1 Introduction

1.1 Overview

This is where we start, by looking at the human visual system to investigate what is meant by vision, on to how a computer can be made to sense pictorial data and how we can process an image. The overview of this Chapter is shown in Table 1.1; you will find a similar overview at the start of each chapter. References/citations are collected at the end of each Chapter.

Main topic	Sub topics	Main points
Human vision system	How the **eye** works, how visual **information** is **processed** and how it can **fail**.	*Sight*, *vision*, lens, retina, image, colour, monochrome, processing, brain, visual illusions.
Computer vision systems	How electronic **images** are formed, how **video** is fed into a **computer** and how we can **process** the information using a computer.	*Picture elements*, *pixels*, *video* standard, *camera* technologies, pixel technology, performance effects, specialist cameras, video conversion.
Processing Images	How we can process images using the Python computer language and **mathematical packages**; introduction to Python and to **Matlab.**	*Programming* and processing images, visualisation of results, availability, use.
Literature	Other **textbooks** and other places to find **information** on image processing, computer vision and feature extraction.	*Magazines*, *textbooks*, websites and this book's website.

Table 1.1 Overview of Chapter 1

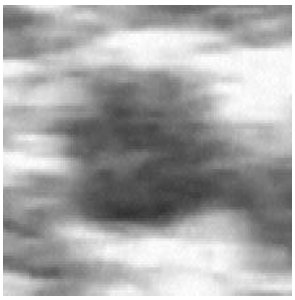
1.2 Human and Computer Vision

A computer vision system processes images acquired from an electronic camera, which is like the human vision system where the brain processes images derived from the eyes. Computer vision is a rich and rewarding topic for study and research for electronic engineers, computer scientists and many others. Now that cameras are cheap and widely available and computer power and memory are vast, computer vision is found in many places. There are now many vision systems in routine industrial use: cameras inspect mechanical parts to check size, food is inspected for quality, and images used in astronomy benefit from computer vision techniques. Forensic studies and biometrics (ways to recognise people) using computer vision include automatic face recognition and recognising people by the 'texture' of their irises. These studies are paralleled by biologists and psychologists who continue to study how our human vision system works, and how we see and recognise objects (and people).

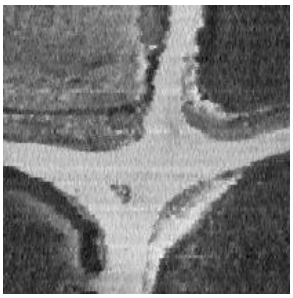
A selection of (computer) images is given in Figure 1.1, these images comprise a set of points or *picture elements* (usually concatenated to *pixels*) stored as an **array of numbers** in a **computer**. To recognise faces, based on an image such as Figure 1.1(a), we need to be able to analyse constituent shapes, such as the shape of the nose, the eyes, and the eyebrows, to make

some measurements to describe, and then recognise, a face. Figure 1.1(b) is an ultrasound image of the carotid artery (which is near the side of the neck and supplies blood to the brain and the face), taken as a cross section through it. The top region of the image is near the skin; the bottom is inside the neck. The image arises from combinations of the reflections of the ultrasound radiation by tissue. This image comes from a study aimed to produce three-dimensional models of arteries, to aid vascular surgery. Note that the image is very **noisy**, and this obscures the shape of the (elliptical) artery. Remotely sensed images are often analysed by their **texture** content. The perceived texture is different between the road junction and the different types of foliage seen in Figure 1.1(c). Finally, Figure 1.1(d) is a Magnetic Resonance Image (MRI) of a cross-section near the middle of a human body. The chest is at the top of the image, and the lungs and blood vessels are the dark areas, the internal organs and the fat appears grey. MRI images are in routine medical use nowadays, owing to their ability to provide high quality images.

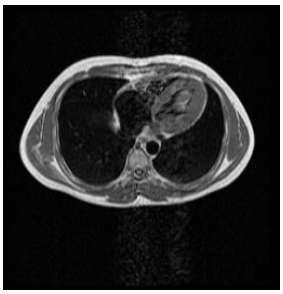
(a) face from a camera



(b) artery from ultrasound



(c) ground by remote-sensing

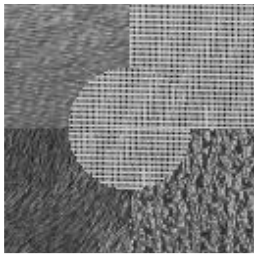


(d) body by magnetic resonance

Figure 1.1 Real images from different sources

There are many different image sources. In medical studies, MRI is good for imaging soft tissue, but does not reveal the bone structure (the spine cannot be seen in Figure 1.1(d)); this can be achieved by using Computerised Tomography (CT) which is better at imaging bone, as opposed to soft tissue. Remotely sensed images can be derived from infrared (thermal) sensors or Synthetic-Aperture Radar, rather than by cameras, as in Figure 1.1(c). Spatial information can be provided by two-dimensional arrays of sensors, including sonar arrays. There are perhaps more varieties of sources of spatial data in medical studies than in any other area. But computer vision techniques are used to analyse any form of data, not just the images from cameras.

(a) circles



(b) textures

Figure 1.2 Examples of synthesised images

Synthesised images are good for **evaluating** techniques and finding out how they work, and some of the bounds on **performance**. Two synthetic images are shown in Figure 1.2. Figure 1.2(a) is an image of circles that were specified **mathematically**. The image is an ideal case: the circles are perfectly defined and the brightness levels have been specified to be constant. This type of synthetic image is good for evaluating techniques which find the borders of the shape (its edges),

the shape itself and even for making a description of the shape. Figure 1.2(b) is a synthetic image made up of sections of real image data. The borders between the regions of image data are exact, again specified by a program. The image data comes from a well-known texture database, the Brodatz album of textures. This was scanned and stored as a computer image. This image can be used to analyse how well computer vision algorithms can identify regions of differing texture.

This Chapter will show you how basic computer vision systems work, in the context of the human vision system. It covers the main elements of human vision showing you how your eyes work (and how they can be deceived!). For computer vision, this Chapter covers the hardware and the software used for image analysis, giving an introduction to Python and Matlab, the software and mathematical package, respectively, used throughout this text to implement computer vision algorithms. Finally, a selection of pointers to other material is provided, especially those for more detail on the topics covered in this Chapter.

1.3 The Human Vision System

Human vision is a sophisticated system that senses and acts on **visual stimuli**. It has evolved for millions of years, primarily for defence or survival. Intuitively, computer and human vision appear to have the same function. The purpose of both systems is to interpret **spatial** data, data that is indexed by more than one dimension. Even though computer and human vision are functionally similar, you cannot expect a computer vision system to exactly replicate the function of the human eye. This is partly because we do not understand fully how the vision system of the eye and brain works, as we shall see in this section. Accordingly, we cannot design a system to exactly replicate its function. In fact, some of the properties of the human eye are useful when developing computer vision techniques, whereas others are actually undesirable in a computer vision system. But we shall see computer vision techniques which can, to some extent replicate, and in some cases even improve upon, the human vision system.

You might ponder this, so put one of the fingers from each of your hands in front of your face and try to estimate the distance between them. This is difficult, and I am sure you would agree that your measurement would not be very accurate. Now put your fingers very close together. You can still tell that they are apart even when the distance between them is tiny. So human vision can distinguish **relative** distance well, but is poor for **absolute** distance. Computer vision is the other way around: it is good for estimating absolute difference, but with relatively poor resolution for relative difference. The number of pixels in the image imposes the accuracy of the computer vision system, but that does not come until the next Chapter. Let us start at the beginning, by seeing how the human vision system works.

In human vision, the sensing element is the eye from which images are transmitted via the optic nerve to the brain, for further processing. The optic nerve has insufficient bandwidth to carry all the information sensed by the eye. Accordingly, there must be some pre-processing before the image is transmitted down the optic nerve. The human vision system can be modelled in three parts:

1) the eye – this is a physical model since much of its function can be determined by pathology;
2) a processing system – this is an experimental model since the function can be modelled, but not determined precisely; and
3) analysis by the brain – this is a psychological model since we cannot access or model such processing directly, but only determine behaviour by experiment and inference.

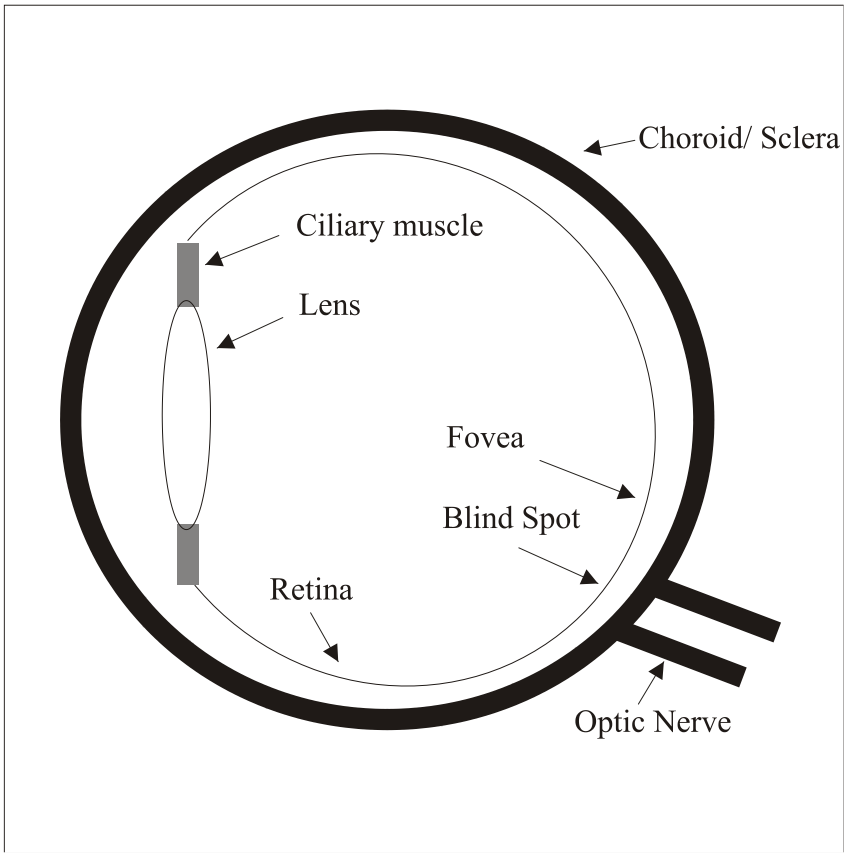


Figure 1.3 Human eye

1.3.1 The Eye

The function of the eye is to form an image; a cross-section of the eye is illustrated in Figure 1.**3**. Vision requires an ability to selectively focus on objects of interest. This is achieved by the *ciliary muscles* that hold the *lens*. In old age, it is these muscles which become slack and the eye loses its ability to focus at short distance. The *iris*, or pupil, is like an **aperture** on a camera and controls the amount of light entering the eye. It is a delicate system and needs protection, this is provided by the cornea (sclera). This is outside the *choroid* which has blood vessels that supply nutrition and is **opaque** to cut down the amount of light. The *retina* is on the inside of the eye, which is where light falls to form an image. By this system, muscles rotate the eye, and shape the lens, to form an image on the *fovea* (focal point) where the majority of sensors are situated. The *blind spot* is where the optic nerve starts, there are no sensors there.

