The University of Reading

Colour object recognition using a complex colour representation and the frequency domain

Thesis submitted for the Degree of Doctor of Philosophy

A L Thornton BEng AMIEE

Department of Engineering

May 1998

Acknowledgements

I would like to thank my supervisor, Dr. S.J. Sangwine, for his enthusiasm and guidance during the course of the research and also for his patience while the thesis was being written. I would also like to thank my family for their support and help and my employer, Sundance Multiprocessor Technology Ltd., who kindly allowed me to work irregular hours so that I could finish writing up the thesis.

This research was supported by the University of Reading Research Endowment Trust Fund.

Abstract

This thesis documents the research into a method for object recognition in colour images based on a new method for coding the chromatic information for each pixel. It is based on the use of Fourier transforms to generate information about an object. It is of use in situations where the monochromatic recognition of objects either fails or is not applicable. In addition, for the processing described in this thesis, it overcomes problems with Fourier transforming separate colour components which is the traditional method used in Fourier transforming colour images.

The research described in this thesis has used Fourier transforms to correlate two colour images. The colour is encoded so that the chromatic information is stored as a complex number for each pixel. This is made possible by storing the colour not as RGB values but as hue and saturation values so that only two components are required to describe the chromaticity, while intensity is kept separate. This enables the location of colour objects using phase correlation, where previously it would either have been impossible or would have required more Fourier transforms to have been calculated. This processing is combined with the use of the Fourier-Mellin transform to enable the quantification of the translation, rotation and scaling of an object in an image in comparison with an object in a reference image. This has obvious uses in industrial applications. The Fourier-Mellin transform can be implemented using Fourier transforms and log-polar sampling of the pixels. Unfortunately, this non-linear sampling can reduce the accuracy of the phase correlation peak which determines the scaling and rotation values so a novel spatially variable filter has been developed to reduce the possibility of the log-polar sampling ignoring pixels containing valuable data.

The use of the new representation for colour together with Fourier transform techniques has resulted in the use of quaternions for storing full colour. The supervisor of the author has developed a quaternion discrete Fourier transform (QDFT) which has been proven to convolve a colour image successfully. However, using the current QDFT with correlation produces a correct peak but also three other incorrect peaks. Much further work is needed if correlation is to be implemented using quaternions and other applications are to be explored.

Contents

List of Abbreviations

ADC	Analog-to-Digital Converter
CCD	Charge coupled device
CID	Charge injection device
CIE	Commission Internationale de L'Éclairage
DAC	Digital-to-Analog Converter
FFT	Fast Fourier Transform
FT	Fourier Transform
FMT	Fourier-Mellin Transform
HSI	Hue, Saturation and Intensity
HT	Hartley Transform
i	Square root of –1
j	Square root of –1
LPT	Log-Polar Transform
O( )	Order of ( )
QDFT	Quaternion Discrete Fourier Transform
rgb	Red, Green and Blue
RGB	Red, Green and Blue
XYZ	CIE standard colour space

1. INTRODUCTION

The use of image processing has developed over the past few decades so that it is hard to escape its influences in our daily lives. Apart from the development of new techniques, caused by the requirement of new applications and through research, one of the other major factors in this revolution is the decrease in cost of the hardware, such as processors, memory and cameras, required to perform the processing of images. In comparison with monochrome images, the use of colour in image processing generally requires more complicated cameras, extra processing and memory, and it is only relatively recently that its use in industrial applications has grown. This explains why, although the literature concentrating on colour image processing is a fast expanding field, there is far more published literature concerned with monochrome processing.

Areas of image processing research include image coding, image analysis and interpretation, segmentation and texture analysis, motion estimation, neural networks, 3D vision and of course colour. Developments in these and other areas have led to the use of image processing for applications as diverse as the sorting of pharmaceutical pills and medical imaging through to vision systems for missiles, number plate recognition and photogrammetry. This thesis does not explore many of the areas mentioned above because the research presented in it is concerned with pattern recognition and colour.

The manipulation of images can be split into two groups; these are: low-level processing, which does not require *a priori* knowledge of an image and includes compression, pre-processing for filtering, edge extraction, sharpening and segmentation; and high-level processing which could also be called *pattern recognition*, which uses knowledge about an image to aid and determine its processing for the interpretation of scenes.

1.1 RESEARCH OBJECTIVES

The research presented in this thesis is concerned with object recognition using colour data to provide useful additional information which would not be found in monochrome images. Apart from the ability to recognise objects in a coloured image, one requirement

was to make maximum use of the information presented to the processor and not waste unnecessary time in processing a coloured image. Much supposedly colour processing is actually monochromatic processing repeated for the three components which generally make up a colour image.

Before discussing the field of the thesis, the use of colour and its specification is introduced below. There is then a brief survey of some object recognition issues before the issues in the thesis are discussed.

1.1.1 COLOUR

It has been estimated, [Judd and Wyszecki 1975], that the number of different colours which humans can distinguish is around ten million[1.1]. Many imaging systems presently in use ignore the useful contribution that colour can make in assisting the processing of an image to obtain a required result. However, even if the colour information is used, in many cases the processing just uses monochromatic techniques and operates separately on the individual colour components. This, it could be argued, is not truly 'colour processing' and requires more time to process than a single monochromatic signal. The aim of colour processing should be to improve the required processing in a way that monochrome images cannot, and not reduce processing speed by requiring the colour to be split up and operated on separately. It is also the case that the processing of the individual colour components does not produce correct output for many algorithms.

Colour has been used to help visualise mathematical data so that shading colour maps are often used to help visualise surface shape and functional colour maps are used to map functional values into colours. The segmentation of coloured objects is aided if the colour information is used in processing; if an image contains objects which have the same intensity but different colour, normal monochrome processing of the intensity will fail to differentiate between the two objects but if the colours are taken into account, the objects will be seen to be different. Much work has been done on the segmentation of

[1.1] This compares only 2 colours at once. Within an image the number of colours distinguished can fall to around 30,000 [Moorhead II and Zhu 1995].

coloured images but there is comparatively little published on recognition using colour and this thesis addresses this topic.

1.1.1.1 HUMANS AND COLOUR

At this stage it should be understood what is meant by ‘colour’. The visible range of light wavelengths for humans is between 380-780 nm. The retina in the human eye contains four types of photoreceptors which absorb light. Rods allow monochromatic vision under low levels of illumination and are most sensitive to light in the blue-green part of the spectrum. The three types of cone are sensitive to different parts of the visible spectrum which can roughly be described as the colours red, green and blue which are called the primary colours. The human eye is most sensitive to green followed by red and then blue. The range of colours which humans can see is generated by different proportions of signals from the three types of cones. Although many processing systems and displays use red, green and blue values to represent colour there are other (sometimes more suitable) representations. However, according to trichromatic theory first suggested by Young [Young 1807], three values are required to specify colour.

In computer graphics, the three components are often red, green and blue, (RGB), values. However, using these signals to describe and manipulate colours is not very intuitive to humans. Colour can also be thought of as consisting of three quantities which are known as hue, saturation and lightness (reflecting) or brightness (self-luminous). These terms, which can be thought of as the way humans perceive colour, are also defined in colorimetry as the dominant wavelength, excitation purity and intensity (or luminance). Hue indicates the perceived colour and it can be described, for instance, as red, green, yellow or blue. Saturation indicates the amount of a hue with reference to white so that pink is only slightly saturated but a vivid red is very saturated. Intensity refers to an achromatic value which has no saturation and no hue.

Hue can be described quantitatively by a value whose range is between 0 and 360 degrees, where the range is continuous so that a hue outside this range is identical with the hue modulo 360; for instance, the same hue (pure red) is described by 0 and 360

degrees. This allows hue to have no discontinuities caused by values outside this range which can be a great advantage when processing colour. Saturation and intensity however, are not continuous scales so care must be taken when processing to ensure that the values stay within valid ranges and are not clipped to remain within bounds.

