \mathbf{b}\|_2^2 + \lambda \|\mathbf{x} - \mathbf{x}_{k-1}\|_2^2 .
\end{aligned}$$

7.8 Results

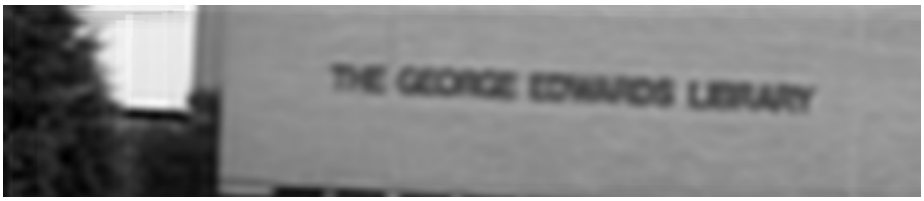
Results on a thirty-frame video sequence, with and without photometric registration, is given in Figure 7.12. Figure 7.13 illustrates reconstruction using the 1-norm as well as frame-by-frame updates. Figure 7.14 compares different optimisation methods.



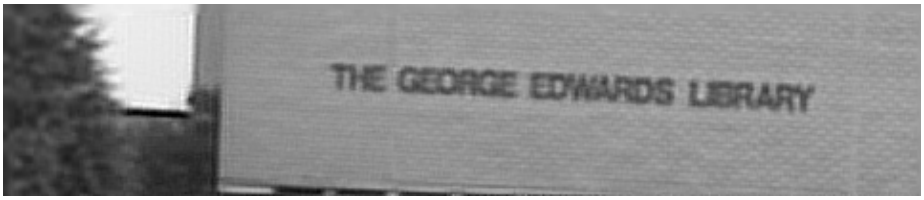
(a) Example input frame, upscaled (one of thirty).



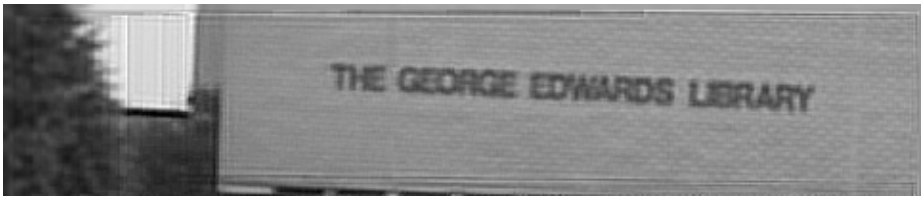
(b) All input frames, upsampled and stacked, after photometric registration.



(c) All input frames, upsampled and stacked, after multiplying by a random, uniformly distributed exposure factor between 0.75 and 1.25. Note the edges introduced in the region of sky.

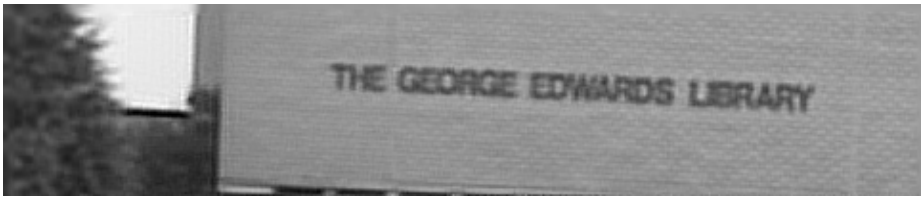


(d) Super-resolution result after photometric registration.

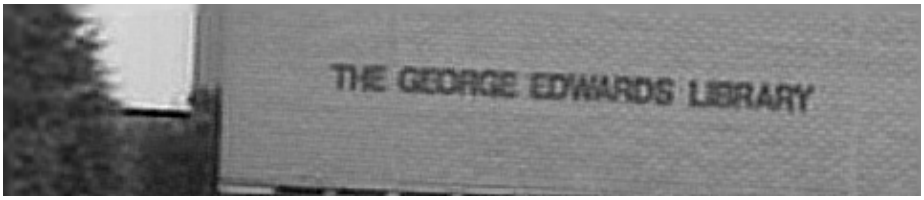


(e) Super-resolution result after disturbing the exposure as in (c).

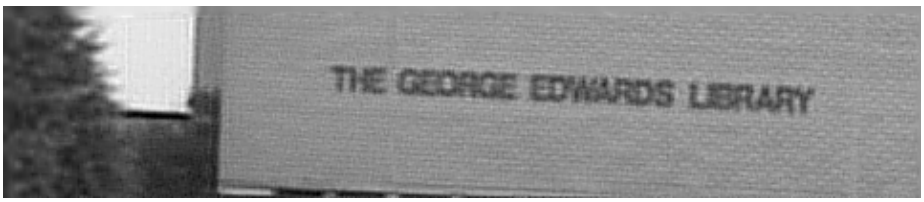
Figure 7.12: *The effect of photometric registration on super-resolution results. Note that, in the areas where there is significant frame overlap, the photometric registration has little effect. Edge effects, however, are markedly visible when photometric registration is neglected.*



(a) Super-resolution result: 4× resolution increase, polygon interpolator, 2-norm, $\lambda = 0.05$ and photometric adjustment. Note how even the bricks on the walls are resolved. This is not simply an oscillation, since the pattern is clearly tilted and aligned with the text and is absent in areas other than the wall. To be certain, we confirmed that the George Edwards Library of Surrey University has brick walls.



(b) Super-resolution result: 4× resolution increase, polygon interpolator, 2-norm, $\lambda = 0.01$ and photometric adjustment, *updated a single frame at a time*. This result is very similar to the direct calculation above.



(c) Super-resolution result: 4× resolution increase, polygon interpolator, *1-norm*, $\lambda = 0.2$ and photometric adjustment. Note how the 1-norm restoration is noisier than that of the 2-norm in (a).

Figure 7.13: *Super-resolution results on* library *data-set.*

(a) Original image, upscaled (one of ten).

(b) Gradient descent. Step size $\lambda = 0.5$

(c) Conjugate gradient method, $\lambda = 0.05$.

(d) LSQR, $\lambda = 0.2$.

(e) L-BFGS, $\lambda = 0.05$.

Figure 7.14: *Comparison of optimisation methods. In all frames a polygon interpolation operator is used, after applying photometric registration. The 2-norm is used and a single calculation is made (not a frame-by-frame update). The algorithms all return similar results.*

Software

Super-Resolution

All reconstructions shown in this chapter were performed using the `super_resolve.py` script located in the `doc/examples` subdirectory:

```
Usage: super_resolve.py [options] vgg_dir
Options:
  -h, --help            show this help message and exit
  -s SCALE, --scale=SCALE
                        Resolution improvement required
                        [default: 2]
  -d DAMP, --damp=DAMP  Damping coefficient --
                        suppresses oscillations [default: 0.1]
  -m METHOD, --method=METHOD
                        'CG', 'LSQR', 'L-BFGS-B' or 'descent'.
                        Specifies optimisation algorithm
                        [default: CG]
  -o OPERATOR, --operator=OPERATOR
                        'polygon' or 'bilinear'. The camera model
                        is approximated by this interpolation
                        scheme. [default: polygon]
  -u, --update          Use images as incremental evidence
                        [default: False]
  -p, --photo-adjust    Do not perform photometric adjustment
                        [default: True]
  -L NORM, --norm=NORM  The norm used to measure errors.
                        [default: 2]
  -c PREVIOUS_RESULT, --convergence=PREVIOUS_RESULT
                        Use a previously calculated result to track
                        convergence in "update"-mode. The image
                        file should be specified as the parameter.
  -i IGNORE, --ignore=IGNORE
                        Ignore this frame nr. May be specified more
                        than once.
```

Bibliography

[Ban09] V. Bannore. *Iterative-Interpolation Super-Resolution Image Reconstruction: A Computationally Efficient Technique*. Springer, 2009.