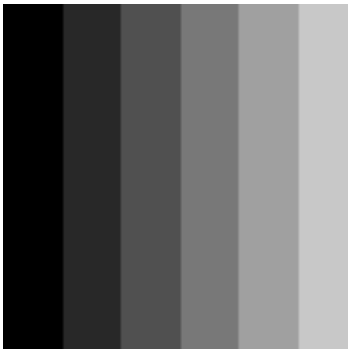
Focusing involves **shaping** the lens, rather than positioning it as in a camera. The lens is shaped to refract close images greatly, and distant objects little, essentially by 'stretching' it. The distance of the focal centre of the lens varies from approximately 14 mm to around 17 mm depending on the lens shape. This implies that a world scene is translated into an area of about 2 mm^2. Good vision has high *acuity* (sharpness), which implies that there must be very many sensors in the area where the image is formed.

There are actually nearly 100 million sensors dispersed around the retina. Light falls on these sensors to stimulate photochemical transmissions, which results in nerve impulses that are collected to form the signal transmitted by the eye. There are two types of sensor: firstly the *rods* – these are used for **black and white** (*scotopic*) vision; and secondly the *cones*- these are used for **colour** (*photopic*) vision. There are approximately 10 million cones and nearly all are found within 5^o of the fovea. The remaining 100 million rods are distributed around the retina, with the

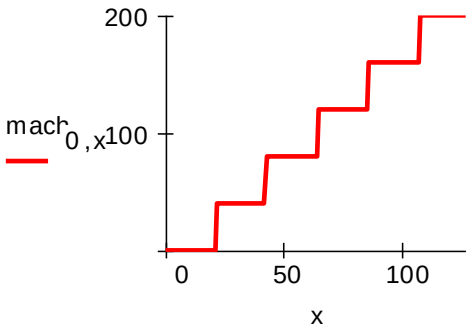
majority between 20° and 5° of the fovea. Acuity is actually expressed in terms of spatial resolution (sharpness) and brightness/ colour resolution and is greatest within 1° of the fovea.

There is only one type of rod, but there are three types of cones. These types are:

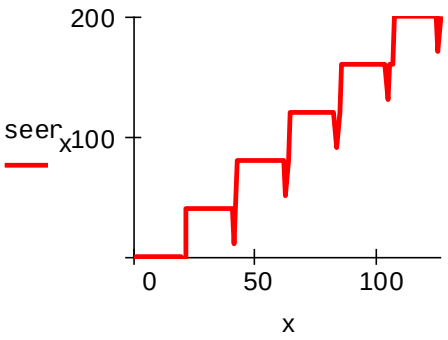
1. S – short wavelength: these sense light towards the blue end of the visual spectrum;
2. M – medium wavelength: these sense light around green; and
3. L – long wavelength: these sense light towards the red region of the spectrum.



(a) image showing the Mach band effect



(b) cross-section through (a)



(c) perceived cross-section through (a)

Figure 1.4 Illustrating Mach bands

The total response of the cones arises from summing the response of these three types of cone, this gives a response covering the whole of the visual spectrum. The rods are sensitive to light within the entire visual spectrum, giving the monochrome capability of scotopic vision. Accordingly, when the light level is low, images are formed away from the fovea, to use the superior sensitivity of the rods, but without the colour vision of the cones. Note that there are actually very few of the blueish cones, and there are many more of the others. But we can still see a lot of blue (especially given ubiquitous denim!). So, somehow, the human vision system compensates for the lack of blue sensors, to enable us to perceive it. The world would be a funny place with red water! The vision response is actually logarithmic and depends on brightness adaption from dark conditions where the image is formed on the rods, to brighter conditions where images are formed on the cones. More on colour sensing is to be found in Chapter 13, Appendix 4.

One inherent property of the eye, known as *Mach bands*, affects the way we perceive images. These are illustrated in Figure 1.4 and are the bands that appear to be where two stripes of constant shade join. By assigning values to the image brightness levels, the cross-section of plotted brightness is shown in Figure 1.4(a). This shows that the picture is formed from stripes of constant brightness. Human vision **perceives** an image for which the cross-section is as plotted in

Figure 1.4(c). These Mach bands do not really exist, but are introduced by your eye. The bands arise from overshoot in the eyes' response at boundaries of regions of different intensity (this aids us to differentiate between objects in our field of view). The **real** cross-section is illustrated in Figure 1.4(b). Note also that a human eye can distinguish only relatively few grey levels. It actually has a capability to discriminate between 32 levels (equivalent to five bits) whereas the image of Figure 1.4(a) could have many more brightness levels. This is why your perception finds it more difficult to discriminate between the low intensity bands on the left of Figure 1.4(a). (Note that Mach bands cannot be seen in the earlier image of circles, Figure 1.2(a), due to the arrangement of grey levels.) This is the limit of our studies of the first level of human vision; for those who are interested; [Cornsweet70] provides many more details concerning visual perception.

So we have already identified two properties associated with the eye that it would be difficult to include, and would often be unwanted, in a computer vision system: Mach bands and sensitivity to unsensed phenomena. These properties are integral to human vision. At present, human vision is far more sophisticated than we can hope to achieve with a computer vision system. Infrared guided-missile vision systems can actually have difficulty in distinguishing between a bird at 100 m and a plane at 10 km. Poor birds! (Lucky plane?) Human vision can handle this with ease.

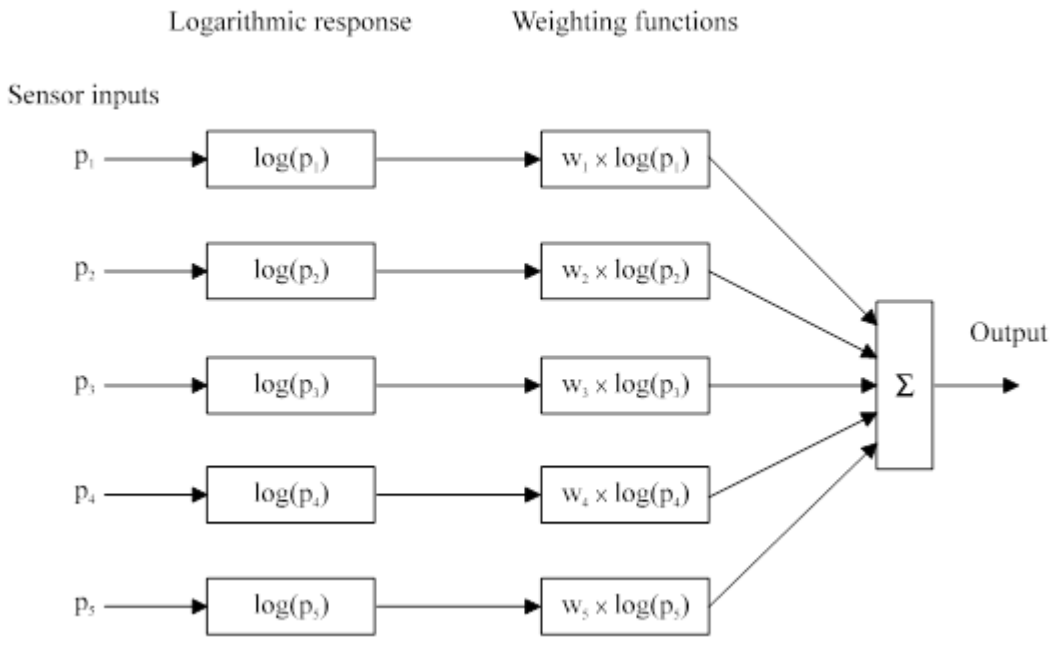


Figure 1.5 Neural processing

1.3.2 The Neural System

Neural signals provided by the eye are essentially the transformed response of the wavelength dependent receptors, the cones and the rods. One **model** is to combine these transformed signals by addition, as illustrated in Figure 1.5. The response is transformed by a logarithmic function, mirroring the known response of the eye. This is then multiplied by a weighting factor that controls the contribution of a particular sensor. This can be arranged to allow combination of responses from a particular region. The weighting factors can be chosen to afford particular filtering properties. For example, in *lateral inhibition*, the weights for the centre sensors are much greater than the weights for those at the extreme. This allows the response of the centre sensors to

dominate the combined response given by addition. If the weights in one half are chosen to be negative, whilst those in the other half are positive, then the output will show detection of contrast (change in brightness), given by the differencing action of the weighting functions.

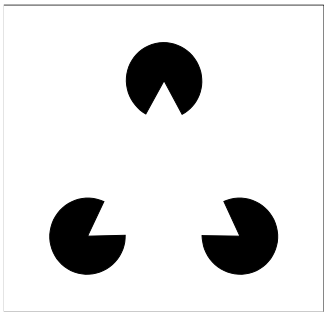
The signals from the cones can be combined in a manner that reflects *chrominance* (**colour**) and *luminance* (**brightness**). This can be achieved by subtraction of logarithmic functions, which is then equivalent to taking the logarithm of their ratio. This allows measures of chrominance to be obtained. In this manner, the signals derived from the sensors are combined prior to transmission through the optic nerve. This is an experimental model, since there are many ways possible to combine the different signals together.

Visual information is then sent back to arrive at the *Lateral Geniculate Nucleus* (LGN) which is in the thalamus and is the primary processor of visual information. This is a layered structure containing different types of cells, with differing functions. The axons from the LGN pass information on to the visual cortex. The function of the LGN is largely unknown, though it has been shown to play a part in coding the signals that are transmitted. It is also considered to help the visual system focus its attention, such as on sources of sound. For further information on retinal neural networks, see [Ratliff65]; an alternative study of neural processing can be found in [Overington92].