1.1.1.2 SPECIFYING COLOUR

The requirement for objectivity in specifying colours led, in 1931, to the Commission Internationale de L'Éclairage (CIE) adopting standard response curves [Wyszecki and Stiles 1982] for colour-matching for a hypothetical Standard Colorimetric Observer[1.2]. The wavelengths of red, green and blue used for defining the CIE 1931 Standard Colorimetric Observer are 700 nm, 546.1 nm and 435.8 nm respectively. Any colour in this visible range can be specified from a combination of the RGB (tristimulus) values since the spectral response curves for each of these R, G and B colours overlap each other by varying amounts. The amounts of R, G and B required to generate a colour are denoted as colour matching functions $\bar{r}$, $\bar{g}$ and b. However, for some wavelengths, certain values are negative so the CIE introduced new XYZ primary stimuli whose (positive only) values can be obtained by the use of linear equations using the values of $\bar{g}$ and b to obtain new colour matching functions $\bar{}$, $\bar{y}$ and $\bar{z}$.

The important next step is to decide how to represent colour. The CIE XYZ representation does not represent perceived colours uniformly and although it can be used to standardise values required to generate colours it is usual to process colours in other colour spaces which might be more useful. The representation of colour is an important consideration, [Connolly and Leung 1995], as each colour representation has its own characteristics and successful processing depends to a certain extent on what process is to be performed on the colour information and what resultant information is

[1.2] The colour-matching function was updated by the CIE in 1964 using a wider visual field and different wavelengths for the RGB primary stimuli curves [Wyszecki and Stiles op. cit. 1982 p.142].

required. Although this will be discussed in more detail in the next chapter, the RGB representation, which is what many computer users are familiar with and is also the form in which data is input to the processing algorithms described in this thesis, is now introduced.

1.1.1.3 THE RGB COLOUR SPACE

This is an additive system so that when the red (R), green (G) and blue (B) components are at their maximum values the resulting output is white. Intensity is calculated as the average of the R, G and B values whereas *unscaled* intensity which has a greater range sums the components. If all the components are zero, the resultant output is black. If R, G and B are equal then the colour being specified is monochromatic, has an undefined hue, a saturation value of zero and lies on what is known as the 'grey' line connecting black to white since it has no colour content. If the RGB components are unequal there is a defined hue and a saturation value larger than zero.

Many processing systems operate separately on the R, G and B values in order to perform colour processing. In addition, if certain colours only were required to be operated on, it would require the specification of three values to define the colour which can be a disadvantage. However, since the capture and display of colour data is usually accomplished using R, G and B data, knowledge of this colour space is necessary and its limitations and constraints during conversion to and from other colour spaces must be understood; this is discussed in chapter 2.

1.1.2 OBJECT RECOGNITION

One of the goals in machine vision is to develop hardware and software to the stage where a machine can behave as though it were human. One of the obvious uses of computers in relation to humans is in the area of vision. This is evident in computer vision where image processing solutions are becoming more prevalent as the answer to vision problems. A recent example, designed by Racal, is the installation of a vehicle

license plate recognition system by the police which is able to recognise and compare many more number plates than a human sitting at a screen could cope with.

Vision is one of the more versatile of the human senses. It would be impossible to attempt to mimic human vision by treating it as a whole and so the challenge is split into smaller problems that are, individually, easier to solve. Humans view the world in a 3-dimensional (3D) space but it has been suggested that the 3D space is generated using 2-dimensional (2D) views [Bulthoff and Edelman 1992]. Object recognition is an important part of the vision process and this is reflected in the numerous applications for which it is used; however, the recognition process is not without problems.

1.1.2.1 OBJECT RECOGNITION ISSUES

Geometrical and perspective changes can badly affect the ability to recognise objects unless the possibility of distortions due to these changes is taken into account. Other factors which affect recognition success are occlusion, noise, lighting changes and distortion. These problems are described in the next few sections.

1.1.2.1.1 VARIANCE DUE TO OBJECT MOVEMENT

Translation variances occur when an object in an image is moved in relation to its expected position. This can be caused either by the camera moving vertically or horizontally but not forwards or backwards or by the object moving parallel to the focal plane of the camera. Rotation variances cause an object to be a rotated version to that expected by the recognition algorithm. This can be caused by the object rotating about the axis which is perpendicular to the camera. Scaling results in an object appearing larger or smaller than its reference version and can be produced either by moving the focal plane of the camera forwards or backwards, for instance by using a zoom lens or by the object moving perpendicularly to the focal plane of the camera. Perspective changes cause the viewed object to distort in relation to the expected object and can be due to the camera being moved so the viewing angle changes.

1.1.2.1.2 OCCLUSION

Occlusion is caused when an object of interest is partly hidden behind another object which is of no concern. Whether this occurs or not depends on the situation; some scenes may be such that there is no possibility of occlusion. However, when it does occur it can be a great problem and some recognition schemes are more successful than others in dealing with it.

1.1.2.1.3 NOISE

Random noise results in, as the name suggests, a random distribution of pixels having incorrect values and is caused by thermal noise in the camera and other electronics such as a frame-grabber. Quantization noise is generated due to the discrete nature of the pixel values which are output from the camera via the ADC. Analogue values of light have to be converted into discrete values so that the computer can work with the pixel values. Unfortunately, when the analogue values are sampled it may be that there is no discrete value exactly the same as the analogue value and so errors are caused when the nearest discrete value is chosen to represent the similar analogue value. This problem can be lessened by increasing the resolution so that more discrete levels are used to represent the analogue values. This follows since the more levels there are, the less the error when converting an analogue value to its nearest discrete level.

1.1.2.1.4 LIGHTING AND COLOUR

If the lighting is not carefully set up and monitored to stay within a required range this can result in errors in recognition. The light falling on objects could be different to that expected and so the processing could fail. For instance, the light could be much brighter or darker or the lighting could create highlights and shadows which in a monochrome scene could result in edges being detected along the highlight or shadow edge leading to a scene containing many more edges and much more confusion. This is more easily dealt with in a colour processing system which could detect edges using hue information too. However, even then care has to be taken as light reflected from coloured objects and

falling on a scene could cause errors in a simple recognition scheme using colour. In addition, the colour temperature of the lighting could be different which again causes errors.

1.1.2.1.5 DISTORTION

In some practical situations it is possible that the object to be identified is damaged in some way or deformed (physically or by the lens) and a practical object recognition system should be able to deal with this situation.

1.1.3 RESEARCH OBJECTIVES

Having seen that the field of colour image processing is not as advanced as it has the potential to be, this section introduces the areas involved in the research described in this thesis. Although there are many possible methods which can be used as the basis for object recognition, the decision was taken to concentrate on Fourier transforms (FTs) of images. The rationale behind this is that the transform is a very powerful mathematical tool for converting images from a spatial domain to a spatial-frequency domain (the 2D equivalent of time and frequency respectively), with no loss to the data content so that the transformed data can be inverse transformed to become identical with the original images. Although many researchers use the spatial domain data only, this ignores the important and useful information which can be found in the spatial-frequency content. It is this information which is to be used in the research presented here. The FT is also ‘safe’ in that much is known about the properties of the FT and as long as certain constraints are noted and adhered to, it is well behaved. In addition, there is a useful literature base of monochrome FT processing. There are, of course, many other transforms which could be used in object recognition. Some of them are mentioned in chapter 2 together with an explanation as to why they were not used in the research in this thesis.

Having decided that Fourier transforms were to be used, attention was paid to the colour representation requirement. The input to the processing system is in the form of the three

RGB components. One common method used to transform colour images is to separately transform the three signals. This results in three times the processing time in comparison with a monochrome image and this is undesirable. It may also be the case that separately processing the transformed data will not generate correct results. The FT has a complex input, so for real input only values (such as a monochrome signal) there is redundancy in the computation since the imaginary part is set to zero. Although a Hartley transform can be used to exploit the symmetry in the result for real data only inputs, and so reduce the processing time for the three individual RGB components, this would not help in *colour* processing.