[BG09] Nathan Bell and Michael Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Conference on High Performance Networking and Computing*, 2009.

[Bjo96] Ake Bjorck. *Numerical Methods for Least Squares Problems*. SIAM, 1996.

[Bou] Paul Bourke. Calculating The Area And Centroid Of A

Polygon. http://local.wasp.uwa.edu.au/~pbourke/geometry/polyarea/.

[BR96] Michael J. Black and Anand Rangarajan. On the unification of line processes, outlier rejection, and robust statistics with applications in early vision. *International Journal of Computer Vision*, 19(1), 1996.

[Cap01] David Peter Capel. *Image Mosaicing and Super-resolution.* Ph.D. dissertation, University of Oxford, 2001.

[Dem94] P. Demartines. *Analyse de données par réseaux de neurones auto-organisés.* Ph.D. dissertation, Institut National Polytechnique de Grenoble, 1994.

[FH02] A. S. Fruchter and R. N. Hook. Drizzle: A Method for the Linear Reconstruction of Undersampled Images. *Publications of the Astronomical Society of the Pacific*, 114(792):144–152, February 2002.

[GG07] Murat Gevrekci and Bahadir K. Gunturk. Superresolution under Photometric Diversity of Images. *EURASIP Journal on Advances in Signal Processing*, 2007:1–13, 2007.

[KKL+07] Seung-jean Kim, K. Koh, M. Lustig, Stephen Boyd, and Dimitry Gorinevsky. An Interior-Point Method for Large-Scale l1-Regularized Least Squares. *IEEE Journal of Selected Topics in Signal Processing*, 1(4):606–617, 2007.

[LB83] You-Dong Liang and Brian A. Barsky. An analysis and algorithm for polygon clipping. *Communications of the ACM*, 26(11):868–877, 1983.

[LV07] J.A. Lee and M. Verleysen. *Nonlinear dimensionality reduction.* Springer Verlag, 2007.

[Mai92] Patrick-Gilles Maillot. A new, fast method for 2D polygon clipping: analysis and software implementation. *ACM Transactions on Graphics*, 11(3):276–290, July 1992.

[NW00] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, August 2000.

[PS82a] Christopher C. Paige and Michael A. Saunders. Algorithm 583: LSQR: Sparse Linear Equations and Least Squares Problems. *ACM Transactions on Mathematical Software*, 8(2):195–209, June 1982.

[PS82b] Christopher C. Paige and Michael a. Saunders. LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares. *ACM Transactions on Mathematical Software*, 8(1):43–71, March 1982.

[Sco92] D.W. Scott. *Multivariate density estimation: theory, practice, and visualization*. Wiley-Interscience, 1992.

[SWG+09] Mehul P. Sampat, Zhou Wang, Shalini Gupta, Alan Conrad Bovik, and Mia K. Markey. Complex wavelet structural similarity: a new image similarity index. *IEEE Transactions on Image Processing*, 18(11):2385–401, November 2009.

[VH07] S.J. Van Der Walt and B.M. Herbst. Methods Used in Increased Resolution Processing: Polygon based interpolation and robust log-polar based registration. In *Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP) 2007*, Barcelona, Spain, 2007.

[WB09] Zhou Wang and Alan C. Bovik. Mean Squared Error: Love It or Leave It? *IEEE Signal Processing Magazine*, pages 98–117, January 2009.

[Weg90] E.J. Wegman. Hyperdimensional data analysis using parallel coordinates. *Journal of the American Statistical Association*, 85(411):664–675, 1990.

[WS08] Zhou Wang and Eero P. Simoncelli. Maximum differentiation (MAD) competition: A methodology for comparing computational models of perceptual quantities. *Journal of Vision*, 8:1–13, 2008.

[ZBLN97] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization.

ACM Transactions on Mathematical Software (TOMS), 23(4), 1997.

Chapter 8

Conclusion

This study explores each component of super-resolution image reconstruction. First, an object is photographed from slightly different viewpoints; the relative object-camera motion causes light to illuminate the sensor differently in each image, providing the information that allows super-resolution. We discuss several common distortions that arise during acquisition, and focus on noise removal, which proved to be beneficial under certain circumstances.

Next, the input images are aligned; our emphasis is on feature-based registration. We provide a new feature detector, based on the discrete pulse transform, and show how to implement the transform efficiently. A statistical matching algorithm is introduced that is more robust than correlation under mild geometric transformations. Images are also photometrically registered by reviving an affine lighting model developed for correcting LANDSAT images.

The well-established maximum a-posteriori framework is used to obtain a super-resolution reconstruction. The underlying imaging model is linearised, wherafter possible simplifications to the model matrix are considered. We introduce a linear interpolation operator that models the individual pixels of the camera sensor using polygons. Based on this interpolation operator, a new model matrix is constructed at low cost; unlike other approaches, no parameters need to be specified.

Using one of several least-squares techniques, the over-determined system is solved using regularisation. The results obtained with the new polygon interpolation operator are highly satisfactory.

The entire software stack developed for these experiments is made available under an open source license, and may be used to verify the results presented in this dissertation.

Future directions

Colour super-resolution

In this work, the polygon-based interpolation operator, introduced to model sensor pixels, is applied to grey-level images only. However, there is no reason why the same model cannot be applied to colour sensors that use Bayer-masks.

Owing to the mask, not all colours are equally represented in the raw camera data. We propose seeking a solution, $\mathbf{x}$, that represents a high-resolution colour image as

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_R \\ \mathbf{x}_G \\ \mathbf{x}_B \end{pmatrix}.$$

Unlike any of the input frames, $\mathbf{x}$ has a red, green and blue value *at each pixel position.*

For each colour band $C \in \{R, G, B\}$, the linear system

$$\mathbf{x}_C = A_C \mathbf{b}_C$$

can be set up, based on the overlap of the low resolution pixels in $\mathbf{b}_C$ with the high-resolution pixels in $\mathbf{x}_C$. Since $\mathbf{x}_C$ covers the entire sensor, no special care needs to be taken of "holes" introduced by the Bayer mask. Given enough input images, a solution for $\mathbf{x}$ can be found.

A better norm for reconstruction

As discussed in Chapter 7, the two-norm is not well suited to comparing high-dimensional vectors. It would be interesting to compare suggested replacements, such as SSID, in order to see how reconstructions differ.

Non-linear geometric transformations

Our polygon-based interpolation operator models each pixel using a quadrilateral (a polygon with four vertices). This is well suited to the homographic transformation model used, but more advanced transformation models (that include radial distortion, for example), require a higher number of vertices. A comparison to current distortion models should prove interesting.