1.3.3 Processing

The neural signals are then transmitted to two areas of the brain for further processing. These areas are the *associative cortex*, where **links** between objects are made, and the *occipital cortex*, where **patterns** are processed. It is naturally difficult to determine precisely what happens in this region of the brain. To date, there have been no volunteers for detailed study of their brain's function (though progress with new imaging modalities such as Positive Emission Tomography or Electrical Impedance Tomography will doubtless help). For this reason, there are only psychological models to suggest how this region of the brain operates.

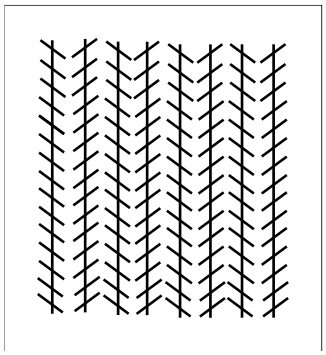
(a) word?



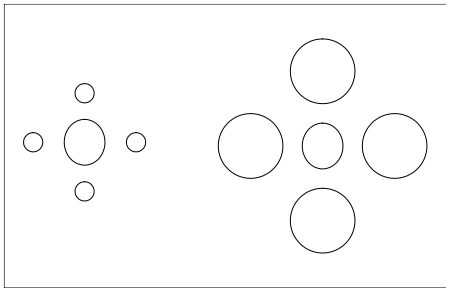
(b) Pacmen?

Figure 1.6 How human vision uses edges

It is well known that one function of the human vision system is to use edges, or boundaries, of objects. We can easily read the word in Figure 1.6(a), this is achieved by filling in the missing boundaries in the knowledge that the pattern most likely represents a printed word. But we can infer more about this image; there is a suggestion of illumination, causing shadows to appear in unlit areas. If the light source is bright, then the image will be washed out, causing the disappearance of the boundaries which are interpolated by our eyes. So there is more than just physical response, there is also knowledge, including prior knowledge of solid geometry. This situation is illustrated in Figure 1.6(b) that could represent three 'pacmen' about to collide, or a white triangle placed on top of three black circles. Either situation is possible.



(a) Zollner



(b) Ebbinghaus

Figure 1.7 Static illusions

It is also possible to deceive human vision, primarily by imposing a scene that it has not been trained to handle. In the famous *Zollner illusion*, Figure 1.7(a), the bars appear to be **slanted**, whereas in reality they are **vertical** (check this by placing a pen between the lines): the small crossbars mislead your eye into perceiving the vertical bars as slanting. In the *Ebbinghaus illusion*, Figure 1.7(b), the inner circle appears to be **larger** when surrounded by **small** circles, than it is when surrounded by larger circles.

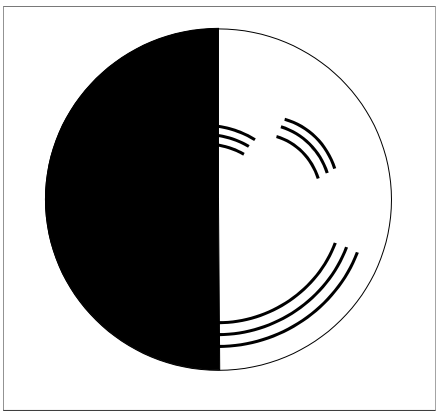


Figure 1.8 Benham's disk

There are dynamic illusions too: you can always impress children with the "see my wobbly pencil" trick. Just hold the pencil loosely between your fingers then, to whoops of childish glee, when the pencil is shaken up and down, the solid pencil will appear to bend. *Benham's disk*, Figure 1.8, shows how hard it is to model vision accurately. If you make up a version of this disk into a spinner (push a matchstick through the centre) and spin it anti-clockwise, you do not see three dark rings, you will see three **coloured** ones. The outside one will appear to be **red**, the middle one a sort of **green**, and the inner one will appear deep **blue**. (This can depend greatly on lighting - and contrast between the black and white on the disk. If the colours are not clear, try it in a different place, with different lighting.) You can appear to explain this when you notice that the red colours are associated with the long lines, and the blue with short lines. But that is from physics, not psychology. Now spin the disk clockwise. The order of the colours reverses: **red** is associated with the **short** lines (inside), and **blue** with the **long** lines (outside). So the argument from physics is clearly incorrect, since red is now associated with short lines not long ones, revealing the need for psychological explanation of the eyes' function. This is not colour

perception, see [Armstrong91] for an interesting (and interactive!) study of colour theory and perception.

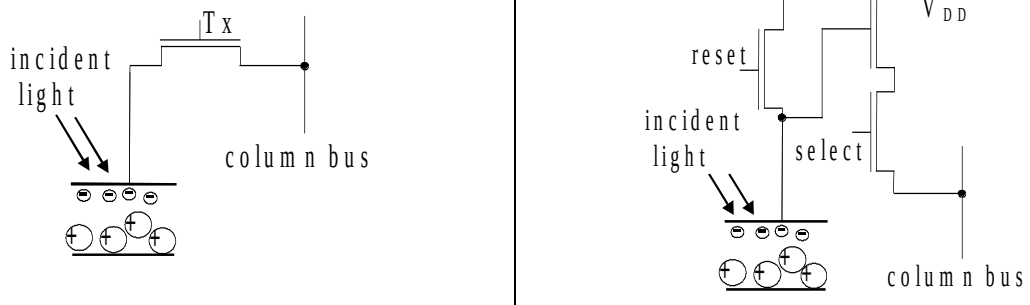
Naturally, there are many texts on human vision – one popular text on human visual perception (and its relationship with visual art) is by Livingstone [Livingstone14]; there is an online book: *The Joy of Vision* (http://www.yorku.ca/eye/thejoy.htm) – useful, despite its title! Marr's seminal text [Marr82] is a computational investigation into human vision and visual perception, investigating it from a computer vision viewpoint. For further details on pattern processing in human vision, see [Bruce90]; for more illusions see [Rosenfeld82] and an excellent – and dynamic – collection at michaelbach.de/ot. Many of the properties of human vision are hard to include in a computer vision system, but let us now look at the basic components that are used to make computers see.

1.4 Computer Vision Systems

Given the progress in computer technology and domestic photography, computer vision hardware is now relatively inexpensive; a basic computer vision system requires a camera, a camera interface and a computer. These days, some personal computers offer the capability for a basic vision system, by including a camera and its interface within the system. There are specialised systems for computer vision, offering high performance in more than one aspect. These can be expensive, as any specialist system is.

1.4.1 Cameras

A *camera* is the **basic sensing element**. In simple terms, most cameras rely on the property of light to cause hole/electron pairs (the charge carriers in electronics) in a conducting material. When a potential is applied (to attract the charge carriers), this charge can be sensed as current. By Ohm's law, the voltage across a resistance is proportional to the current through it, so the current can be turned in to a voltage by passing it through a resistor. The number of hole/electron pairs is proportional to the amount of incident light. Accordingly, greater charge (and hence greater voltage and current) is caused by an increase in brightness. In this manner cameras can provide as output, a voltage which is proportional to the brightness of the points imaged by the camera.



(a) passive **(b)** active

Figure 1.9 Pixel sensors

There are three main types of camera: *vidicons*, *charge coupled devices* (CCDs) and, more recently, *CMOS* cameras (Complementary Metal Oxide Silicon - now the dominant technology for logic circuit implementation). Vidicons are the **old** (analogue) technology, which though cheap (mainly by virtue of longevity in production) have largely been replaced by the **newer** CCD and CMOS **digital** technologies. The digital technologies now dominate much of the

camera market because they are **lightweight** and **cheap** (with other advantages) and are therefore used in the domestic video market.

Vidicons operate in a manner akin to a television in reverse. The image is formed on a screen, and then sensed by an electron beam that is scanned across the screen. This produces an output which is continuous, the output **voltage** is proportional to the **brightness** of points in the scanned line, and is a continuous signal, a voltage which varies continuously with time. On the other hand, CCDs and CMOS cameras use an array of sensors; these are regions where **charge** is collected which is proportional to the **light** incident on that region. This is then available in discrete, or **sampled**, form as opposed to the continuous sensing of a vidicon. This is similar to human vision with its array of cones and rods, but digital cameras use a **rectangular** regularly spaced lattice whereas human vision uses a **hexagonal** lattice with irregular spacing.

Two main types of semiconductor pixel sensors are illustrated in Figure 1.9. In the *passive sensor*, the charge generated by incident light is presented to a bus through a pass transistor. When the signal Tx is activated, the pass transistor is enabled and the sensor provides a capacitance to the bus, one that is proportional to the incident light. An *active pixel* includes an amplifier circuit that can compensate for limited fill factor of the photodiode. The select signal again controls presentation of the sensor's information to the bus. A further reset signal allows the charge site to be cleared when the image is re-scanned.

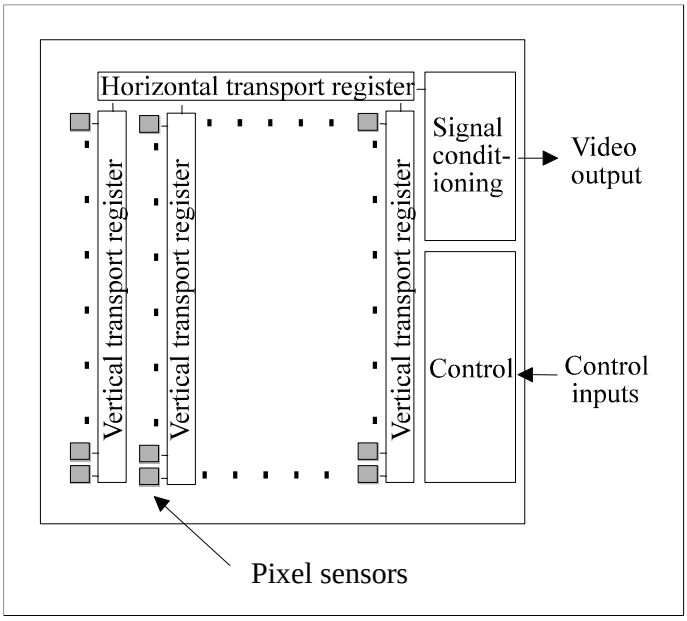


Figure 1.10 CCD sensing element

The basis of a CCD sensor is illustrated in Figure 1.10. The number of charge sites gives the resolution of the CCD sensor; the contents of the charge sites (or buckets) need to be converted to an output (voltage) signal. In simple terms, the contents of the buckets are emptied into vertical transport registers which are shift registers moving information towards the horizontal transport registers. This is the column bus supplied by the pixel sensors. The horizontal transport registers empty the information row by row (point by point) into a signal conditioning unit which transforms the sensed charge into a voltage which is proportional to the charge in a bucket, and hence proportional to the brightness of the corresponding point in the scene imaged by the camera. CMOS cameras are like a form of memory: the charge incident on a particular site in a two-dimensional lattice is proportional to the brightness at a point. The charge is then read like computer memory. (In fact, a computer memory RAM chip can act as a rudimentary form of camera when the circuit - the one buried in the chip - is exposed to light.)