Therefore, a method had to be found to represent colour in such a way that only one FT is required for a colour image. The decision was taken to use a novel colour representation so that a complex number (Z) represents the chromaticity values of hue and saturation whilst retaining the intensity (I) as a separate value and FT the Z values when the FT of colour images was required.

The next stage in the research was to use suitable processing algorithms once the FT of the data had been obtained. Two processes were identified as being of interest in their use with the complex number representation and Fourier transforms. Phase correlation and Cepstrum processing are used in this research to generate information about the position of a *coloured* object in an image. This is in contrast to the situation where there may be many different coloured objects in an image having similar intensities which would create recognition problems to a recognition system based on monochrome (intensity) processing.

The processing mentioned above is only able to remove translational variance and not the other variances due to object movement so another transform, the log-polar transform, is used to convert rotation and scaling variances to translations. However, a novel filtering algorithm is incorporated into the transform stage which can significantly reduce errors due to the inherent nature of the log-polar sampling. Once this stage is completed, another phase correlation can be calculated to determine the rotation and scaling of the object. In this way, a coloured object can be successfully tracked among other coloured objects as it is translated, rotated and scaled in an image.

1.2 ORGANISATION OF THESIS

Following this introductory chapter which explains the need for colour object recognition and the objectives of the research, there is a chapter giving background information on the areas which are involved in object recognition and colour. There then follows a chapter which focuses more closely on previous research in those areas of interest which will be used in attaining the research objectives. Once this has been presented, chapters 4 to 7 move on to deal with the novel aspects of the research presented in this thesis. Chapter 4 introduces a novel representation for colour and explains methods for conversion to and from other colour spaces together with practical considerations; chapter 5 deals with finding the placement of a coloured object in an image using this new representation for colour; chapter 6 uses this information in a recognition system, explains problems with this system and introduces a novel method to overcome them. The final chapter describing the research, chapter 7, introduces the use of quaternion hypercomplex numbers in a recognition system.

Following this, the final chapter discusses the research and results and then proposes further work which could lead on from the research presented in chapters 4 to 7.

2. GENERAL THEORY

This chapter describes the hardware generally required in an image processing system and then introduces some of the theory concerning colour, in particular the RGB and HSI colour spaces. Following this, the Fourier transform and Mellin transform are introduced and some of their properties which are used in later chapters to obtain colour object recognition are explained. The chapter following this then discusses the use made of this processing in the literature.

2.1 IMAGE PROCESSING HARDWARE

In general, image processing systems consist of a camera, capture board, processor and monitor, so some of the individual components will now be briefly discussed.

2.1.1 CAMERA

The 1960s onwards saw the development of solid-state imaging using photodiode arrays, charge-coupled devices (CCDs) and charge injection devices (CIDs) [Tompsett 1979]. More usually used in a line arrangement rather than as an area array, an array of photodiodes together with control circuitry are contained on a chip. Due to the amount of control circuitry, the size of the arrays is limited in practice.

2.1.1.1 CHARGE-COUPLED DEVICES (CCDS)

CCDs are created using light sensitive silicon which has a rectangular array of sensing sites (wells). These sensing sites are potential wells (representing pixels) in the silicon and when the charge due to light falling on the well is to be read out, the charge in each well is shifted along the wells until it reaches the output. For area CCDs there are a number of architectures for the charge data to be read out, known as full-frame, interline transfer and frame transfer [Castleman 1996]. The architecture affects the speed of frame rate and also the size of silicon required since the latter two architectures enable simultaneous integration and readout but require more area.

The generation of colour signals has been performed using prisms and dichroic beam splitters to separate light into its red, green and blue components which are then detected using three image sensors. This involves precise positioning of the sensors so that mis-alignment of the colours does not occur. However, with the use of CCDs this is not a problem once the initial positioning has been done. Colour can be detected using 3 CCDs or one CCD together with an array of dichroic or stripe filters on the sensor; the use of 3 CCDs reduces colour smearing.

CCDs are not perfect however and it is wise to have some knowledge as to their uses and limitations.

i. Their integration time is flexible (from frame rate to hours) with the constraint that the temperature of the sensor is kept below room temperature. This is due to dark currents filling wells with thermal electrons. However, the effect varies from well to well due to crystal lattice imperfections. Fortunately, unless the well is saturated by the dark current, the effect can be overcome by subtracting the known dark current pattern from an image.
ii. As the charge is read out of a CCD array, readout noise can affect charge values due to the sensor circuitry. The effect is more obvious when the charges are low and when the transfer rate of the charge is higher.
iii. When charge is transferred from well to well, if the transfer of charge is not 100% efficient the readout of charge will not be totally correct.
iv. Blooming is caused by the sensor being overexposed so that charge leaks into neighbouring wells, or pixels.
v. In addition, if there are errors in the crystal lattice which mean that photoelectrons cannot be stored, this prevents any pixel charges being transferred via that pixel from being readout and so at its worst can affect an entire column.

2.1.1.2 CHARGE INJECTION DEVICES (CIDS)

CIDs are similar to CCDs in that they use the light sensing properties of silicon. However, it is in the readout method that differences can be found. It is possible to read the charges accumulating in the wells without destroying the charges in their pixel

positions. This is unlike CCDs where reading the charge results in an individual pixel no longer retaining the charge belonging to it due to charge transfers. This is possible because the CID contains two adjacent potential wells per pixel, each connected to a different electrode. The arrangement of the sensors in the CID means that they can either be individually addressed or they can all be addressed at once. When both the electrodes are turned on integration is started; if only one is on the accumulated charge is shifted to its neighbouring well which creates a pulse in the second electrode and therefore the external circuitry. The size of the pulse is proportional to the transferred charge and this is how pixel values can be read out without destroying the charge in a pixel. If both electrodes are turned off, the charge is removed to leave an empty well but a pulse is still produced and this is a method for reading out charge and destroying data (as with CCDs).

This non-destructive readout method enables images to be stored and seen and is of use if the integration time is not known. This mechanism means that CIDs are not so liable as CCDs to blooming since adjacent pixels do not have connected wells. Therefore CIDs are also better in extreme lighting situations and this is also due to the ability to check individual pixels and clear them if they saturate. CIDs do not have the problem of efficient charge-transfer or errors caused by the effects of having a dead pixel. However, they are less sensitive to light than CCDs.

2.1.2 CALIBRATION

Accurate calibration of the video camera in a colour system is very important. Frey and Palus [1993] presented a detailed discussion of the calibration method, consisting of white and black balance, linearization and colour calibration. Firstly, a coarse white balance is performed by adjusting the RGB values and then the black value is measured for each RGB channel. Then the dynamic range is changed so that the white value is inside the dynamic range of the camera and processing system. The white-black balance is then set so that white is represented by a value of 255 for each R, G and B component and black is represented by 0 for each of the components. Linearization is then conducted with the use of a look-up table using values determined using known filters.

However, by this stage, only the correct measurement of achromatic objects has been achieved so the final step is colour balance, the process of which is similar to linearization. A look-up table can be used and its values found using colour charts so that the measured colour values are transformed to the correct values.

2.1.3 IMAGE CAPTURE

Modern frame stores or capture boards are much more sophisticated than early designs and can store more bits per pixel (to reduce quantization errors) and capture colour data from more than one source for real-time 3D processing. Essentially, they consist of an analog-to-digital converter (ADC), memory and some addressing control, and a digital-to-analog converter (DAC). Due to sampling theory, it is important that the sampling frequency of the ADC (and the DAC for display) is high enough to allow an adequate spatial resolution of an image so that data is not missed. Another important parameter of the ADC / DAC is the number of bits used to represent an analogue signal which determines the maximum level of quantization noise possible.