Appendix A

Data-set format

Directory Structure

The data-sets distributed with this package have been converted to the format used by Oxford's Vision and Geometry Group (http://www.robots.ox.ac.uk/~vgg/data/data-various.html). Each data-set has the following directory structure:

```
data_set/
data_set/png
data_set/png/data_set.000.png
data_set/png/data_set.001.png
data_set/H/data_set.000.001.H
```

The images are stored in the directory "`data_set/format`" where "`format`" is one of png, jpg or pgm. For each sequential image pair, a homography is provided in the "`data_set/H`" directory. For example, `data_set.000.001.H` is a text file containing 3 lines with 3 coefficients each, forming the 3×3 transformation matrix, $H_{0,1}$, which transforms a coordinate $\mathbf{c}_0$ from image 0 to a coordinate $\mathbf{c}_1$ in image 1:

$$\mathbf{c}_1 = H_{0,1}\mathbf{c}_0$$

Prefixing image and homography names with the data-set name is optional (i.e., a homography file may be named either `data_set.000.001.H` or `000.001.H`).

Converting relative to absolute homographies

Given the relative homographies $H_{0,1}$, $H_{1,2}$, $H_{2,3}$, etc., we can relate all these transformations to a fixed reference. Such a reference is needed, for example, when we stack images (i.e., transform them to the same reference frame and add them together). Noting that $H_{0,1} = H_{1,0}^{-1}$,

$$H_{K,0} = H_{0,K}^{-1} = \left(H_{K-1,K} \dots H_{2,3}H_{1,2}H_{0,1}\right)^{-1}. \tag{A.1}$$

While this could be written as

$$H_{K,0} = H_{0,1}^{-1} H_{0,2}^{-1} H_{0,3}^{-1} \dots H_{k-1,K}^{-1}$$

it is numerically more accurate to avoid the multiplication of multiple inverses by using (A.1).

In a similar fashion, given the transformation $H_{K,0}$ and $H_{k,0} \forall k < K$ (or, equivalently, $H_{K-1,0}$), we can compute the relative transformation, $H_{K-1,K}$:

$$\begin{aligned} H_{0,K} &= H_{K-1,K} \dots H_{2,3} H_{1,2} H_{0,1} \\ \implies H_{K-1,K} &= H_{0,K} (H_{K-2,K-1} \dots H_{2,3} H_{1,2} H_{0,1})^{-1} \\ &= H_{K,0}^{-1} (H_{K-2,K-1} \dots H_{2,3} H_{1,2} H_{0,1})^{-1} \\ &= H_{K,0}^{-1} H_{K-1,0}. \end{aligned}$$

Software

Loading VGG data-sets

```
>>> from supreme.io import load_vgg
>>> data = load_vgg('path/to/vgg/data_set')
```

The resulting data is an `ImageCollection`, which can be accessed like any container or iterator. The first image is `data[0]`, the second `data[1]` and so forth. Each image has an information dictionary, which contains two associated homographies:

`data[i].info['H']`

A 3×3 transformation matrix that maps image `i` onto image 0.

`data[i].info['H_rel']`

A 3×3 transformation matrix that maps image `i` onto image `i+1`.

Appendix B

Software API

Appendix B: Software Documentation

Release 0.9

Stefan van der Walt <stefan@sun.ac.za>

Contents

B.1 Overview B-2
B.1.1 Introduction B-2
B.1.2 License B-2
B.1.3 Installation from source B-3
B.2 API Reference B-3
B.2.1 Image Acquisition and I/O B-3
B.2.2 Discrete Pulse Transform B-5
B.2.3 Feature detection and matching B-7
B.2.4 Registration B-10
B.2.5 Super-resolution B-16
B.2.6 Miscellaneous B-22

Software References **B-26**

Index **B-27**

B.1 Overview

B.1.1 Introduction

SupReMe, short for Super Resolution Methods, is a library that implements the algorithms necessary to perform super-resolution imaging.

Super-resolution imaging is a process whereby several low-resolution photographs of a single object are combined to form a single, high-resolution reconstruction.

An overview of the underlying theory is given in the accompanying dissertation.

B.1.2 License

This software is released under the following free and open source license.

B.1.3 Installation from source

Obtain the source from the git-repository at http://dip.sun.ac.za/~stefan/code/supreme.git.

The package can be installed system-wide using

```
python setup.py install
```

or locally, using

```
python setup.py install --prefix=${HOME}
```

If preferred, you may use it without installing, by simply adding the source path to your PYTHONPATH variable and compiling the extensions in-place:

```
python setup.py build_ext -i
```

B.2 API Reference

B.2.1 Image Acquisition and I/O

Denoising (`supreme.noise`)

`supreme.noise.dwt_denoise`(X[, wavelet, ...])	Denoise an image using the Discrete Wavelet Transform.

`dwt_denoise`(*X, wavelet='db8', levels=4, alpha=2*)

Denoise an image using the Discrete Wavelet Transform.

Parameters

X : ndarray of uint8

Image to denoise.

wavelet : str

Wavelet family to use. See *supreme.lib.pywt.wavelist()* for a complete list.

levels : int

Number of levels to use in the decomposition.

alpha : float

Parameter used to tweak the Wiener estimator. A larger value of *alpha* results in a smoother output.

Returns

Y : ndarray of float64

Denoised image.

Notes

Implemented according to the overview of [R4] given in [R3].

References

[R3], [R4]

I/O (supreme.io)

supreme.io.Image	Image data with tags.
supreme.io.ImageCollection(file_pattern[, ...])	Load and manage a collection of images.
supreme.feature.SIFT.fromfile(f[, mode])	Read SIFT or SURF features from a file.
supreme.io.imread(name[, flatten])	Read an image file from a filename.
supreme.io.imshow	
supreme.io.load_vgg(path)	Load a VGG super-resolution data-set.
supreme.api.show(*images)	Display images on screen.
supreme.api.test_data()	Return an image for testing purposes.

class ImageCollection(*file_pattern, conserve_memory=True, grey=False*)

Load and manage a collection of images.

ImageCollection.__init__(file_pattern[, ...])	Load image files.
ImageCollection.__getitem__(n[, _cached])	Return image n in the queue.
ImageCollection.__iter__()	Iterate over the images.
ImageCollection.__len__()	Number of images in collection.

load_vgg(*path*)

Load a VGG super-resolution data-set.

Parameters

path : str

Path to the data-set.

Returns

ic : ImageCollection

An imagecollection of all the images, with the homographies stored in `x.info['H']` for each `x` in *ic*.

Notes

A VGG data-set stores the transformations from one frame to the next. This loader modifies all the homographies to be relative to the first frame.

References

[R6]

fromfile(*f, mode='SIFT'*)

Read SIFT or SURF features from a file.

Parameters

f : string or open file

Input file.

mode : string

'SIFT' or 'SURF'

Returns

data : record array with fields

- **row: int**
  row position of feature
- **column: int**
  column position of feature

- **scale: float**
  feature scale
- **orientation: float**
  feature orientation
- **data: array**
  feature values

`imread(`*name, flatten=0*`)`
Read an image file from a filename.