There are many more varieties of vidicon (Chalnicon etc.) than there are of CCD technology (Charge Injection Device etc.), perhaps due to the greater age of basic vidicon technology. Vidicons are cheap but have a number of intrinsic performance problems. The scanning process essentially relies on 'moving parts'. As such, the camera performance will change with time, as parts **wear**; this is known as *ageing*. Also, it is possible to *burn* an image into the scanned screen by using high incident light levels; vidicons can also suffer *lag* that is a delay in response to moving objects in a scene. On the other hand, the digital technologies are dependent on the physical arrangement of charge sites and as such do not suffer from ageing, but can suffer from irregularity in the charge sites' (silicon) material. The underlying technology also makes CCD and CMOS cameras less sensitive to lag and burn, but the signals associated with the CCD transport registers can give rise to *readout effects*. CCDs actually only came to dominate camera technology when technological difficulty associated with *quantum efficiency* (the magnitude of response to incident light) for the shorter, blue, wavelengths were solved. One of the major problems in CCD cameras is *blooming* where bright (incident) light causes a bright spot to grow and disperse in the image (this used to happen in the analog technologies too). This happens much less in CMOS cameras because the charge sites can be much better defined and reading their data is equivalent to reading memory sites as opposed to shuffling charge between sites. Also, CMOS cameras have now overcome the problem of *fixed pattern noise* that plagued earlier MOS cameras. CMOS cameras are actually much more recent than CCDs. This begs a question as to which is best: CMOS or CCD? An early view was that CCD could provide higher quality images whereas CMOS is a cheaper technology and because it lends itself directly to intelligent cameras with on-board processing. The feature size of points (pixels) in a CCD sensor is limited to be about 4 μm so that enough light is collected. In contrast, the feature size in CMOS technology is considerably smaller. It is then possible to integrate signal processing within the camera chip and thus it is perhaps possible that CMOS cameras will eventually replace CCD technologies for many applications. However, modern CCDs' process technology is more mature, so the debate will doubtless continue!

Finally, there are specialist cameras, which include **high-resolution** devices (giving pictures with many points), **low-light level** cameras which can operate in very dark conditions and *infrared* cameras which sense heat to provide thermal images; *hyperspectral* cameras have more sensing bands. For more detail concerning modern camera practicalities and imaging systems see [Nakamura05] and more recently [Kuroda14]. For more detail on sensor development, particularly CMOS, [Fossum97] is still well worth a look. For more detail on images see [Phillips18] with a particular focus on quality (hey - there is even mosquito noise!).

1.4.2 Computer Interfaces

Though digital cameras continue to advance there are still some legacies from the older analog systems to be found in the newer digital systems. There is also some older technology in deployed systems. As such, we shall cover the main points of the two approaches. Essentially, the image sensor converts light into a signal which is expressed either as a continuous signal, or in sampled (digital) form. Some (older) systems expressed the camera signal as an analog continuous signal, according to a standard and this was converted at the computer (and still is in some cases, using a framegrabber). Modern digital systems convert the sensor information into digital information with on-chip circuitry and then provide the digital information according to a specified standard. The older systems, such as surveillance systems, supplied (or supply) video whereas the newer systems are digital. Video implies delivering the moving image as a sequence of *frames* of which one format is Digital Video – DV.

An analog continuous camera signal is transformed into **digital** (discrete) format using an Analogue to Digital (A/D) converter. *Flash converters* are usually used due to the high speed required for conversion (say 11 MHz that cannot be met by any other conversion technology). Usually, 8-bit A/D converters are used; at 6dB/ bit, this gives 48dB which just satisfies the CCIR stated *bandwidth* of approximately 45dB. The outputs of the A/D converter are then stored. Note that there are aspects of the sampling process which are of considerable interest in computer vision; these are covered in Chapter 2.

In digital camera systems this processing is usually performed on the camera chip, and the camera eventually supplies digital information, often in coded form. Currently Thunderbolt dominates the high end of the market and USB the lower end. There was a system called Firewire but it has now faded. Images are constructed from a set of **lines**, those lines scanned by a camera. In the older analog systems, in order to reduce requirements on transmission (and for viewing), the 625 lines (in the *PAL* system, *NTSC* is of lower resolution) were transmitted in two *interlaced fields*, each of 312.5 lines, as illustrated in Figure 1.11. These were the **odd** and the **even** fields. Modern televisions are *progressive scan*, which is like reading a book: the picture is constructed line by line. There is also an *aspect ratio* in picture transmission: pictures are arranged to be longer than they are high. These factors are chosen to make television images attractive to **human** vision. Nowadays, *digital video cameras* can provide digital output, in progressive scan delivering sequences of images that are readily processed. There are are Gigabit Ethernet cameras which transmit high speed video and control information over Ethernet networks. Or there are *webcams*, or just *digital camera systems* that deliver images straight to the computer. Life just gets easier!

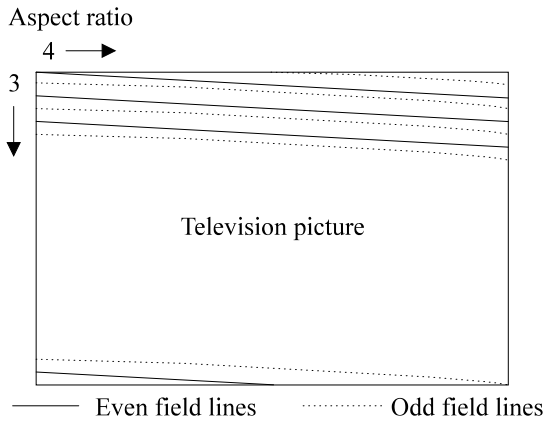


Figure 1.11 Interlacing in television pictures

1.5 Processing Images

We shall be using software and packages to process computer images. There are many programming languages and we have chosen Python as perhaps the most popular language at the time of writing. Several **mathematical systems** allow you transpose mathematics more easily from textbooks, and see how it works. Code functionality is not obscured by the use of data structures, though this can make the code appear cumbersome (to balance though, the range of datatypes is invariably small). A major advantage processing packages is that they provide the low-level functionality and data visualisation schemes, allowing the user to concentrate on

techniques alone. Accordingly, these systems afford an excellent route to understand, and appreciate, mathematical systems prior to development of application code, and to check the final code works correctly. The packages are however less suited to the development of application code. Chacun à son gout!

1.5.1 Processing

Most image processing and computer vision techniques are implemented in computer **software**. Often, only the simplest techniques migrate to hardware; though coding techniques to maximise efficiency in image transmission are of sufficient commercial interest that they have warranted extensive, and very sophisticated, hardware development. The systems include the Joint Photographic Expert Group (JPEG) and the Moving Picture Expert Group (MPEG) image coding formats. C, C++, Python and Java™ are by now the most popular languages for vision system implementation, because of strengths in integrating high- and low-level functions, and the availability of good compilers. As systems become more complex, C++, Python and Java become more attractive when encapsulation and polymorphism may be exploited. Many people use Python and JAVA as a development language partly due to platform independence, but also due to ease in implementation (though some claim that speed/ efficiency is better in C/C++). Python is currently a language of choice, not for any specific reasons. There are some textbooks that offer image processing systems implemented in these languages. Also, there are many commercial packages available, though these are often limited to basic techniques, and do not include the more sophisticated shape extraction techniques – and the underlying implementation can be hard to check. Some popular texts present working algorithms, such as [O'Gorman08], [Parker10] and [Solem12].

In terms of **software packages**, the most popular is OpenCV (Open Source Computer Vision) whose philosophy is to "aid commercial uses of computer vision in human-computer interface, robotics, monitoring, biometrics and security by providing a free and open infrastructure where the distributed efforts of the vision community can be consolidated and performance optimized". This contains a wealth of technique and (optimised) implementation – there's a Wikipedia entry and a discussion website supporting it. The system has now moved to OpenCV 3. That means there were versions 1 and 2 and unfortunately the systems are not compatible. The first textbook describing its use [Bradski08] had/has excellent descriptions of how to use the code (and some great diagrams) but omitted much of the (mathematical) background and analysis so it largely describes usage rather than construction. There are more recent texts for OpenCV 3 but the reviews are mixed: the web is a better source of reference for implementation material and systems that are constantly updated.

Then there are the VXL libraries (the Vision-*something*-Libraries, groan). This is "a collection of C++ libraries designed for computer vision research and implementation". The CImg Library (another duff acronym: it derives from Cool Image) is a system aimed to be easy to use, efficient, and a generic base for image processing algorithms. VLFeat is "a cross-platform open source collection of vision algorithms with a special focus on visual features". Finally, there is Southampton's OpenIMAJ which has lots of functional capabilities and is supported by tutorials and user data (as are all the other packages). Note that these packages are open source, and there are licences and conditions on use and exploitation. Web links are shown in Table 1.2.

Packages		
OpenCV	(originally Intel)	opencv.org/
VXL	Many international contributors	vxl.sourceforge.net/
CImg	Many international contributors	sourceforge.net/projects/cimg/

VLFeat	Oxford and UCLA	vlfeat.org/
OpenIMAJ	Southampton	openimaj.org

Table 1.2 Software packages for computer vision

1.5.2 Hello Python, Hello Images!

We shall be using *Python* as our main vehicle for processing images. It is now a dominant language and it is quite mature. Mark first encountered it a long time ago when we used it to arrange the conference proceedings of the British Machine Vision Conference BMVC 1998 and that was Python 1 (we also used Adobe 3.1). It has clearly moved from strength to strength, part by virtue of simplicity and part by virtue of some excellent engineering. This is not a textbook on programming in Python and for that you would best look elsewhere, e.g. [Lutz13] is a well proven text. Here we are using Python as a vehicle to show how algorithms can be made to work. For that reason, it's not a style guide either.