2.1.4 PROCESSING STAGE

Having converted the data into a digital form it is suitable for processing. With the emergence of microprocessors, decreases in price and increases in power of processors and the use of parallel processing, theoretical developments that would once not have been feasible for practical systems are now routinely processed in a matter of microseconds or even less.

2.1.5 DISPLAY

The display takes the data from the DAC and generates an image formed from the received data. The importance of the display depends on whether it is required by a human operator to help determine processing. For instance, an operator may look at a display and by looking at the colour, or monochromatic data, decide on the thresholding

values. This is a very error-prone method as the display has its own characteristics which will affect the way data is seen. The actual technology used, such as a CRT or LCD display, also plays a part in affecting the data seen. A table contrasting four display technologies can be found in Foley et al. [1990].

Morovic, as quoted in Sangwine and Horne [1998a], names two techniques, calibration and characterisation, which are required when colours are transferred between different media. The subject of calibration has already been mentioned so characterisation is now discussed. This defines the relationship between a device-dependent colour space and a device-independent colour space where the device-independent colour space is usually the CIE XYZ system. In this way, it is possible to display colours on separate devices and have confidence that they will appear similar. At this point the issue of the colour gamut of the devices used should be explained. Each device has a set of colours it can reproduce which can be represented as a solid in a colour space. The task of the colour gamut mapping is to map colours from the input device to the output device, for instance the display. A uniform colour space such as CIELAB or CIELUV, which are introduced later, is often used but the problem with this is that the hue they represent is not the same as perceived hue and this can cause some hues to shift when displayed. Morovic also explains different techniques to deal with gamut clipping.

It is also important that a range of grey values looks linear on a display so that the brightness is proportional to the grey level. This is achieved with the use of a gamma correction curve. A display also has a number of characteristics which affect colour perception. Although the displayed colour should be identical to the expected tristimulus values this is not always the case if the display is not properly calibrated. The chromaticities of the phosphors could be different to that given in the monitor specification due to age or usage, the colour on the display may vary over the display area for a given applied voltage or the electron beams may interact slightly with each other. In addition, the ambient light external to the monitor, also affects visual perception. However, if the purpose of the image processing is to have complete machine determination of the processing with no human intervention then the issue of the display is not of such concern.

2.2 COLOUR PROCESSING

The subject of colour in processing was introduced in chapter 1 and colour processing and spaces will now be covered in more detail. As has been previously mentioned, much ‘colour’ processing is actually monochrome processing applied multiple times to the separate colour components; when the colour images are processed in the RGB space, this is usually the case. While this may produce a correct result, the disadvantage is excessive processing time. Care should be taken, since the required algorithm may not be suitable for applying separately to the three colour components.

The main uses of colour in image processing, [Castleman op. cit. pp.554-561, Gonzalez and Woods 1992, Pratt 1991], are usually for image enhancement and image analysis. Image enhancement covers processes such as manipulating the hue or saturation, the generation of pseudocolour images and image restoration. Although edges formed by different intensity values are more prominent to the human eye, it is possible to obtain edge enhancement using colour values. Robinson [1976] proposed some definitions of colour edges which could be used with different colour spaces with varying success. Colour image analysis typically involves colour segmentation and the processing of the resultant values using monochrome techniques.

2.2.1 COLOUR SYSTEMS

Since the research presented in later chapters requires the use of colour, the subject of colour spaces is now introduced. There are many other colour spaces which have been devised and this chapter does not attempt to detail all of them. However, some of the more popular and relevant schemes are now discussed.

Colour processing is not usually performed on the XYZ values which were introduced in chapter 1. However, more detail on this colorimetric system can be found in appendix A since although it is not directly used in the research presented here it is wise to have knowledge of this system if colour manipulation is to occur. Further information can also be found in books by Wyszecki and Stiles [op. cit.], Foley et al. [op. cit.], Judd and Wyszecki [op. cit.] and Sangwine and Horne (eds.) [1998b].

Although this system provides a standard for colour specification, the distribution of colours is very non-uniform. In addition, although Y corresponds to luminance, the *X* and *Z* do not correspond to any perceptual attributes. Thus, although it is important to understand the reasons why the CIE standardisation is necessary, it is more usual to find colour processing performed on other colour representations.

Both the L*a*b* and L*u*v* colour spaces were recommended by the CIE in 1976 as perceptually approximate uniform colour spaces and the next few sections deal briefly with them.

2.2.1.1 CIE 1976 (L*A*B*) COLOUR SPACE

Also known as the CIELAB colour space and based on Munsell's uniform colour space, the L*, a* and b* values are calculated using a combination of the X, Y, Z tristimulus values and reference X_n, Y_n and Z_n tristimulus values, where L* is a measure of the colour's brightness.

2.2.1.2 CIE 1976 (L*U*V*) COLOUR SPACE

This is also named the CIELUV colour space and u′ and v′ chromaticity values were introduced where $u' = 4X / (X + 15Y + 3Z)$ and $v' = 9Y / (X + 15Y + 3Z)$ which describe an approximately uniform colour space [Wyszecki and Stiles op. cit. 1982 p.165]. The L* in this space is identical to the L* in the L*a*b* space and represents brightness, u* represents the redness-greenness and v* represents the yellowness-blueness. Again, the u* and v* values are generated using combinations of Y, u′ and v′, together with reference X_n, Y_n and Z_n tristimulus values for a white stimulus. If the values L, u* and v* are known for two colours, the difference in colour between the two can be calculated using a colour difference formula. This space and colour difference formula together with the CIELAB representation are an improvement on, and supersede, the CIE 1964 (U*V*W*) colour space and colour difference formula since the systems are more accurate.

2.2.1.2.1 COMPARISON OF THE L*U*V* AND L*A*B* COLOUR SPACES

In neither colour space are the u* and v*, and the a* and b* values uniquely related to chromaticity since they depend on the value of L*. However, one point to note is that the L*a*b* space cannot be used to convert values to saturation while the L*u*v* is able to do this. Both spaces can convert values to hue and chroma.

2.2.1.3 THE RGB COLOUR SPACE

One of the most widely used colour spaces is the RGB colour model which has already been introduced. The RGB colour space can be thought of as a cube where black is one vertex of the cube and the red, green and blue vectors are orthogonal to each other, as shown in figure 2.1.

In a processing system that represents each colour value as a byte the range of permissible values is 0 .. 255. This means that if intensity is to be calculated the sum of the components should be divided by 3 to remain within the valid range.

A problem which can occur when processing colours is colour-space clipping. This happens when the result of the processing causes one or more components to exceed the valid range of values. With a colour space such as the RGB representation it can be awkward even if only one of the components is outside the range of allowable values. It is not sensible simply to clip the erroneous value as this impacts on the colour properties as the hue, saturation and intensity are all altered. In contrast, if the processing is performed in a space such as the HSI representation, this is not such a problem. The hue never falls out of range due to its continuous nature and the saturation can be clipped although this can change the intensity value. In addition to the processing issue, conversion between colour spaces can also cause values to fall outside valid ranges, but if converting from HSI to RGB space, it is easier to alter the HSI values to attain valid RGB values. There is more discussion of methods to deal with this situation in chapter 4.