Optional arguments:

•flatten (0): if true, the image is flattened by calling convert('F') on

the resulting image object. This flattens the color layers into a single grayscale layer.

`show(`**images*`)`
Display images on screen.

`test_data()`
Return an image for testing purposes.

Returns
I : ndarray of uint8

512x512 test image.

B.2.2 Discrete Pulse Transform

`supreme.lib.dpt`

`connected_regions`	Return ConnectedRegions that, together, compose the whole image.
`decompose`	Decompose a two-dimensional signal into pulses.
`reconstruct`	Reconstruct an image from the given connected regions / pulses.

`connected_regions()`
Return ConnectedRegions that, together, compose the whole image.

Parameters
img : ndarray

Input image.

Returns
labels : ndarray

img, labeled by connectivity.

c : dict

Dictionary of ConnectedRegions, indexed by label value.

`decompose()`
Decompose a two-dimensional signal into pulses.

Parameters
img : 2-D ndarray of ints

Input signal.

Returns

pulses : dict

Dictionary of ConnectedRegion objects, indexed by pulse area.

See Also:

`reconstruct`

`reconstruct()`

Reconstruct an image from the given connected regions / pulses.

Parameters

regions : dict

Impulses indexed by area. This is the output of *decompose*.

shape : tuple

Shape of the output image.

min_area, max_area : int

Impulses with areas in [min_area, max_area] are used for the reconstruction.

Returns

out : ndimage

Reconstructed image.

class `ConnectedRegion()`

A 4 or 8-connected region is stored in a modified Compressed Sparse Row matrix format.

Since the region is connected, we only have to store one value. Along a single row, connected regions are stored as index pairs, e.g.

—00-000– would be represented as [3, 5, 6, 9]

This class should be queried using the methods in *connected_region_handler*.

Parameters

shape : tuple

Shape of the region.

Attributes

rowptr	list of int	*rowptr[i]* tells us where in *colptr* the elements of row i are described
colptr	list of int	Always contains 2N elements, where N are the number of connected regions (see description above). Each entry describes the half-open interval `(start_position, end_position]`.

supreme.lib.dpt.connected_region_handler

boundary_maximum	Return the maximum value on the boundary of the connected region.
boundary_minimum	Return the minimum value on the boundary of the connected region.
bounding_box	Return the bounding box of the connected region.
contains	Does the connected region contain an element at (r, c)?
copy	Return a deep copy of the connected region.
get_colptr	
get_rowptr	
get_shape	Return the shape of the connected region.
get_start_row	Return the first row where values of the connected region occur.
get_value	Return the value of the connected region.
merge	Merge b into a.
nnz	Return the number of non-zero elements.
outside_boundary	Calculate the outside boundary using a scanline approach.
reshape	Set the shape of the connected region.
set_array	Set the value of the array over the entire connected region.
set_start_row	Set the first row where values occur.
set_value	Set the value of the connected region.
todense	Convert the connected region to a dense array.
validate	Check the validity of the connected region descriptor.

B.2.3 Feature detection and matching

supreme.feature

dpt.features	Find feature points, using the discrete pulse transform.
match(features, featureset[, threshold])	For each given feature, find the nearest feature from a feature-set.
ransac	RANdom SAmple Consensus

features()

Find feature points, using the discrete pulse transform.

Parameters

pulses : dict

The pulses dictionary returned by the discrete pulse transform.

shape : tuple of ints

Shape of the image on which the DPT was performed.

win_size : int

Do not return more than one feature from any `win_size x win_size` shaped area.

Returns

weight : ndarray of float

An array of the same shape as the image, with values indicating the likelihood of any pixel being a feature.

area : ndarray of float

The estimated area of the feature at each pixel.

See Also:

`supreme.lib.dpt`

`match`(*features, featureset, threshold=0.59999999999999998*)

For each given feature, find the nearest feature from a feature-set.

Parameters

features : (M,N) array

M row-wise features of length N.

featureset : (Q,N) array

Q row-wise features of length N. This is typically the field 'data' of the record array produced by SIFT.fromfile.

Returns

nearest : Length M integer array.

Indices into featureset.

distances : Length M floating point array.

distances[i] is the distance between features[i] and featureset[nearest[i]], i.e. the distance between features[i] and the nearest feature in the feature-set.

valid : boolean array

A boolean array indicating whether the given feature match is valid, according to the criterion described in the SIFT README. It states that a match is valid when the match is less than 0.6 times the distance to the second-closest match.

See original implementation of vector quantisation by Tim Hochberg at :

http://thread.gmane.org/gmane.comp.python.numeric.general/8459/focus=8459 :

`supreme.feature.ransac`

class `RANSAC`(*model=None, p_inlier=0.5*)

RANdom SAmple Consensus

`RANSAC.__init__`([model, p_inlier])	Construct a RANSAC model fitter.
`RANSAC.__call__`([data, inliers_required, ...])	Execute RANSAC.

class `IModel`()

`IModel.__call__`(data[, confidence])	Evaluate data fit.
`IModel.estimate`(data)	Estimate model parameters from data.

`supreme.register`

`correspond`(fA, A, fB, B[, win_size])	Given coordinates of features in two images, determine

`correspond`(*fA, A, fB, B, win_size=9*)

Given coordinates of features in two images, determine possible correspondences using a Quantile-Quantile comparison.

Parameters

fA : list of tuple (x,y)

Coordinates of the features in the source image.

A : (m,n) ndarray of type uint8

Source image.

fB : list of tuple (x,y)

Coordinates of the features in the target image.

A : (m,n) ndarray of type uint8

Target image.

Returns

matches : list

[((coord_source), (coord_target)), ...]

`supreme.lib.fast`

Features from Accelerated Segment Test (FAST) corner detection.

Rosten and Drummond, "Fusing points and lines for high performance tracking." IEEE International Conference on Computer Vision, 2005

Rosten and Drummond, "Machine learning for high-speed corner detection", European Conference on Computer Vision, 2006

http://mi.eng.cam.ac.uk/~er258/work/fast.html

`corner_detect`(image[, barrier, size])	Detect corners.

`corner_detect(`*image, barrier=10, size=12*`)`

Detect corners.

Parameters

image : array of uint8

Input image.

barrier : int

Resistance to finding nearby corners.

size : int

Size of operator, must be in [9,12].

Returns

xy : Mx2 array

The M returned coordinates.

B.2.4 Registration

supreme.register

PointCorrespondence(ref_feat_rows, ...)	Estimate point correspondence homographies.
affine_tm([theta, tx, ty, scale, scale_x, ...])	Return the transformation matrix for an affine transformation.
dense_MI(A, B[, p, levels, fast, std, ...])	Register image B to A, using mutual information and an image pyramid.
joint_hist	Estimate the joint histogram of A and B.
lp_patch_match(a, b[, angles, Rs, plot_corr])	Align two patches, using the log polar transform.
mutual_info	Given the joint histogram of two images, calculate their mutual information.
ncc	Circular normalised cross-correlation of source and template image.
phase_corr(A, B)	Phase correlation of two images.
radial_sum	Sum the elements of an array outward along 360 directions (1-degree increments).
refine(reference, target, M_ref, M_target)	Refine registration parameters iteratively.
register	Perform image registration.
sat	Summed area table / integral image.
sat_sum	Using a summed area table / integral image, calculate the sum over a given window.
sparse(ref_feat_rows, ref_feat_cols, ...)	Compatibility wrapper.

class PointCorrespondence(*ref_feat_rows, ref_feat_cols, target_feat_rows, target_feat_cols, **args*)

Estimate point correspondence homographies.