Python is a scripting programming language that was developed to provide clear and simple code. It is also very general, it runs on many platforms and it has comprehensive standard libraries. Here, we use Python to show how the maths behind machine vison technique can be translated into working algorithms. In general, there are several levels at which you can understand a technique. It is possible to understand the basic ideas by studying the meaning and aims of a method that your program calls from a library. You can also understand a more formal definition by studying the mathematics that support the ideas. Alternatively, you can understand a technique as a series of steps that define an algorithm that processes some data. In this book, we discuss the formal ideas with mathematical terms and then present a more practical approach by including algorithms derived from the mathematical formulation.

However, presenting working algorithms is not straightforward and we have puzzled on how to code without ending up with a lot of wrapper material that can obscure functionality or whether to go to the other extreme and just end up calling a function from a package. There again, one could argue that it's best to accommodate complex code in the support material, and show its operations in the book. However, that could limit functionality, performance and complexity. Today's implementations of machine vision algorithms include concepts like multi-threading and GPU processing. Implementations follow object oriented, functional or procedural definitions that organize computations in complex and efficient data structures. They also include specific code to improve robustness. In order to keep the scope manageable and for clarity, the code in this book is limited to the main steps of algorithms. However, you should be able to understand and develop more complex implementations once you have the knowledge of the basic approach.

If you want to execute the presented code, you will need to set up Python, install some Python packages and set up the book's scripts. There are several ways to set up a Python environment and we provide detailed setup instructions on the book's website. The scripts presented in this book do not call Python packages directly, but through two utility scripts (`ImageSupport` and `PlotSupport`) that wrap the basic image manipulations and drawing functionalities. The main idea is to present the code independently of software for drawing and loading images. You can either execute the scripts using the `ImageSupport` and `PlotSupport` definitions provided or you can implement similar utilities by using alternative Python libraries.

`ImageSupport` implements functions for reading, showing or creating images. `PlotSupport` implements functions for drawing surfaces, curves and histograms. The scripts in this book contain imports from these two utility files. In our implementation, `ImageSupport` uses pillow (a 'friendly' Python Imaging Library fork no less) for loading and saving images and

Numpy to define arrays for image data. PlotSupport uses Numpy since it requires image data and it also uses Matplotlib for drawing. Thus, if you use the support utilities, you need to install PIL, Numpy and Matplot and set your environment path such that scripts can access ImageSupport and PlotSupport utilities.

```
# Feature Extraction and Image Processing
# Mark S. Nixon & Alberto S. Aguado
# Code1_1_Imagedisplay.py: Loads, inverts and displays an image. Shows as a surface and prints pixel data

# Set utility folder
import sys
sys.path.insert(0, '../Utilities/')

# Set utility functions
from ImageSupport import imageReadL, showImageL, printImageRangeL, createImageF
from PlotSupport import plot3DColorHistogram

'''
Parameters:
   pathToDir = Input image directory
   imageName = Input image name
'''
pathToDir = "Input/"
imageName = "Square.png"

# Read image into array
inputImage, width, height = imageReadL(pathToDir + imageName)

# Show input image
showImageL(inputImage)s

# Print pixel's values in an image range
printImageRangeL(inputImage, [0, width-1], [0, height-1])

# Create an image to store the z values for surface
outputZ = createImageF(width, height)

# Three float arrays to store colors of the surface
colorsRGB = createImageF(width, height, 3)

# Set surface and colour values
for x in range(0, width):
   for y in range(0, height):
      pixelValue = float(inputImage[y,x])
      outputZ[y,x] = pixelValue
      pointColour = float(inputImage[y,x])/255.0
      colorsRGB[y,x] = [pointColour, pointColour, pointColour]

# Plot histogram to show image
plot3DColorHistogram(outputZ, colorsRGB, [0,400])
```

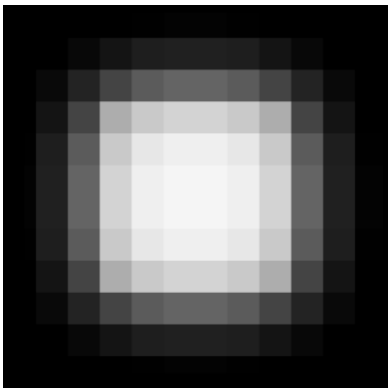
Code 1.1 Load, invert and display an image

In addition to image and plot support, scripts also import functions from other utility files. This is done in the code in this book to show all the steps in an implementation algorithm whilst avoiding repetition. For example, a technique such as edge detection will be shown as a script

with all the steps of the algorithm. When edge detection is used in more complex algorithms such as corner detection, the corner detection algorithm will just call a function to perform the edge detection steps. Thus, you will find the edge detection code as a script in an algorithm or as a utility function. The algorithm functions are organized by chapter, so `FourierTransformUtilities` defines functions for the scripts explained in Chapter 2 and `FeatureExtractionUtilities` for functions in Chapter 4. The code in the utilities is similar to the code explained in other parts of the book.

Code 1.1 shows our first example of a Python script. This script is used to visualize an image in different forms and loads an image, inverts it (subtracts points from 255) and then displays the output image. To run this code (from, say, the directory `c:\imagebook\` with the images stored in `c:\imagebook\input\`) then in the command line type `Python Code1_1_Imagedisplay.py` and it will run and produce images. The first lines of the script set the path to the directory of the utilities. Scripts in other chapters will omit these lines, but you need to set the environment in your system to access the utility files. The next part of the script imports the functions that we use for image handling and plotting. Then, we set the variables that define the parameters of the algorithm. In this case, the file name of the image we want to display. The utility function `imageReadL` reads an image and stores the data in an array (a matrix) of grey level values. The storage of image pixel data is first covered in the next Chapter, Section 2.2. The function returns an array containing a single grey level value `uint8` (8-bit unsigned integer) per pixel and the size of the image. The function `showImageL` displays an image on the screen as shown in Figure 1.13(a). In this example, the image is very small (12×12 pixels), so we can see the pixels as squares. The script uses the utility function `printImageRangeL` to display the values of the pixels in matrix form as shown in Figure 1.13(b). Note that the outer rows and columns are largely white with some slight variation – this variation cannot be perceived in the displayed image.

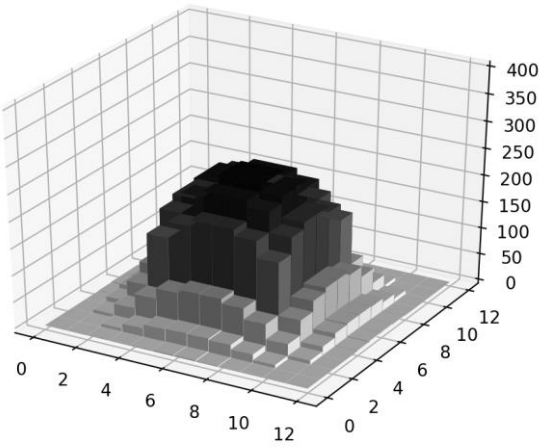
In order to display images as surfaces, the script in Code 1.1 creates images that store the height and the colour for each surface sample. The function `createImageF` creates an array of floats (floating point numbers) and its last parameter defines the number of floats per pixel. In this example, we create a one-dimensional array to store the height value of the surface and a three-dimensional array to store the colour. The arrays are filled by using the data in the input image in two nested loops. Notice that indentation is used to define the scope of each loop and that image data is accessed in (column, row) format. In a grey level image, the maximum value is 255 which corresponds to white. Colour is a complex topic studied next in Section 2.2 and later in detail in Chapter 11 – we need colour here to show the images. We set the height to be larger for brighter pixels. The colour array gives the same value to the red, green and blue components, so it defines values of grey that correspond to the pixel grey value. The script uses the utility function `plot3DColorHistogram` to draw a discrete surface as shown in Figure 1.13(c).



(a) image

```
[   0   0   0   0   1   1   1   1   0   0   0   0 ]
[   0   0   6  16  23  26  26  23  16   6   0   0 ]
[   0   6  27  56  76  84  84  76  56  27   6   0 ]
[   0  16  55 153 185 197 197 185 153  55  16   0 ]
[   0  24  76 185 223 235 235 223 185  76  24   0 ]
[   2  26  83 197 235 246 246 235 197  83  26   2 ]
[   2  26  83 197 235 246 246 235 197  83  26   2 ]
[   0  24  76 185 223 235 235 223 185  76  24   0 ]
[   0  16  55 153 185 197 197 185 153  55  16   0 ]
[   0   6  27  56  76  84  84  76  56  27   6   0 ]
[   0   0   6  16  23  26  26  23  16   6   0   0 ]
[   0   0   0   0   1   1   1   1   0   0   0   0 ]
```

(b) pixel values



(c) discrete surface

Figure 1.12 Python image visualization

```
# Feature Extraction and Image Processing
# Mark S. Nixon & Alberto S. Aguado
# Chapter 1: Image brightening

# Set utility folder
import sys
sys.path.insert(0, '../Utilities/')

# Iteration
from timeit import itertools

# Set utility functions
from ImageSupport import imageReadL, showImageL, printImageRangeL, createImageL

'''
Parameters:
   pathToDir = Input image directory
   imageName = Input image name
   brightDelta = Increase brightness
   printRange = Image range to print
'''
pathToDir = "Input/"
```

```
imageName = "Zebra.png"
brightDelta = 80;
printRange = [0, 10]

# Read image into array and create and output image
inputImage, width, height = imageReadL(pathToDir + imageName)
outputImage = createImageL(width, height)

# Set the pixels in the output image
for x,y in itertools.product(range(0, width), range(0, height)):
    outValue = int(inputImage[y,x]) + brightDelta
    if outValue < 255:
        outputImage[y,x] = outValue
    else:
        outputImage[y,x] = 255

# Show images
showImageL(inputImage)
showImageL(outputImage)

# Print image range
printImageRangeL(inputImage, printRange, printRange)
printImageRangeL(outputImage, printRange, printRange)
```