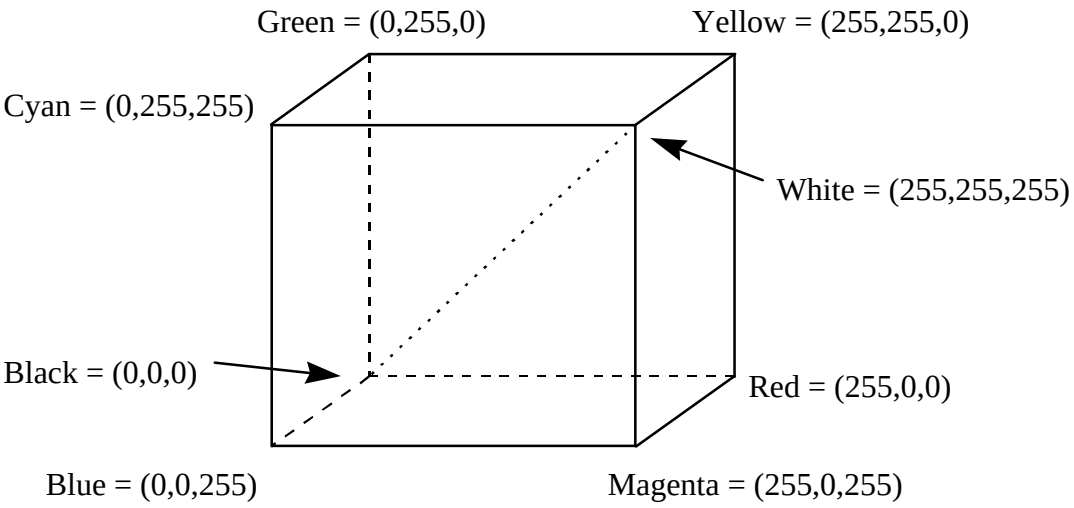


Figure 2.1: RGB Colour Cube

2.2.1.4 OTHER COLOUR REPRESENTATIONS

There are numerous other representations for colour to be found in the literature. However, brief mention will only be made of a couple more as there is the possibility they can be used in the research presented later. The YIQ colour coordinate system is used for the transmission of colour signals under the NTSC television system where Y is the luminance and I and Q describe the hue and saturation. The YUV colour coordinate system is used in PAL and SECAM systems where again, Y represents the luminance and U and V represent colour differences such that U=0.493(B-Y) and V=0.877(R-Y). For the purpose of television, these systems have an advantage over RGB systems since the luminance is represented by a single channel and the two colour signals can be transmitted using a smaller bandwidth.

2.2.1.5 COLOUR PERCEPTIONS

In this section, three methods for representing the way humans perceive colour are introduced. Humans are not identical and so some representations are more applicable to certain observers than others.

Unique (or unitary) hues are hues which cannot be described by the use of other hue names. There are four such hues, red, green, yellow and blue, with red opposite to green and yellow opposite to blue as shown in figure 2.2. A colour space which uses this idea is the CIELUV space.

Alternatively, representations using hue are used. Although observers can perceive hue and lightness, the saturation attribute is different in some people so that if the lightness of a certain colour is changed, some people think the saturation has changed, while others do not agree. The former case is represented by a conical shape, as seen in figure 2.3a while the latter case is represented by a cylindrical shape as shown in figure 2.3b.

The following section is the subject of perceptual colour spaces using hue, saturation and intensity.

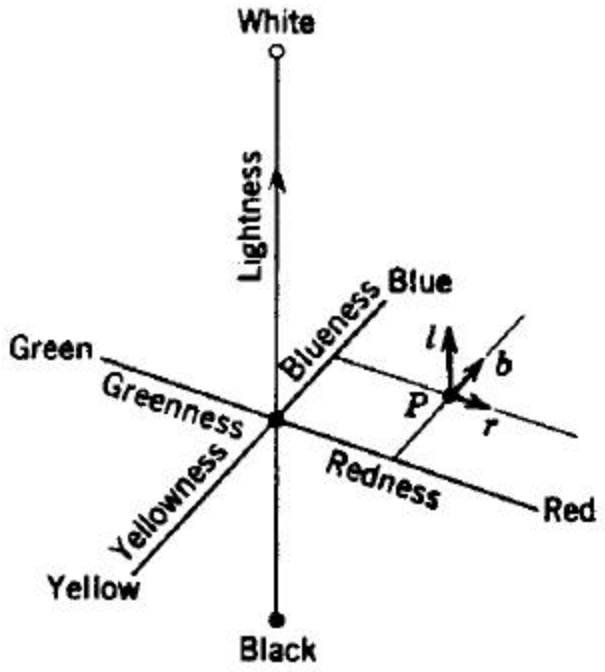
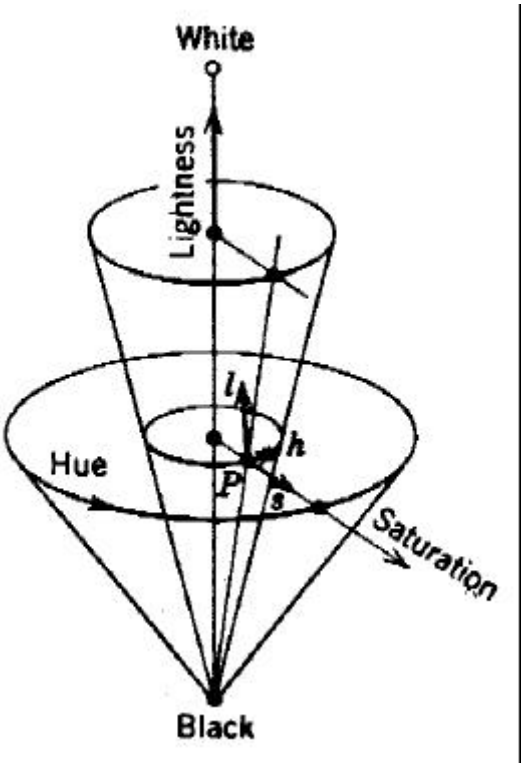
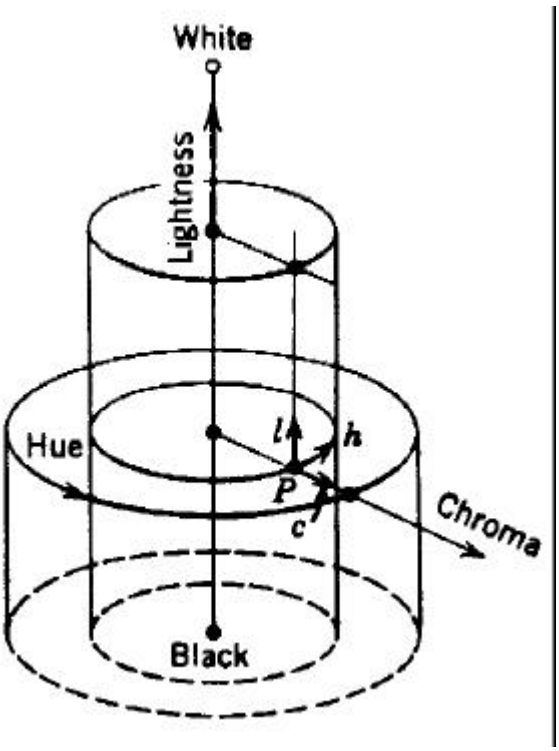


Figure 2.2: Perceptual Colour Space, from Judd and Wyszecki [1975]



(a)



(b)

Figure 2.3: Other Perceptual Colour Spaces, from Judd and Wyszecki [1975]

2.2.1.5.1 Hue, Saturation and Intensity Colour Spaces

It is much easier to describe a colour in terms of its hue, saturation and intensity than by using red, green and blue attributes or L*, u* and v* values. There are a number of colour models [Smith 1978], and names for them, using hue and saturation, or chroma, and with the final attribute being either intensity, value, brightness or lightness.

The hue, saturation, brightness or value (HSB or HSV) hexcone is similar to the space shown in figure 2.3a so that it is represented by a somewhat conical shape that becomes wider as the brightness increases. The hue, lightness, chroma (HLC) colour space is similar to the cylindrical space shown in figure 2.3b and is a variation of the CIELUV colour space. The hue, lightness, saturation, (HLS) colour space can be thought of as two hexcone shapes joined so that the two vertices are at the two ends. Both the HSB and HLS systems can be thought of as slight deformations of projections of the RGB cube looking down the grey line so that for each intensity in the cube, there is a plane perpendicular to the grey line which yields a hexagonal disk, although in the case of the HSB space as the brightness increases, so does the size of the hexagon. The deformations mentioned above result since at low or high intensities, a slice of the RGB space at a constant intensity results in a triangular, not hexagonal shape.