Methods

RANSAC()	Estimate the homography using RANSAC.
estimate()	Estimate the homography.

PointCorrespondence.estimate()	Estimate the homography.
PointCorrespondence.RANSAC()	Estimate the homography using RANSAC.

affine_tm(*theta=0, tx=0, ty=0, scale=None, scale_x=None, scale_y=None*)

Return the transformation matrix for an affine transformation.

Parameters

theta : float

Rotation angle in radians.

tx, ty : float

X and Y translations.

scale : float

Scaling in both the X and the Y directions. Defaults to 1.

scale_x : float

Scaling in the X direction. Cannot be used together with *scale*.

scale_y : float

Scaling in the Y direction. Cannot be used with *scale*.

Returns

M : ndarray of float

Transformation matrix with the supplied parameters. Can be used to transform any homogeneous coordinate `p = [[x, y, 1]].T` by `np.dot(M, p)`.

`dense_MI`(*A, B, p=None, levels=3, fast=False, std=1, win_size=5, translation_only=False, fixed_scale=False*)

Register image B to A, using mutual information and an image pyramid.

Parameters

A, B : ndarray of uint

Images to register.

levels : int

Number of levels in the image pyramid. Each level is downsampled by 2.

p : list of floats, optional

The five initial parameters passed to the optimiser. These are rotation angle, skew in the X direction, skew in the Y direction, translation in x and translation in y.

fast : bool

If true, the histogram is not smoothed.

std : float

Standard deviation used by the smoothing window.

win_size : int (odd)

Window size of the smoother.

translation_only : bool

Whether to use a translation-only motion model. By default, a full homography is estimated.

fixed_scale : bool

Limit the scale of the motion model to 1.

Returns

M : (3,3) ndarray of float

Transformation matrix that transforms B to A.

`joint_hist`()

Estimate the joint histogram of A and B.

Parameters

A, B : (M, N) ndarray of uint8

Input images.

L : int

Number of grey-levels in histogram.

win_size : int

Width of Gaussian window used in the approximation. A larger window can represent the Gaussian kernel somewhat more accurately.

std : float

Standard deviation of the Gaussian used in the Parzen estimation. The higher the standard deviation, the smoother the resulting histogram. *win_size* must be made large enough to accommodate an increased standard deviation.

fast : bool

Calculate the classical histogram, instead of using a Parzen Window. Fast, but does not estimate the PDF as accurately.

Returns

H : (256, 256) ndarray of float

Estimation of the joint probability density function between A and B.

`lp_patch_match(`*a, b, angles=360, Rs=None, plot_corr=False*`)`

Align two patches, using the log polar transform.

Parameters

a : ndarray of uint8

Reference image.

b : ndarray of uint8

Target image.

angles : int

Number of angles to use in log-polar transform.

Rs : int

Number of radial samples used in the log-polar transform.

plot_corr : bool, optional

Whether to plot the phase correlation coefficients.

Returns

c : float

Peak correlation value.

theta : float

Estimated rotation angle from *a* to *b*.

scale : float

Estimated scaling from *a* to *b*.

`mutual_info()`

Given the joint histogram of two images, calculate their mutual information.

Parameters

H : (256, 256) ndarray of double

Returns

S : float

Mutual information.

`ncc()`

Circular normalised cross-correlation of source and template image.

Parameters

imgS : ndarray of uint8

Source image.

imgT : ndarray of uint8

Template image. The dimensions of the template image must be smaller or equal to that of the source.

Returns

ncc : ndarray of float

Normalised correlation coefficients, of the same shape as the source image.

Notes

While integral images are used, not all the suggestions made in [2] have been investigated.

References

[R13], [R14], [R15]

`phase_corr(`*A, B*`)`

Phase correlation of two images.

Parameters

A, B : (M,N) ndarray

Input images.

Returns

out : (M,N) ndarray

Correlation coefficients.

Examples

Set up test data. One array is offset (10, 10) from the other.

```
>>> x = np.random.random((50, 50))
>>> y = np.zeros_like(x)
>>> y[10:, 10:] = x[0:-10, 0:-10]
```

Correlate the two arrays, and ensure the peak is at (10, 10).

```
>>> out = phase_corr(y, x)
>>> m, n = np.unravel_index(np.argmax(out), out.shape)
>>> print m, n
(10, 10)
```

`radial_sum()`

Sum the elements of an array outward along 360 directions (1-degree increments).

Parameters

img : (M,N) ndarray of double

Input image.

Returns

R : (360,) ndarray of double

Summed elements of *img* along each of 360 directions. The central element, which belongs to all directions, is discarded.

Examples

```
>>> x = np.array([[2, 0, 1],
...               [0, 5, 0],
...               [3, 0, 4]], dtype=np.double)
>>> R = radial_sum(x)
>>> R[[45, 135, 225, 315]] == [1, 2, 3, 4]
```

`refine`(*reference, target, M_ref, M_target*)

Refine registration parameters iteratively.

`register`()

Perform image registration.

`sat`()

Summed area table / integral image.

The integral image contains the sum of all elements above and to the left of it, i.e.:

$$S[m, n] = \sum_{i \leq m} \sum_{j \leq n} X[i, j]$$

Parameters

X : ndarray of uint8

Input image.

Returns

S : ndarray

Summed area table.

References

[R16]

`sat_sum`()

Using a summed area table / integral image, calculate the sum over a given window.

Parameters

sat : ndarray of uint64

Summed area table / integral image.

r0, c0 : int

Top-left corner of block to be summed.

r1, c1 : int

Bottom-right corner of block to be summed.

Returns

S : int

Sum over the given window.

`sparse`(*ref_feat_rows, ref_feat_cols, target_feat_rows, target_feat_cols, **kwargs*)

Compatibility wrapper. Calculate the PointCorrespondence homography which maps reference features to target features.

See also: PointCorrespondence

Parameters

ref_feat_rows, ref_feat_cols : array of floats

Coordinates in the reference image.

target_feat_rows, target_feat_cols : array of floats

Coordinates in the target image.

mode : {'direct', 'iterative', 'RANSAC'}, optional

Method used to estimate the correspondences. See also `PointCorrespondence`. Use `direct` by default.

RANSAC_mode : {'direct', 'iterative'}, optional

Whether RANSAC should estimates homographies directly or iteratively.

`supreme.register.stack`

`with_transform`(images, matrices[, weights, ...])	Stack images after performing coordinate transformations.

`with_transform`(*images, matrices, weights=None, order=1, oshape=None, save_tiff=False, method='interpolate'*)

Stack images after performing coordinate transformations.