Code 1.2 Image brightening

The script in Code 1.2 illustrates the way data in an image array is processed. The script uses the function `createImageL` to create a grey level image that contains the processed data. In addition to the name of the image, the script has two variables that are used as parameters. `brightDelta` defines an amount that is added to the input pixel to produce an output pixel and `printRange` defines a range of pixels that are printed out. Similar to Code 1.1, the script iterates through all the values of the input image by addressing the columns from 0 to width, and the rows from 0 to height. However, Code 1.2 defines the two nested loops in a single line by using the function `itertools.product` imported from `timeit`. Both the iterative and the nested loops have the same functionality. This book uses the iterator to perform nested loops to reduce indentation and keep the code clearer. Inside the loop, the output pixels $q(x,y)$ result from adding Δ_{bright} to some input pixels $p(x,y)$ as

$$q(x,y) = p(x,y) + \Delta_{bright} \tag{1.1}$$

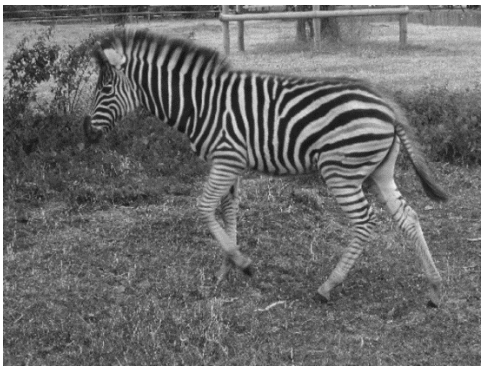
In code, this is computed as the value of the input pixel plus the value of `brightDelta`. There is some jacketing software which prevents values exceeding 255 – if the result of addition exceeds this then the value stops at 255. This is one large difference between describing algorithms as maths and as code – some of the code never appears in the maths but is necessary to make the maths work correctly. Figure 1.14 shows the result of the script. The figure shows the input and the resulting image as well as some of the pixel values in matrix form.

```
[  55  51  55  46  56  58  67  59  58  46  70  ]
[  44  54  62 101  75  66  54  55  71  61  53  ]
[  46  46  57  59  61  33  46  64  74  41  65  ]
[  50  42  44  56  54  52  58  70  65  54  58  ]
[  43  46  44  54  45  47  50  54  54  63  49  ]
[  46  45  55  54  42  52  49  52  50  49  51  ]
[  52  44  32  59  46  46  60  54  40  62  61  ]
[ 121 121 109 119 104  97 102  83  74  83  61  ]
[  66  73  52  46  47  49  48  47  54  49  48  ]
[  54  49  29  53  50  46  48  39  48  49  47  ]
[  29  40  44  52  44  38  40  46  42  44  44  ]
```

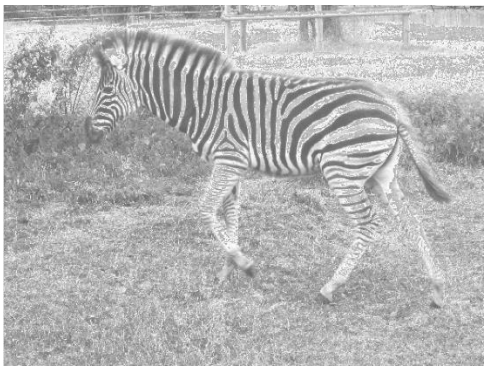
(a) a region of the image

```
[ 135 131 135 126 136 138 147 139 138 126 150  ]
[ 124 134 142 181 155 146 134 135 151 141 133  ]
[ 126 126 137 139 141 113 126 144 154 121 145  ]
[ 130 122 124 136 134 132 138 150 145 134 138  ]
[ 123 126 124 134 125 127 130 134 134 143 129  ]
[ 126 125 135 134 122 132 129 132 130 129 131  ]
[ 132 124 112 139 126 126 140 134 120 142 141  ]
[ 201 201 189 199 184 177 182 163 154 163 141  ]
[ 146 153 132 126 127 129 128 127 134 129 128  ]
[ 134 129 109 133 130 126 128 119 128 129 127  ]
[ 109 120 124 132 124 118 120 126 122 124 124  ]
```

(b) processed values in same region



(c) original image



(d) processed image

Figure 1.13 Processing an image in Python

1.5.4 Mathematical Tools

Mathematica, Maple and Matlab are amongst the most popular of current mathematical systems, and Mathcad was used in earlier editions of this book. There have been surveys that compare their efficacy, but it is difficult to ensure precise comparison due to the impressive speed of development of techniques. Most systems have their protagonists and detractors, as in any commercial system. There are many books which use these packages for particular subjects, and there are often handbooks as addenda to the packages. We shall use Matlab throughout this text, aiming to expose the range of systems that are available. Matlab dominates this market these days, especially in image processing and computer vision, and its growth rate has been enormous; older versions of this text used Mathcad which was more sophisticated (and WYSWYG) but had a more chequered commercial history. Note that there is an open source compatible system for Matlab called Octave. This is of course free. Most universities have a site license for Matlab and without that it can be rather (too) expensive for many. The web links for the main mathematical packages are given in Table 1.3.

General		
Guide to available Mathematical Software	NIST	gams.nist.gov/
Vendors		
Mathcad	Parametric Technology Corp.	mathcad-store.co.uk/
Mathematica	Wolfram Research	wolfram.com/

Matlab	Mathworks	mathworks.com/
Maple	Maplesoft	maplesoft.com/
Matlab Compatible		
Octave	Gnu (Free Software Foundation)	gnu.org/software/octave/

Table 1.3 Mathematical package websites

1.5.5 Hello Matlab

For those who are less enamoured with programming we shall use the Matlab package to accompany Python. Matlab offers a set of mathematical tools and visualisation capabilities in a manner arranged to be very similar to conventional computer programs. The system was originally developed for matrix functions, hence the 'Mat' in the name. There are a number of advantages to Matlab, not least the potential speed advantage in computation and the facility for debugging, together with a considerable amount of established support. There is an image processing toolkit supporting Matlab, but it is rather limited compared with the range of techniques exposed in this text. Matlab's popularity is reflected in a book dedicated to its use for image processing [Gonzalez09], by perhaps one of the subject's most popular authors. It is of note that many researchers make available Matlab versions for others to benefit from their new techniques.

The Octave open source compatible system was built mainly with Matlab compatibility in mind. It shares a lot of features with Matlab such as using matrices as the basic data type, with availability of complex numbers and built in functions as well the capability for user-defined functions. There are some differences between Octave and Matlab, and there is extensive support for both. We shall refer to both systems using Matlab as a general term.

Essentially, Matlab is the set of instructions that process the data stored in a workspace, which can be extended by user-written commands. The workspace stores the different lists of data and these data can be stored in a MAT file; the user-written commands are functions that are stored in M-files (files with extension .M). The procedure operates by instructions at the command line to process the workspace data using either one of Matlab's own commands, or using your own commands. The results can be visualised as graphs, surfaces or as images.

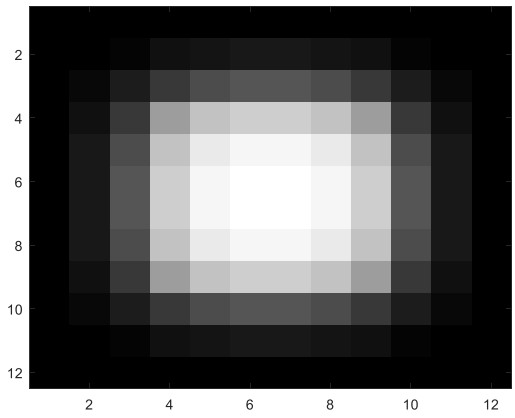
Matlab provides powerful matrix manipulations to develop and test complex implementations. In this book, we avoid matrix implementations in favour of a more C++ algorithmic form. Thus, matrix expressions are transformed into loop sequences. This helps students without experience in matrix algebra to understand and implement the techniques without dependency on matrix manipulation software libraries. Implementations in this book only serve to gain understanding of the techniques' performance and correctness, and favour clarity rather than speed.

Matlab processes images, so we need to know what an image represents. Images are spatial data, which is data that is indexed by two spatial co-ordinates. The camera senses the *brightness* at a point with co-ordinates *x,y*. Usually, *x* and *y* refer to the horizontal and vertical axes, respectively. Throughout this text we shall work in *orthographic projection*, ignoring **perspective**, where real world co-ordinates map directly to *x* and *y* co-ordinates in an image. The *homogeneous co-ordinate system* is a popular and proven method for handling three-dimensional co-ordinate systems (*x*, *y* and *z* where *z* is depth), though it will not be used here as we shall process two-dimensional images only. The brightness sensed by a camera is transformed to a signal which is and stored as a value within the computer, referenced to the co-ordinates *x,y* in the image. Accordingly, a computer image is a matrix of points. For a greyscale image, the value of

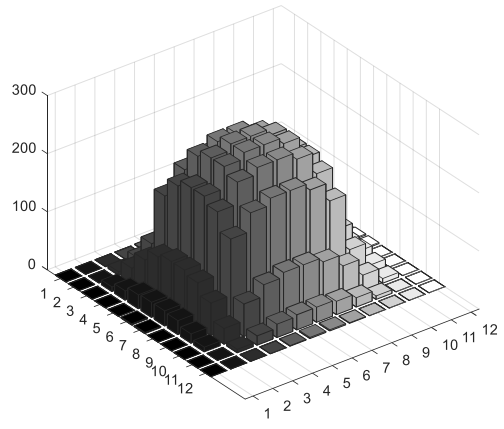
each point is proportional to the brightness of the corresponding point in the scene viewed by the camera. These points are the picture elements, or **pixels.**

0	0	0	0	1	2	2	1	0	0	0	0
0	0	6	16	23	26	26	23	16	6	0	0
0	24	76	185	223	235	235	223	185	76	24	0
0	24	76	185	223	235	235	223	185	76	24	0
0	24	76	185	223	235	235	223	185	76	24	0
2	26	83	197	235	246	246	235	197	83	26	2
2	26	83	197	235	246	246	235	197	83	26	2
0	24	76	185	223	235	235	223	185	76	24	0
0	24	76	185	223	235	235	223	185	76	24	0
0	24	76	185	223	235	235	223	185	76	24	0
0	0	6	16	23	26	26	23	16	6	0	0
0	0	0	0	1	2	2	1	0	0	0	0

(a) set of pixel values



(b) Matlab image



(c) Matlab surface plot

Figure 1.14 Matlab image visualisation

```
% Feature Extraction and Image Processing
% Mark S. Nixon & Alberto S. Aguado
% Chapter 1: Introduction (Hello Matlab)

%read in an image
a=imread('Input\Square.png');
%view image
imagesc(a);
%view it in black and white
disp ('We shall  display the data as a grey level image')
```

```
colormap(gray);
% Let's hold so we can view it
disp ('When you are ready to move on, press RETURN')
pause;
disp ('Now we shall display the data as a surface')
bar3(a);
```

Code 1.3 Example Matlab script

Consider for example, the set of pixel values in Figure 1.14(a). These values were derived from the image of a bright peak on a dark background. The peak is brighter where the pixels have a larger value (here over 200 brightness levels); the background is dark and those pixels have a smaller value (near 0 brightness levels). (We shall consider how many points we need in an image and the possible range of values for pixels in Chapter 2.) The peak can be viewed as an image in Figure 1.14(b), or as a surface in Figure 1.14(c).