The hue, saturation, intensity HSI space is another similar system but it does not use hexagons to represent intensity planes for all intensities. A projection down the grey line in the RGB cube, shown in figure 2.4, shows the red axis at 0 degrees, with green and blue at 120 degree intervals. In between these hues are the subtractive hues - yellow, cyan and magenta.

To convert from RGB space to HSI space the following equations (2.1a-c) are used:

$$H = \arctan\left(\frac{\sqrt{3}(G-B)}{2R-G-B}\right) \qquad (2.1a)$$

$$S = 1 - \frac{\min(R,G,B)}{I} \qquad (2.1b)$$

$$I = \frac{R+G+B}{3} \qquad (2.1c)$$

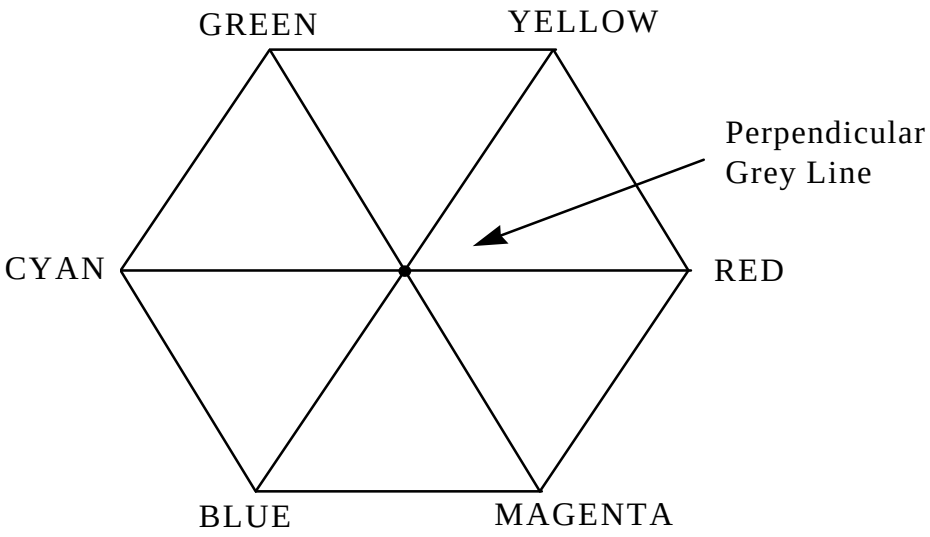


Figure 2.4: Projection down the grey line in the RGB colour cube

The hue equation is easily calculated using the geometry of the hexagon in figure 2.4 above. Unfortunately, saturation is not an independent variable since intensity is used in its calculation. If the intensity is at its lower or higher extremes, the accuracy of the saturation value is adversely affected since the size of the plane representing intensity is much reduced. Intensity is sometimes calculated using a different scaling value (for instance $\sqrt{3}$ due to the geometry of the RGB axes in relation to the plane perpendicular to an intensity of zero) but in this thesis the conversion will be performed using equation (2.1c) unless otherwise stated. Once processing has been accomplished in HSI space the values usually have to be converted back to RGB space for display on a monitor.

Levkowitz and Herman [1993] introduced the idea of a generalised LHS model since many of the LHS family of models have common properties and proposed a transform to and from the RGB and GLHS spaces. Kender [1976] presented a detailed analysis of the problems associated with transforming colour values. He pointed out that there are singularities and other problems which can affect image processing such as segmentation techniques and proposed methods of dealing with it.

2.3 IMAGE REGISTRATION

The accurate determination of the location, scale and rotation of an object in an image in comparison with a known situation is very important in many machine vision systems and is one of the more challenging problems to face researchers. Situations such as occlusion, skewing, distortion and noise can also generate errors in the recognition process. Schalkoff [1989] provides some illustrations of the effects of rotation and scaling on correlation values.

Many different techniques have been applied to obtain accurate registration of images. The basis of the method that has been used in the research presented in this thesis is the use of Fourier transforms and so a general survey of some of the properties and uses of the Fourier transform, which can be used to extract information from images, will now be presented since knowledge of some of the properties is required to understand the processing which is presented in later chapters. Although some of the techniques which are to be described would once not have been feasible for some situations due to their

high computation requirement or price, as computers have become faster and less expensive this has become less of a problem. The survey covers the Fourier transform, its properties and some general theory of its use. There are a number of other transforms related to the Fourier transform such as the Discrete Cosine transform and the Discrete Sine transform; however, they use only real inputs and so were not applicable to the research in this thesis since the colour space used is complex.

Although the research to be presented in later chapters does not require the use of wavelets, the wavelet transform, [Pittner et al. 1993], has emerged over the last few years as a powerful tool not just for pattern recognition but also for data compression. The Fourier transform represents functions using sinusoids by summing and cancelling sinusoids of varying frequency and amplitude. This results in an inefficient representation for features which are localised, such as edges. In contrast, wavelets are waves of finite duration which vary in both position and frequency. Wavelet transforms use these wavelets as their basis functions. Indeed, the Haar transform is the earliest example of a wavelet since its basis functions vary both in scale and position. Wavelet transforms are now used in many applications including image compression, image enhancement, texture analysis and object detection. More information on different transforms can be found in Pratt [op. cit.] and in the published literature.

Having described the properties and uses of the Fourier transform, the Mellin and Fourier-Mellin transforms, which are used here to overcome rotation and scaling variances, are described and compared with the use of Moments, which are a statistical method for describing a shape.

2.3.1 FOURIER TRANSFORMS

The use of Fourier transforms in image processing [Watson 1993] has been applied to different requirements such as pattern recognition, description, restoration, encoding and enhancement. Much of the traditional theory to be found in standard textbooks on the subject explains the Fourier transform using continuous variables but in the case of

digital image processing, discrete variables are used. It is worthwhile now to briefly discuss some theory before using it in later chapters. In addition, the 2D DFT will be dealt with as 2D images are of interest and the 1D case is to be more easily found in the textbooks. It is assumed here that sampling theory is understood, so that once the data has been sampled correctly it is ready for the calculation of the DFT.

The DFT is closely related to the continuous Fourier transform as long as it obeys certain sampling requirements. This means functions must be band-limited and data is then assumed to be periodic.

The discrete relative of the continuous Fourier transform, the 2D DFT is shown in equation (2.2a) and its inverse in equation (2.2b).

$$\mathrm{F}(\mathrm{u},\mathrm{v}) = \frac{1}{\mathrm{N}} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \exp\left\{ \frac{-2pi}{N} (ux + vy) \right\} \qquad (2.2a)$$

$$\mathrm{f}(\mathrm{x},\mathrm{y}) = \frac{1}{\mathrm{N}} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u,v) \exp\left\{ \frac{2pi}{N} (ux + vy) \right\} \qquad (2.2b)$$

It should be noted that although F(u,v) is complex valued, its spatial equivalent does not have to be and, indeed, is usually real for general situations.

An interesting observation is that the inverse DFT can be obtained by first conjugating the input, then calculating the DFT and then conjugating the output such that if $\mathbf{F}(f(x,y)) = F(u,v)$ then $f(x,y) = \mathbf{F}(F^{*}(u,v))^{*} = \mathbf{F}^{-1}(F(u,v))$, where $\mathbf{F}$ denotes the forward DFT.

This result means that an implementation of the DFT can be used to compute both the forward and inverse transform and thus reduce the amount of code required to calculate the forward and inverse transforms. The division by N^2 does not have to be shared between the two equations although it allows easier implementation of the inverse if each equation is divided by N.