Parameters

images : list of ndarray

Images to be stacked.

matrices : list of (3,3) ndarray

Coordinate transformation matrices.

weights : list of float

Weight of each input image. By default, all images are weighted equally. The merging algorithm takes into account whether images overlap.

order : int

Order of the interpolant used by the scaling algorithm. Linear, by default.

oshape : tuple of int

Output shape. If not specified, the output shape is auto determined to include all images.

save_tiff : bool

Whether to save copies of the warped images. False by default.

method : {'interpolate', 'polygon'}

Use standard interpolation (default) or polygon interpolation. Note: Polygon interpolation is currently disabled.

Notes

For each image, a 3x3 coordinate transformation matrix, `A`, must be given. Each coordinate, `c = [x,y,1]^T`, in the source image is then translated to its position in the destination image, `d = A*c`.

After warping the images, they are combined according to the given weights. Note that the overlap of frames is taken into account. For example, in areas where only one image occurs, the pixels of that image will carry a weight of one, whereas in other areas it may be less, depending on the overlap of other images.

`supreme.photometry`

`histogram_adjust`(source, target)	Transform the histogram of the source so that it is similar to that of the target.
`photometric_adjust`(source, target)	Adjust the intensity of source to look like target.

`histogram_adjust`(*source, target*)

Transform the histogram of the source so that it is similar to that of the target.

Parameters

source, target = ndarray :

Source and target images.

Returns

source_adj : callable, f(x)

When applied to the source image, an image with similar response to target is generated.

`photometric_adjust`(*source, target*)

Adjust the intensity of source to look like target.

Parameters

source, target : ndarray

Source and target images.

Returns

a, b : float

Adjustment factors so that `source * a + b` approximates target.

B.2.5 Super-resolution

`supreme.resolve`

`initial_guess_avg`(images, tf_matrices, ...)	From the given low-resolution images and transforms, make an initial guess of the high-resolution image.
`lsqr.lsqr`(A, b[, damp, atol, btol, conlim, ...])	Find the least-squares solution to a large, sparse, linear system of equations.
`solve`(images, tf_matrices, scale[, x0, tol, ...])	Super-resolve a set of low-resolution images by solving a large, sparse set of linear equations.

`initial_guess_avg`(*images, tf_matrices, scale, oshape*)

From the given low-resolution images and transforms, make an initial guess of the high-resolution image.

Parameters

images : list of ndarray

Low-resolution images.

tf_matrices : list of (3, 3) ndarray

Transformation matrices that warp the images to the reference image (usually `images[0]`).

scale : float

The scale of the high-resolution reconstruction relative to the low-resolution frames. Typically between 1 and 2.

oshape : tuple of int

Shape of the high-resolution reconstruction.

`lsqr`(*A, b, damp=0.0, atol=1e-08, btol=1e-08, conlim=100000000.0, iter_lim=None, show=False, calc_var=False*)

Find the least-squares solution to a large, sparse, linear system of equations.

The function solves `Ax = b` or `min ||b - Ax||^2` or ‘‘min ||Ax - b||^2 + d^2 ||x||^2.

The matrix A may be square or rectangular (over-determined or under-determined), and may have any rank.

```
1. Unsymmetric equations --    solve  A*x = b

2. Linear least squares  --    solve  A*x = b
                               in the least-squares sense

3. Damped least squares  --    solve  (   A    )*x = ( b )
                                      ( damp*I )     ( 0 )
                               in the least-squares sense
```

Parameters

A : LinearOperator or equivalent

A representation of an mxn matrix. It is required that the linear operator can produce `Ax` and `A.T x`.

b : (m,) ndarray

Right-hand side vector `b`.

damp : float

Damping coefficient.

atol, btol : float

Stopping tolerances. If both are 1.0e-9 (say), the final residual norm should be accurate to about 9 digits. (The final x will usually have fewer correct digits, depending on cond(A) and the size of damp.)

conlim : float

Another stopping tolerance. lsqr terminates if an estimate of `cond(A)` exceeds *conlim*. For compatible systems `Ax = b`, *conlim* could be as large as 1.0e+12 (say). For least-squares problems, conlim should be less than 1.0e+8. Maximum precision can be obtained by setting `atol = btol = conlim = zero`, but the number of iterations may then be excessive.

iter_lim : int

Explicit limitation on number of iterations (for safety).

show : bool

Display an iteration log.

calc_var : bool

Whether to estimate diagonals of `(A'A + damp^2*I)^{-1}`.

Returns

x : ndarray of float

The final solution.

istop : int

Gives the reason for termination. 1 means x is an approximate solution to Ax = b. 2 means x approximately solves the least-squares problem.

itn : int

Iteration number upon termination.

r1norm : float

`norm(r)`, where `r = b - Ax`.

r2norm : float

`sqrt( norm(r)^2 + damp^2 * norm(x)^2 )`. Equal to *r1norm* if `damp == 0`.

anorm : float

Estimate of Frobenius norm of `Abar = [[A]; [damp*I]]`.

acond : float

Estimate of `cond(Abar)`.

arnorm : float

Estimate of `norm(A'*r - damp^2*x)`.

xnorm : float

`norm(x)`

var : ndarray of float

If `calc_var` is True, estimates all diagonals of `(A'A)^{-1}` (if `damp == 0`) or more generally `(A'A + damp^2*I)^{-1}`. This is well defined if A has full column rank or `damp > 0`. (Not sure what var means if `rank(A) < n` and `damp = 0.`)

Notes

LSQR uses an iterative method to approximate the solution. The number of iterations required to reach a certain accuracy depends strongly on the scaling of the problem. Poor scaling of the rows or columns of A should therefore be avoided where possible.

For example, in problem 1 the solution is unaltered by row-scaling. If a row of A is very small or large compared to the other rows of A, the corresponding row of (A b) should be scaled up or down.

In problems 1 and 2, the solution x is easily recovered following column-scaling. Unless better information is known, the nonzero columns of A should be scaled so that they all have the same Euclidean norm (e.g., 1.0).

In problem 3, there is no freedom to re-scale if damp is nonzero. However, the value of damp should be assigned only after attention has been paid to the scaling of A.

The parameter damp is intended to help regularize ill-conditioned systems, by preventing the true solution from being very large. Another aid to regularization is provided by the parameter acond, which may be used to terminate iterations before the computed solution becomes very large.

If some initial estimate `x0` is known and if `damp == 0`, one could proceed as follows:

1. Compute a residual vector `r0 = b - A*x0`.
2. Use LSQR to solve the system `A*dx = r0`.
3. Add the correction dx to obtain a final solution `x = x0 + dx`.

This requires that `x0` be available before and after the call to LSQR. To judge the benefits, suppose LSQR takes k1 iterations to solve A*x = b and k2 iterations to solve A*dx = r0. If x0 is "good", norm(r0) will be smaller than norm(b). If the same stopping tolerances atol and btol are used for each system, k1 and k2 will be similar, but the final solution x0 + dx should be more accurate. The only way to reduce the total work is to use a larger stopping tolerance for the second system. If some value btol is suitable for A*x = b, the larger value btol*norm(b)/norm(r0) should be suitable for A*dx = r0.

Preconditioning is another way to reduce the number of iterations. If it is possible to solve a related system `M*x = b` efficiently, where M approximates A in some helpful way (e.g. M - A has low rank or its elements are small relative to those of A), LSQR may converge more rapidly on the system `A*M(inverse)*z = b`, after which x can be recovered by solving M*x = z.