As in the Python, there are considerably more sophisticated systems and usage than this version of Matlab code. Like Python in Section 1.5.4, our aim here is not to show people how to use Matlab and all its wonderful functionality. We are using the simplest versions with default settings so that people can process images quickly and conveniently. The Matlab code might appear clearer than the Python code. It is, but that is at the cost of access to data and Python has much greater freedom there. One can get to Manchester from London using a Mini or a Rolls Royce: the style is different (and the effect on the environment is different….) but the result is the same. That is how we shall use Python and Matlab in this textbook.

Matlab runs on Unix/Linux, Windows and Macintosh systems; a student version is available. We shall use a script, to develop our approaches, which is the simplest type of M-file, as illustrated in Code 1.3. To start the Matlab system, type MATLAB at the command line. At the Matlab prompt `(>>) type chapter1` to load and run the script (given that the file `Code_1_3.m` is saved in the directory you are working in). Alternatively, open the directory containing `Code_1_3.m` and click on it to run it. Comments are preceded by a `%`; we first read in an image. Having set the display facility to black and white, we can view the array of points `a` as a surface. When the image, illustrated in Figure 1.14(b), has been plotted, then Matlab has been made to pause until you press `Return` before moving on. Here, when you press `Return`, you will next see the surface of the array, Figure 1.14(c).

```
% Feature Extraction and Image Processing
% Mark S. Nixon & Alberto S. Aguado
% Chapter 1: Invert an image

function inverted = invert(image)

% Subtract image point brightness from maximum
% Usage: new image = invert(old image)
% Parameters: image      - array of points

% get dimensions
[rows,cols]=size(image);

% find the maximum
maxi = max(max(image));

% subtract image points from maximum
for x = 1:cols %address all columns
  for y = 1:rows %address all rows
    inverted(y,x)=maxi-image(y,x);
```

```
   end
end
```

Code 1.4 Matlab function `invert.m` to invert an Image

We can use Matlab's own command to interrogate the data: these commands find use in the M-files that store subroutines. An example routine is called after this. This subroutine is stored in a file called `invert.m` and is a function that inverts brightness by subtracting the value of each point from the array's maximum value. The code is illustrated in Code 1.4. Note that this code uses for loops which are best avoided to improve speed, using Matlab's vectorised operations. The whole procedure can actually be implemented by the command `inverted=max(max(pic))-pic`. In fact, one of Matlab's assets is a 'profiler' which allows you to determine exactly how much time is spend on different parts of your programs. Naturally, there is facility for importing graphics files, which is quite extensive (i.e., it accepts a wider range of file formats). When images are used we find that Matlab has a range of datatypes. We must move from the unsigned integer datatype, used for images, to the double precision datatype to allow processing as a set of real numbers. In these ways Matlab can, and will be used to process images throughout this book, with the caveat that this is not a book about Matlab (or Python). Note that the translation to application code is perhaps easier via Matlab than for other systems and it offers direct compilation of the code. There are some Matlab scripts available at the book's website (users.ecs.soton.ac.uk/msn/book) for on-line tutorial support of the material in this book. There are many other implementations of techniques available on the Web in Matlab. The edits required to make the Matlab worksheets run in Octave are described in the file readme.txt in the downloaded zip.

1.6 Associated Literature

1.6.1 Journals, Magazines and Conferences

As in any academic subject, there are many sources of literature and when used within this text the cited references are to be found at the end of each chapter. The **professional magazines** include those that are more systems oriented, like *Vision Systems Design*. These provide more general articles, and are often a good source of information about new computer vision products. For example, they survey available equipment, such as cameras and monitors, and provide listings of those available, including some of the factors by which you might choose to purchase them.

There is a wide selection of **research journals** - probably more than you can find in your nearest library unless it is particularly well-stocked. These journals have different merits: some are targeted at short papers only, whereas some have short and long papers; some are more dedicated to the development of new theory whereas others are more pragmatic and focus more on practical, working, image processing systems. But it is rather naive to classify journals in this way, since all journals welcome good research, with new ideas, which has been demonstrated to satisfy promising objectives.

The main research journals include: *IEEE Transactions on*: *Pattern Analysis and Machine Intelligence* (in later references this will be abbreviated to *IEEE Trans. on PAMI*); *Image Processing* (*IP*)*; Systems, Man and Cybernetics* (*SMC*); and on *Medical Imaging* (there are many more IEEE transactions, e.g. the *IEEE Transactions on Biometrics, Behavior and Identity Science*, which sometimes publish papers of interest in or application of image processing and computer vision). The *IEEE Transactions* are usually found in (university) libraries since they are available at comparatively low cost; they are online to subscribers at the IEEE Explore site (ieeexplore.ieee.org/) and this includes conferences and IET Proceedings. *Computer Vision and Image Understanding* and *Graphical Models and Image Processing* arose from the splitting of

one of the subject's earlier journals, *Computer Vision, Graphics and Image Processing* (*CVGIP*), into two parts. Do not confuse *Pattern Recognition* (*Pattern Recog.*) with *Pattern Recognition Letters* (*Pattern Recog. Lett.*), published under the aegis of the Pattern Recognition Society and the International Association of Pattern Recognition, respectively, since the latter contains shorter papers only. The *International Journal of Computer Vision* is a more recent journal whereas *Image and Vision Computing* was established in the early 1980s. Finally, do not miss out on the *IET Proceedings Computer Vision* and other journals.

Most journals are now on-line but usually to subscribers only; some go back a long way. Academic Press titles include *Computer Vision and Image Understanding*, *Graphical Models and Image Processing* and *Real-Time Image Processing.*

There are plenty of conferences too: the proceedings of IEEE conferences are also held on the Explore site and two of the top conferences are *Computer Vision and Pattern Recognition* (*CVPR*) which is held annually in the USA, the *International Conference on Computer Vision* (*ICCV*) is biennial and moves internationally. The IEEE also hosts specialist conferences, e.g. on biometrics or on computational photography. *Lecture Notes in Computer Science* are hosted by Springer (springer.com) and are usually the proceedings of conferences. Some conferences such as the *British Machine Vision Conference* series maintain their own site (bmva.org). The excellent Computer Vision Conferences page conferences.visionbib.com/Iris-Conferences.html is brought to us by Keith Price and lists conferences in Computer Vision, Image Processing and Pattern Recognition.

1.6.2 Textbooks

There are many textbooks in this area. Increasingly, there are web versions, or web support, as summarised in Table 1.4. The difficulty is of access as you need a subscription to be able to access the online book (and sometimes even to see that it is available online), though there are also Kindle versions. For example, this book is available online to those subscribing to Referex in Engineering Village engineeringvillage.org. The site given in Table 1.**4** for this book is the support site which includes demonstrations, worksheets, errata and other information. The site given next, at Edinburgh University UK, is part of the excellent CVOnline site (many thanks to Bob Fisher there) and it lists current books as well pdfs of some which are more dated, but still excellent (e.g. [Ballard82]). There is also continuing debate on appropriate education in image processing and computer vision, for which review material is available [Bebis03].

For support material, the *World of Mathematics* comes from Wolfram research (the distributors of Mathematica) and gives an excellent web based reference for mathematics. *Numerical Recipes* [Press07] is one of the best established texts in signal processing. It is beautifully written, with examples and implementation and is on the web too. *Digital Signal Processing* is an online site with focus on the more theoretical aspects which will be covered in Chapter 2. *The Joy of Visual Perception* is an online site on how the human vision system works. We haven't noted *Wikipedia* – computer vision is there too.

By way of context, for comparison with other textbooks, this text aims to start at the foundation of computer vision, and to reach current research. Its content specifically addresses techniques for image analysis, considering feature extraction and shape analysis in particular. Matlab is used as a vehicle to demonstrate implementation. There are of course other texts, and these can help you to develop your interest in other areas of computer vision.

This book's homepage	Southampton University	users.ecs.soton.ac.uk/msn/book
CVOnline – online book compendium	Edinburgh University	homepages.inf.ed.ac.uk/rbf/CVonline/books.htm

World of Mathematics	Wolfram Research	mathworld.wolfram.com
Numerical Recipes	Cambridge University Press	numerical.recipes
Digital Signal Processing	Steven W. Smith	dspguide.com
The Joy of Visual Perception	York University	http://www.yorku.ca/research/vision/eye/thejoy.htm

Table 1.4 Web textbooks and homepages

Some of the main textbooks are now out of print, but pdfs can be found at the CVOnline site. There are more than given here, some of which will be referred to in later Chapters; each offers a particular view or insight into computer vision and image processing. Some of the main textbooks include: *Vision* [Marr82] which concerns vision and visual perception (as previously mentioned); *Fundamentals of Computer Vision* [Jain89] which is stacked with theory and technique, but omits implementation and some image analysis, as does *Robot Vision* [Horn86]; *Machine Vision* [Jain95] offers concise and modern coverage of 3D and motion; *Digital Image Processing* [Gonzalez17] has more tutorial element than many of the basically theoretical texts and has a fantastic reputation for introducing people to the field; *Digital Picture Processing* [Rosenfeld82] is very dated now, but is a well-proven text for much of the basic material; and *Digital Image Processing* [Pratt01], which was originally one of the earliest books on image processing and, like *Digital Picture Processing*, is a well-proven text for much of the basic material, particularly image transforms. Despite its name, *Active Contours* [Blake98] concentrates rather more on models of motion and deformation and probabilistic treatment of shape and motion, than on the active contours which we shall find here. As such it is a more research text, reviewing many of the advanced techniques to describe shapes and their motion. *Image Processing - The Fundamentals* [Petrou10] (by two Petrous!) surveys the subject (as its title implies) from an image processing viewpoint. *Computer Vision* [Shapiro01] includes chapters on image databases and on virtual and augmented reality. *Computer Vision: A Modern Approach* [Forsyth11] offers much new – and needed - insight into this continually developing subject (two chapters that didn't make the final text – on probability and on tracking – are available at the book's website luthuli.cs.uiuc.edu/~daf/book/book.html). One text [Brunelli09] focusses on object recognition techniques "employing the idea of projection to match image patterns" which is a class of approaches to be found later in this text. A much newer text *Computer Vision: Algorithms and Applications* [Szeliski11] is naturally much more up to date than older texts, and has an online (earlier) electronic version available too. An even newer text *Computer Vision Models, Learning, and Inference* [Prince12] – electronic version available - is based on models and learning. One of the bases of the book is "to organize our knowledge …what is most critical is the model itself - the statistical relationship between the world and the measurements" and thus covers many of the learning aspects of computer vision which complement and extend this book's focus on feature extraction.