The DFT is a useful transform, as will be seen, but it is computationally expensive. This made its computation impractical, even for 1D data, until 1965 when an efficient method was developed, [Cooley and Tukey 1965], for calculating the DFT so that for a 2D

image only $O(N^2 \log_2 N^2)$ additions and multiplications are required instead of $O(N^4)$ under the DFT. This faster method was named the Fast Fourier Transform (FFT) and it works by reducing the size of DFT required by decomposing the transform into smaller DFTs. This means that the original size (NxN) of an image must be able to be divided down to a product of small integers. For instance, if only radix-2 DFTs are to be used then N must be a power of 2 whereas if a mixed radix algorithm is to be used, for instance radix-2 and radix-4, then N must be able to be factorised by a combination of a power of 2 and a power of 4.
The time required to compute FFTs is decreasing all the time. Currently, the time to calculate a 1D 1024 point complex FFT using a single processor can be as little as 66μs.

Having calculated the FFT of an image some use must be made of the result. Since the result is complex valued, the form of the output is dependent upon the input required for further processing. Either the complex values themselves can be used or the phase or magnitude can be calculated. The phase values can be calculated for each pixel by dividing the imaginary value by the real value and calculating the arctan; the more common requirement is to use the power spectrum or amplitude spectrum.

2.3.1.1 POWER SPECTRUM

The Power Spectrum of a Fourier transform is defined as $P(u,v) = F(u,v)F^*(u,v)$ and the Amplitude Spectrum is defined as $\sqrt{(P(u,v))}$. It can be seen that the power spectrum of an image can be obtained by squaring the real part and the imaginary part and adding the two together for each pixel. This is one method for viewing spatial frequency data although it removes phase information.

The following section describes some properties of the Fourier transform since some of those properties are used in the research presented in later chapters.

2.3.1.2 FOURIER TRANSFORM PROPERTIES

The FT has various properties which are useful in analysing images. For a comprehensive account of the properties of the Fourier transform see Brigham [1988]. The property of translation is exploited in later chapters through the use of phase correlation, and knowledge of the rotation and scaling properties is necessary when the processed data undergoes further transforms as described in later chapters.

2.3.1.2.1 TRANSLATION

Due to the Fourier transform property of time shifting, if the real and imaginary parts of a transformed image are each squared and then added together to generate a power spectrum, the resulting data is identical to the power spectrum of the same image having undergone translation. This is because the difference in phase caused by the translation is removed by squaring the real and imaginary values of each pixel. This means that the power spectrum of a transformed image is invariant to translations.

2.3.1.2.2 ROTATION

If an image is rotated by an angle θ, its power spectrum is rotated by the same angle θ. Therefore the Fourier transform of $f(x\cos\theta+y\sin\theta,-x\sin\theta+y\cos\theta)$ is $F(u\cos\theta+v\sin\theta,-u\sin\theta+v\cos\theta)$.

2.3.1.2.3 SCALING

From the scaling property of the Fourier transform, if an image is scaled its frequency content will be inversely scaled and its Fourier transform amplitude also inversely scaled. So, in general terms, a large object in an image will give rise to lower frequencies in the power spectrum in comparison with the same object which has been scaled to appear smaller and which will contain higher frequencies.

2.3.1.2.4 PERIODICITY

The DFT, and therefore the FFT, are defined for periodic sampled functions only. This periodicity can be seen in the corresponding spatial frequency data since $F(u+mN, v+nN) = F(u,v)\exp\left(-2pi(mx+ny)\right)$, and when m, n are integers, the exponential equates to 1.

2.3.1.2.5 CONJUGATE SYMMETRY

For the case where the input image has zero imaginary parts, the Fourier transform produces an output which contains an even real part and an odd imaginary part. The transformed data exhibits conjugate symmetry and almost half its values are redundant. The transformed data therefore has Hermitian symmetry. This is an important result as it means the DFT can be exploited to process twice as much data than would otherwise be possible. Two real images may be processed simultaneously by combining their real data into a complex image so that one real image is the real part in the complex image and the other real image is the imaginary part of the complex image. The DFT is then computed and the DFTs of each real image can be separated. Conversely, in a variation on the DFT, the Hartley transform can be used which reduces the processing requirement for real-only inputs. For real-only input data the Fourier transform result can be recovered from the Hartley transform result.

2.3.1.3 WINDOWING

Although the research presented in this thesis is not specifically concerned with the various types of windows which can be applied to the data before calculating the Fourier transform, it should be understood that windowing is usually desirable. When using the DFT, to obtain correct transformation the data has to be periodic. However, in practice there is usually some discontinuity in the data between periods. This results in the DFT data containing additional unwanted frequency-domain components, known as *leakage*. This problem can be overcome through the use of windows, or weighting functions, which are multiplied with the spatial image data before undergoing the DFT. This does

not need to cause a processing overhead as the window multiplication with the data can be applied during the first butterfly pass of the FFT. There are different types of windows which have different spectral weighting functions and the choice as to which is used is influenced by the opposing values of spectral leakage and resolution.

At this stage, two uses of the Fourier transform are introduced. One is convolution and the other is correlation. As can be seen from the following sections, they are mathematically similar with the difference between them being the conjugation of one of the input data sets.

2.3.2 CONVOLUTION

If an image, f, is convolved with another, g, then the resultant, h, is obtained through equation (2.3).

$$h(j,k) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)g(j-x,k-y) \qquad (2.3)$$

This has the property that the Fourier transform of the convolution (in the spatial domain) of two images is identical to the multiplication of the Fourier transform of each image, equation (2.4a-b).

$$f(x,y) * g(x,y) \Leftrightarrow F(u,v)G(u,v) \qquad (2.4a)$$

$$f(x,y)g(x,y) \Leftrightarrow F(u,v) * G(u,v) \qquad (2.4b)$$

2.3.3 CORRELATION

If an image, f, is correlated with another, g, then the resultant, h, is obtained through equation (2.5).

$$h(j,k) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)g(j+x,k+y) \qquad (2.5)$$

This has the property that the Fourier transform of the correlation (in the spatial domain) of two images is identical to the multiplication of the Fourier transform of one image by the complex conjugate of the Fourier transform of the other image, equation (2.6).

$$f(x,y) \bullet g(x,y) \Leftrightarrow F^{*}(u,v)G(u,v) \qquad (2.6a)$$

$$f^{*}(x,y)g(x,y) \Leftrightarrow F(u,v) \bullet G(u,v) \qquad (2.6b)$$

These properties are well known but it is worth repeating the relationship here as correlation will be used in later chapters. The result of correlating the same image with itself is known as auto correlation and correlating two different images is known as cross correlation.

Convolution can be used to determine the output of a system in response to a given input when the impulse response function is known whereas correlation can be used in pattern recognition to give an indication as to whether an unknown image is similar to a known image. The spatial domain method of correlation is demonstrated in the use of template matching where a template is applied over an unknown image and the position of the template which results in the highest amplitude is most likely to correspond with the position of the matched template in the unknown image. For images above a certain size it is more efficient to perform the correlation in the spatial frequency domain by calculating equation (2.6a), i.e. the multiplication of the Fourier transform of two images where one is conjugated before the multiplication.

2.3.4 MELLIN TRANSFORM

The Mellin transform is defined in equation (2.7a) and can be seen to be a scale-invariant transform if the case of an image, f(x,y), and a scaled copy, f(αx,αy), is considered, equation (2.7b), where M(u,v) and M_2(u,v) are the Mellin transforms of f(x,y) and f(αx,αy) respectively.

$$M(u,v) = \int_0^{\infty}\int_0^{2\pi} f(x,y)x^{-ju-1}y^{-jv-1}dxdy \qquad (2.7a)$$

$$M_2(u,v) = \alpha^{-ju-jv}M(u,v)$$
$$\therefore |M_2(u,v)| = |M(u,v)| \qquad (2.7b)$$

It can be implemented by logarithmically scaling the input image coordinates and then computing a Fourier transform.

The input function to the Mellin transform is expressed using polar coordinates. However, if the r,θ coordinates are log-polar mapped to log(r),θ a Fourier transform can be used to calculate the Mellin transform.