If A is symmetric, LSQR should not be used!

Alternatives are the symmetric conjugate-gradient method (cg) and/or SYMMLQ. SYMMLQ is an implementation of symmetric cg that applies to any symmetric A and will converge more rapidly than LSQR. If A is positive definite, there are other implementations of symmetric cg that require slightly less work per iteration than SYMMLQ (but will take the same number of iterations).

References

[R20], [R21], [R22]

`solve(`*images, tf_matrices, scale, x0=None, tol=1e-10, iter_lim=None, damp=0.10000000000000001, method='CG', operator='bilinear', norm=1, standard_form=False*`)`
Super-resolve a set of low-resolution images by solving a large, sparse set of linear equations.

This method approximates the camera with a downsampling operator, using bilinear or polygon interpolation. The LSQR method is used to solve the equation $A\mathbf{x} = b$ where A is the downsampling operator, $\mathbf{x}$ is the high-resolution estimate (flattened in raster scan/ lexicographic order), and $\mathbf{b}$ is a stacked vector of all the low-resolution images.

Parameters

images : list of ndarrays

Low-resolution input frames.

tf_matrices : list of (3, 3) ndarrays

Transformation matrices that relate all low-resolution frames to a reference low-resolution frame (usually `images[0]`).

scale : float

The resolution of the output image is *scale* times the resolution of the input images.

x0 : ndarray, optional

Initial guess of HR image.

damp : float, optional

If an initial guess is provided, *damp* specifies how much that estimate is weighed in the entire process. A larger value of *damp* results in a solution closer to *x0*, whereas a smaller version of *damp* yields a solution closer to the solution obtained without any initial estimate.

method : {'CG', 'LSQR', 'descent', 'L-BFGS-B'}

Whether to use conjugate gradients, least-squares, gradient descent or L-BFGS-B to determine the solution.

operator : {'bilinear', 'polygon'}

The camera model is approximated as an interpolation process. The bilinear interpolation operator only works well for zoom ratios < 2.

norm : {1, 2}

Whether to use the L1 or L2 norm to measure errors between images.

standard_form : bool

Whether to convert the matrix operator to standard form before processing.

Returns

HR : ndarray

High-resolution estimate.

`supreme.resolve.operators`

`bilinear`	Represent the camera process as a simple bilinear interpolation.
`convolve`	A linear operator that represents a convolution operation.
`block_diag`	Linear operator that represents diagonal block stacking.
`op_repeat`	Apply the given operator to N identically sized images.

`bilinear()`

Represent the camera process as a simple bilinear interpolation.

Parameters

MM, NN : int

Shape of the high-resolution image.

HH : list of (3,3) ndarray

Transformation matrices that warp the high-resolution frame to the individual low-resolution frames.

M, N : int

Dimensions of a single low-resolution output frame.

boundary : {0, 1}

Outside boundary use zero (0) or mirror (1).

Returns

A : (len(HH) * M * N, MM * NN) ndarray

Linear-operator representing bilinear interpolation from the HR image to the different LR images.

`convolve()`

A linear operator that represents a convolution operation.

Parameters

M, N : int

Shape of the output image.

mask_arr : (K,K) ndarray where K is odd

Mask to convolve with.

Returns

A : (M*N, M*N) sparse array

Linear operator that performs a convolution.

`block_diag()`

Linear operator that represents diagonal block stacking.

Repeats an (M, N) matrix diagonally to fit into and (MM, NN)-shaped matrix.

`op_repeat()`

Apply the given operator to N identically sized images.

`supreme.ext.poly_operator`

`poly_interp_op`	Construct a linear interpolation operator based on polygon overlap.

`poly_interp_op()`

Construct a linear interpolation operator based on polygon overlap.

Parameters

MM, NN : int

Shape of the high-resolution source frame.

H : (3, 3) ndarray

Transformation matrix that warps the high-resolution image to the low-resolution image.

M, N : int

Shape of the low-resolution target frame.

search_win : int

Search window size. Note TODO: this parameter should be automatically determined.

Returns

op : (M*N, MM*NN) sparse array

Interpolation operator.

B.2.6 Miscellaneous

supreme.geometry

Grid(rows, cols)	Regular grid.
Polygon(xp, yp)	Polygon class ..
window.gauss([size, std])	Discretised Gaussian window.

class Grid(*rows, cols*)

Regular grid.

__init__(*rows, cols*)

Create a grid given rows and columns.

coords()

Return an array of all coordinates.

class Polygon(*xp, yp*)

Polygon class

Methods

area()	Return the area of the polygon.
centroid()	Return the centroid of the polygon
inside(xp, yp)	Check whether the given points are inside the polygon.

__init__(*xp, yp*)

Given xp and yp (both 1D arrays or sequences), create a new polygon. The polygon is closed at instantiation.

gauss(*size=5, std=1.0*)

Discretised Gaussian window.

Parameters

size : int

The generated window has dimensions `(size, size)`.

std : float

Standard deviation.

Returns

w : (size, size) ndarray

Discretised Gaussian window.

supreme.ext

interp_bilinear(grey_image[, ...])	Calculate values at given coordinates using bi-linear interpolation.
interp_transf_polygon(grey_image, transform)	Compute an image transformation using polygon interpolation.
line_intersect(x0, y0, x1, y1, x2, y2, x3, y3)	Calculate the intersection between two lines.
npn_poly	
poly_clip(x, y, xleft, xright, ytop, ybottom)	Clip a polygon to the given bounding box.

`interp_bilinear`(*grey_image, transform_coords_r=None, transform_coords_c=None, mode='N', cval=0, output=None*)

Calculate values at given coordinates using bi-linear interpolation.

The output is of shape transform_coords_*. For each pair of values (transform_coords_r,transform_coords_c) the input image is interpolated to give the output value at that point.

`interp_transf_polygon`(*grey_image, transform, oshape=None*)

Compute an image transformation using polygon interpolation.

Parameters

grey_image : ndarray of uint8

Input image.

transform : 3x3 matrix of float

Transformation matrix.

oshape : tuple, optional

Shape of output image. Equal to input size if not specified.

Notes

This operation may also be performed by constructing a sparse polygon interpolation operator using `poly_interp_op`.

`line_intersect`(*x0, y0, x1, y1, x2, y2, x3, y3*)

Calculate the intersection between two lines.

The first line runs from (x0,y0) to (x1,y1) and the second from (x2,y2) to (x3,y3).

Return the point of intersection, (x,y), and its type:

0 – Normal intersection 1 – Intersects outside given segments 2 – Parallel 3 – Co-incident

`poly_clip`(*x, y, xleft, xright, ytop, ybottom*)

Clip a polygon to the given bounding box.

x and y are 1D arrays describing the coordinates of the vertices. xleft, xright, ytop and ybottom specify the borders of the bounding box. Note that a cartesian axis system is used such that the following must hold true:

x_left < x_right y_bottom < y_top

The x and y coordinates of the vertices of the resulting polygon are returned.

supreme.transform

`chirpz`(x, A, W, M)	Compute the chirp z-transform.
`homography`(image, matrix[, output_shape, ...])	Perform a matrix transform on an image.
`logpolar`(image[, angles, Rs, mode, cval, ...])	Perform the log polar transform on an image.