Also, Kasturi, R., and Jain, R. C. Eds.: *Computer Vision: Principles* [Kasturi91a] and *Computer Vision: Advances and Applications* [Kasturi91b] presents a collection of seminal papers in computer vision, many of which are cited in their original form (rather than in this volume) in later chapters. There are other interesting edited collections [Chellappa92], one edition [Bowyer96] honours Azriel Rosenfeld's many contributions.

Section 1.5 describes some of the image processing software packages available, and their textbook descriptions. Of the texts with a more practical flavour *Image Processing and Computer*

Vision [Parker10] includes description of software rather at the expense of lacking range of technique. There is excellent coverage of practicality in *Practical Algorithms for Image Analysis* [O'Gorman08]. One JAVA text, *The Art of Image Processing with Java*, [Hunt11] emphasises software engineering more than feature extraction (giving basic methods only).

Other textbooks include: the longstanding *The Image Processing Handbook* [Russ17] which contains much basic technique with excellent visual support, but without any supporting theory; *Computer Vision: Principles, Algorithms, Applications, Learning* [Davies17] which is targeted primarily at (industrial) machine vision systems but covers much basic technique, with pseudo-code to describe their implementation; and the *Handbook of Pattern Recognition and Computer Vision* [Cheng16] covers much technique. There are specialist texts too and they usually concern particular Sections of this book, and they will be mentioned there. Last but by no means least, there is even a (illustrated) dictionary [Fisher2013] to guide you through the terms that are used.

1.6.3 The Web

This book's homepage (ecs.soton.ac.uk/~msn/book/) details much of the support material, including worksheets, code and demonstrations, and any errata we regret have occurred (and been found). The Computer Vision Online CVOnline homepage dai.ed.ac.uk/CVonline/ has been brought to us by Bob Fisher from the University of Edinburgh. There's a host of material there, including its description. If you have access to the web-based indexes of published papers, the Web of Science covers a lot and includes INSPEC which has papers more related to engineering, together with papers in conferences. Explore is for the IEEE – with paper access for subscribers; many researchers turn to Google Scholar as this is freely available with ability to retrieve the papers as well as to see where they have been used. Many use Scholar since it also allows you to see the collection (and metrics) of papers in authors' pages. The availability of papers within these systems has been changing with the new Open Access systems, which at the time of writing is still in some flux.

1.7 Conclusions

This Chapter has covered most of the pre-requisites for feature extraction in image processing and computer vision. We need to know how we see, in some form, where we can find information and how to process data. More importantly we need an image, or some form of spatial data. This is to be stored in a computer and processed by our new techniques. As it consists of data points stored in a computer, this data is sampled or discrete. Extra material on image formation, camera models and image geometry is to be found in Chapter 10, but we shall be considering images as a planar array of points hereon. We need to know some of the bounds on the sampling process, on how the image is formed. These are the subjects of the next Chapter which also introduces a new way of looking at the data, how it is interpreted (and processed) in terms of frequency.

1.8 Chapter 1 References

[Armstrong91] Armstrong, T., *Colour Perception - A Practical Approach to Colour Theory*, Tarquin Publications, Diss UK, 1991

[Ballard82] Ballard, D. H., Brown, C. M., *Computer Vision*, Prentice-Hall Inc New Jersey USA, 1982

[Bebis03] Bebis, G., Egbert, D., and Shah, M., Review of Computer Vision Education, *IEEE Transactions on Education*, **46**(1), pp. 2-21, 2003

[Blake98] Blake, A., and Isard, M., *Active Contours*, Springer-Verlag London Limited, London UK, 1998

[Bowyer96] Bowyer, K., and Ahuja, N., Eds., *Advances in Image Understanding, A Festschrift for Azriel Rosenfeld*, IEEE Computer Society Press, Los Alamitos, CA USA, 1996

[Bradski08] Bradski, G., and Kaehler, A., *Learning OpenCV: Computer Vision with the OpenCV Library*, O'Reilly Media, Inc, Sebastopol, CA USA, 2008

[Bruce90] Bruce, V., and Green, P., *Visual Perception: Physiology, Psychology and Ecology*, 2nd Edition, Lawrence Erlbaum Associates, Hove UK, 1990

[Chellappa92] Chellappa, R., *Digital Image Processing*, 2nd Edition, IEEE Computer Society Press, Los Alamitos, CA USA, 1992

[Cheng16] Cheng, C. H., and Wang, P. S. P., *Handbook of Pattern Recognition and Computer Vision***,** 5th Edition, World Scientific, Singapore, 2016

[Cornsweet70] Cornsweet, T. N., *Visual Perception*, Academic Press Inc., N.Y. USA, 1970

[Davies17] Davies, E. R., *Computer Vision: Principles, Algorithms, Applications, Learning*, Academic Press, 5th Edition, 2017

[Fisher2013] Fisher, R. B., Breckon, T. P., Dawson-Howe, K., Fitzgibbon, A., and Robertson, C., Trucco, E., and Williams C. K. I., *Dictionary of Computer Vision and Image Processing*, Wiley, Chichester UK, 2nd Ed 2013

[Forsyth11] Forsyth, D., and Ponce, J., *Computer Vision: A Modern Approach*, Prentice Hall, N.J. USA, 2nd Edition 2011

[Fossum97] Fossum, E. R., CMOS Image Sensors: Electronic Camera-On-A-Chip, *IEEE Trans. Electron Devices*, **44**(10), pp1689-1698, 1997

[Gonzalez17] Gonzalez, R. C., and Woods, R. E., *Digital Image Processing*, 4th Edition, Pearson Education, 2017

[Gonzalez09] Gonzalez, R. C., Woods, R. E., and Eddins, S. L., *Digital Image Processing using MATLAB*, Prentice Hall, 2nd Edition, 2009

[Horn86] Horn, B. K. P., *Robot Vision*, MIT Press, Boston USA, 1986

[Hunt11] Hunt, K. A., *The Art of Image Processing with Java*, CRC Press (A. K. Peters Ltd.), Natick MA USA, 2011

[Jain89] Jain A. K., *Fundamentals of Computer Vision*, Prentice Hall International (UK) Ltd., Hemel Hempstead UK, 1989

[Jain95] Jain, R. C., Kasturi, R., and Schunk, B. G., *Machine Vision*, McGraw-Hill Book Co., Singapore, 1995

[Kasturi91a] Kasturi, R., and Jain, R. C., *Computer Vision: Principles*, IEEE Computer Society Press, Los Alamitos CA USA, 1991

[Kasturi91b] Kasturi, R., and Jain, R. C., *Computer Vision: Advances and Applications*, IEEE Computer Society Press, Los Alamitos CA USA, 1991

[Kuroda14] Kuroda, T, *Essential Principles of Image Sensors*, CRC Press, Boca Raton FL USA, 2014

[Livingstone14] Livingstone, M. S., *Vision and Art* (Updated and Expanded Edition), Harry N. Abrams, New York USA, 2nd Edition 2014

[Lutz13] Lutz, M., *Learning Python*, O'Reilly Media, Sebastopol CA USA, 5th Edition 2013

[Marr82] Marr, D., *Vision*, W. H. Freeman and Co., N.Y. USA, 1982

[Nakamura05] Nakamura, J., *Image Sensors and Signal Processing for Digital Still Cameras*, CRC Press, Boca Raton FL USA, 2005

[O'Gorman08] O'Gorman, L., and Sammon, M. J., Seul, M., *Practical Algorithms for Image Analysis*, 2nd Edition, Cambridge University Press, Cambridge UK, 2008

[Overington92] Overington, I., *Computer Vision - A Unified, Biologically-Inspired Approach*, Elsevier Science Press, Holland, 1992

[Parker10] Parker, J. R., *Algorithms for Image Processing and Computer Vision*, Wiley, Indianapolis IN USA, 2nd Edition 2010

[Petrou10] Petrou, M., and Petrou, C., *Image Processing - The Fundamentals*, Wiley-Blackwell, London UK, 2nd Edition, 2010

[Phillips18] Phillips J. B., and Eliasson, H., *Camera Image Quality Benchmarking*, Wiley, Hoboken NJ USA, 2018

[Pratt01] Pratt, W. K., *Digital Image Processing: PIKS Inside*, Wiley, 3rd Edition, 2001

[Press07] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P., *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, Cambridge University Press, Cambridge UK, 3rd Edition, 2007

[Prince12] Prince, S. J. D., *Computer Vision Models, Learning, and Inference*, Cambridge University Press, Cambridge UK, 2012

[Ratliff65] Ratliff, F., *Mach Bands: Quantitative Studies on Neural Networks in the Retina*, Holden-Day Inc., San Francisco USA, 1965

[Rosenfeld82] Rosenfeld, A., and Kak, A. C., *Digital Picture Processing*, 2nd Edition, Vols. 1 and 2, Academic Press Inc., Orlando FL USA, 1982

[Russ17] Russ, J. C., *The Image Processing Handbook*, 7th Edition, CRC Press (Taylor & Francis), Boca Raton FL USA, 2017

[Shapiro01] Shapiro, L. G., and Stockman, G. C., *Computer Vision*, Prentice Hall, 2001

[Solem12] Solem, JE, *Programming Computer Vision with Python: Tools and Algorithms for Analyzing Images*, O'Reilly Media Newton Mass. USA, 2012

[Szeliski11] Szeliski, R., *Computer Vision: Algorithms and Applications*, Springer Verlag, London UK, 2011

1 Introduction 1