`chirpz`(*x, A, W, M*)

Compute the chirp z-transform.

The discrete z-transform,

X(z) = sum_{n=0}^{N-1} x_n z^{-n}

is calculated at M points,

z_k = AW^-k, k = 0,1,...,M-1

for A and W complex, which gives

X(z_k) = sum_{n=0}^{N-1} x_n z_k^{-n}

homography(*image, matrix, output_shape=None, order=1, mode='constant', cval=0.0, _coords=None*)

Perform a matrix transform on an image.

Each coordinate (x,y,1) is multiplied by matrix to find its new position. E.g., to rotate by theta degrees clockwise, the matrix should be

```
[[cos(theta) -sin(theta) 0]
 [sin(theta)  cos(theta) 0]
 [0            0         1]]
```

or to translate x by 10 and y by 20,

```
[[1 0 10]
 [0 1 20]
 [0 0 1 ]].
```

logpolar(*image, angles=None, Rs=None, mode='M', cval=0, output=None, _coords_r=None, _coords_c=None, extra_info=False*)

Perform the log polar transform on an image.

Returns

lpt : ndarray of uint8

Log polar transform of the input image.

angles : ndarray of float

Angles used. Only returned if *extra_info* is set to True.

log_base : int

Log base used. Only returned if *extra_info* is set to True.

References

[R8]

supreme.register

---

window_wrap Calculate the corner-coordinates of the sub-windows resulting when wrapping one window around another.

---

window_wrap()

Calculate the corner-coordinates of the sub-windows resulting when wrapping one window around another.

```
No wrapping:

.--------.
|  ___   |
| |   |  |
| |___|  |
|________|

Column wrapping:
```

```
.--------.
|_      _|
| |    | |
|_|    |_|
|________|

Row wrapping:

.--------.
|  |___| |
|       |
|  .___. |
|__|___|_|

Diagonal wrapping:

.--------.
|__|  |__|
|       |
|__.  .__|
|__|__|__|
```

Notes

The algorithm works as follows:

- •Assume that all wrappings take place
- •Calculate the corners for each resulting window
- •Remove windows with negative coordinates
- •Clip all coordinates to the clipping window boundary

Software References

[R3] J. Fridrich, "Digital Image Forensics," IEEE Signal Processing Magazine, vol. 26, 2009, pp. 26-37.

[R4] M. Mihcak, I. Kozintsev, K. Ramchandran, and P. Moulin, "Low-complexity image denoising based on statistical modeling of wavelet coefficients," IEEE Signal Processing Letters, vol. 6, 1999, pp. 300-303.

[R6] Super-resolution test sequences by David Capel, VGG, University of Oxford. http://www.robots.ox.ac.uk/~vgg/data/data-various.html

[R13] J.P. Lewis, Fast Normalized Cross-Correlation, 1995, http://www.idiom.com/~zilla/.

[R14] D. Tsai and C. Lin, "Fast normalized cross correlation for defect detection," Pattern Recognition Letters, vol. 24, 2003.

[R15] F.C. Crow, "Summed-area tables for texture mapping," ACM SIGGRAPH Computer Graphics, vol. 18, 1984, pp. 207-212.

[R16] F.C. Crow, "Summed-area tables for texture mapping," ACM SIGGRAPH Computer Graphics, vol. 18, 1984, pp. 207-212.

[R20] C. C. Paige and M. A. Saunders (1982a). "LSQR: An algorithm for sparse linear equations and sparse least squares", ACM TOMS 8(1), 43-71.

[R21] C. C. Paige and M. A. Saunders (1982b). "Algorithm 583. LSQR: Sparse linear equations and least squares problems", ACM TOMS 8(2), 195-209.

[R22] M. A. Saunders (1995). "Solution of sparse rectangular systems using LSQR and CRAIG", BIT 35, 588-604.

[R8] Matungka, Zheng and Ewing, "Image Registration Using Adaptive Polar Transform". IEEE Transactions on Image Processing, Vol. 18, No. 10, October 2009.

INDEX

Symbols

__init__() (supreme.geometry.Grid method), B-22
__init__() (supreme.geometry.Polygon method), B-22

A

affine_tm() (in module supreme.register), B-10

B

bilinear() (in module supreme.resolve.operators), B-20
block_diag() (in module supreme.resolve.operators), B-21

C

chirpz() (in module supreme.transform), B-23
connected_regions() (in module supreme.lib.dpt), B-5
ConnectedRegion (class in supreme.lib.dpt.connected_region), B-6
convolve() (in module supreme.resolve.operators), B-21
coords() (supreme.geometry.Grid method), B-22
corner_detect() (in module supreme.lib.fast), B-9
correspond() (in module supreme.register), B-8

D

decompose() (in module supreme.lib.dpt), B-5
dense_MI() (in module supreme.register), B-11
dwt_denoise() (in module supreme.noise), B-3

F

features() (in module supreme.feature.dpt), B-7
fromfile() (in module supreme.feature.SIFT), B-4

G

gauss() (in module supreme.geometry.window), B-22
Grid (class in supreme.geometry), B-22

H

histogram_adjust() (in module supreme.photometry), B-16
homography() (in module supreme.transform), B-24

I

ImageCollection (class in supreme.io), B-4
imread() (in module supreme.io), B-5
initial_guess_avg() (in module supreme.resolve), B-16
interp_bilinear() (in module supreme.ext), B-22
interp_transf_polygon() (in module supreme.ext), B-23

J

joint_hist() (in module supreme.register), B-11

L

line_intersect() (in module supreme.ext), B-23
load_vgg() (in module supreme.io), B-4
logpolar() (in module supreme.transform), B-24
lp_patch_match() (in module supreme.register), B-12
lsqr() (in module supreme.resolve.lsqr), B-17

M

match() (in module supreme.feature), B-8
mutual_info() (in module supreme.register), B-12

N

ncc() (in module supreme.register), B-12

O

op_repeat() (in module supreme.resolve.operators), B-21

P

phase_corr() (in module supreme.register), B-13
photometric_adjust() (in module supreme.photometry), B-16

PointCorrespondence (class in supreme.register), B-10
poly_clip() (in module supreme.ext), B-23
poly_interp_op() (in module supreme.ext.poly_operator), B-21
Polygon (class in supreme.geometry), B-22

R

radial_sum() (in module supreme.register), B-13
RANSAC (class in supreme.feature.ransac), B-8
reconstruct() (in module supreme.lib.dpt), B-6
refine() (in module supreme.register), B-14
register() (in module supreme.register), B-14

S

sat() (in module supreme.register), B-14
sat_sum() (in module supreme.register), B-14
show() (in module supreme.api), B-5
solve() (in module supreme.resolve), B-19
sparse() (in module supreme.register), B-14
supreme.feature.ransac.IModel (class in supreme.feature.ransac), B-8
supreme.lib.fast (module), B-9

T

test_data() (in module supreme.api), B-5

W

window_wrap() (in module supreme.register), B-24
with_transform() (in module supreme.register.stack), B-15