2.3.4.1 FOURIER-MELLIN TRANSFORM

As with the Mellin transform, the Fourier-Mellin transform (FMT), equation (2.8), also uses the Fourier transform but in this case, it requires two Fourier transforms [Casasent and Psaltis 1976, Altmann and Reitböck 1984]. The FMT has the important property of being invariant to translation, scaling and rotation. It has been demonstrated to work using optics or digital data.

$$M(u,v) = \int_0^{\infty} \int_0^{2\pi} f(r,\theta) r^{ju-1} \exp^{-jv\theta} \, dr d\theta \qquad (2.8)$$

By combining the Mellin transform with a Fourier transform a rotation, scale and translation invariant algorithm can be obtained. Firstly, a Fourier transform spectrum is calculated which results in the removal of translations while keeping scale and rotation variances. Then the coordinates are log-polar mapped to make scalings and rotations appear as translations in the image and a final Fourier transform removes these translations. The power spectrum of the result yields identical data to that of the image containing the same object which has undergone rotation, scaling and translation. Therefore, a captured image can be processed as described and correlated with a known reference image.

A relationship between the FMT spectrum and normalised moments was identified by Li [1992]. However, Grace and Spann [1991] compared the effect of noise on Fourier-Mellin descriptors and on moment based features and found that Fourier-Mellin descriptors performed better.

Having introduced the colour spaces and transforms of interest, the next chapter describes the literature on object recognition using these transforms and also on *colour* object recognition.

3. COLOUR OBJECT RECOGNITION

Having discussed in the last chapter the various colour spaces with particular reference to the RGB space and colours defined in terms of human perception (sometimes called phenomenal spaces) this chapter now investigates firstly, the use which has been made of the Fourier transform with regard to object recognition and secondly, the use of colour in the object recognition process with regard to the use of Fourier transforms. However, it should become apparent from reading the section on colour in this chapter that the use of colour in object recognition has not been fully exploited since a lot of processing involves the use of monochromatic techniques repeated for each colour channel. The range of literature on *colour* object recognition is limited and much of it concerns the use of monochromatic techniques, but the relevant literature on colour object recognition as well as the transforms is now discussed in greater detail.

There are four main sections to this chapter. Although there are many different methods which can be used to obtain object recognition only those applicable to the research presented in later chapters are discussed here. Woods [1996], Ashbrook [1993], Pratt [op. cit.] and Gonzalez and Woods [op. cit.] provide reviews of pattern recognition techniques which cover the use of Fourier and Fourier-Mellin transforms, moments, neural networks, segmentation (which is usually performed in the spatial domain before further processing but it can also be used in the spatial frequency domain to obtain a set of features), Fourier descriptors, scale-space, graph matching and Hough transforms. However, as previously explained, the processing in the research presented in later chapters is based on the Fourier transform so the following three sections covering different algorithms for object recognition all require the Fourier transform to be calculated. The first section of this chapter, phase correlation, discusses the use of phase information in a Fourier transformed image and the implementation, applications and potential problems of phase correlation. The second section covers the use of Cepstral analysis which is more commonly used to analyse 1D data and the third section then describes the Fourier-Mellin transform and the literature on object recognition using it. Although colour has started to be used in recognition schemes, it is very rarely used

where Fourier transforms are part of the recognition algorithm. More usually, colour is used in segmentation techniques during the process of recognition. The final section of this chapter discusses the role of colour in object recognition systems.

3.1 FOURIER TRANSFORMS AND OBJECT RECOGNITION

The main use of Fourier transforms in object recognition is through correlation. The general method of correlation is cross-correlation but there are variations on this method which can provide better results. As stated in the previous chapter, cross-correlation is calculated simply by Fourier transforming two images *a* and *b* to obtain *A* and *B*, complex conjugating one to obtain *B**, and multiplying *A* by *B** before inverse Fourier transforming the result. However, improvements to the sharpness of the resultant peak can be made by altering the frequency content of *A* and *B* so that some frequencies have more effect on correlation than others. Taking this alteration to its limit by setting the frequency magnitudes to unity without altering the phase results in correlation due only to the phase information in the two Fourier transformed images and this is investigated in the next section.

Fourier transforms have also been used not just in correlation but also in Cepstral analysis which is most often used for multipath or echo removal in received signals. For the research presented in later chapters, it was thought that these echoes could be used to determine the position of an object in an image so for this reason, cepstral techniques are explored in section 3.1.2 and investigated in a later chapter. Leaving correlation and moving on to geometrical variances, Fourier transforms have been used in conjunction with other transforms to remove not only translation but also rotational and scaling differences between images and this is investigated in section 3.1.3.

3.1.1 PHASE CORRELATION

One method of determining the position of an object in an image is through the use of cross-correlation. This can be thought of as template matching where the template is placed over the image to be searched and shifted around for all positions until a maximum is found. This is a very time consuming process particularly as the image size

increases and since the process of cross-correlation has a counterpart in the spatial frequency domain it is usually faster to employ an FFT.

The phases contained in signals play an important part in their descriptions. This is true for a number of situations, including sound and holography. In the case of images this importance was demonstrated by Oppenheim and Lim [1981], where it was shown that if two different images *a* and *b* are transformed into the frequency domain to become *A* and *B* respectively, and the magnitude of A is combined with the phase of B, after calculating the inverse Fourier transform the resultant image *c* resembles that of the image whose phase information was used (*b*)[3.1].

$$F(a(x)) = A(w) = |A(w)|e^{jqw_a}$$

$$F(b(x)) = B(w) = |B(w)|e^{jqw_b}$$

$$F(c(x)) = C(w) = |A(w)|e^{jqw_b}$$

where *F* indicates the Fourier transform of an image which can be multi-dimensional.

This fact has been used in a number of image processing problems such as image coding and blind deconvolution. Under certain conditions, it is also possible to reconstruct the magnitude to within a scale factor from the phase data [Hayes et al. 1980].

It should therefore be apparent that the use of phase data in an image can be used to provide information to enable the correlation of images and ultimately image recognition. From the shift theorem for Fourier transforms (stated in the previous chapter), it can be seen that if data is shifted in the spatial domain, it corresponds to a phase change in the spatial frequency domain. For instance, if $g_1(k,l)$ and $g_2(k,l)$ are two NxN images where $k,l \in (0,N-1)$, and $G_2(m,n)=\mathbf{F}(g_2(k,l))$ where **F** represents the Fourier transform, then if g_2 is a replica of g_1 cyclically shifted by (k_0,l_0) then

$$g_2(k,l)=g_1((k-k_0)_{modN},(l-l_0)_{modN})$$

$$G_2(m,n)=\exp\{-j[2\pi(k_0m+l_0n)/N]\}G_1(m,n)$$

[3.1] This experiment to prove the importance of phase was recently queried by Tadmor and Tolhurst [1993] when they suggested that the amplitude spectra of most natural scenes were very similar and that both the phase *and* amplitude spectrum may be required to specify certain images.

Kuglin and Hines [1975] first proposed a new correlation method which was called *phase correlation*, [Pearson et al. 1977]. It is based on the simple idea of calculating the inverse transform of the phase difference of the FT between two images. The phase difference contains information relating to the displacement between the two images. Once this phase difference is inverse transformed a sharp peak is generated relating to the spatial displacement of one image relative to the other.

The phase correlation method, now explained in more detail, is calculated as follows; a reference image (g_1) is compared with another which is called the object image (g_2) by multiplying the FFT of one (G_1) by the complex conjugate of the FFT of the other (G_2^*). This obtains the cross-power spectrum which is then normalised and from this the phase correlation surface (P) is calculated by taking the inverse Fourier transform ($\mathbf{F}^{-1}$) of the spectrum, equation (3.1). Assuming that an object is contained in both the reference and object image, the result is an intensity peak in the phase correlation surface (P) whose position can be used to determine the displacement of the object between the reference image and the object image.

$$P = F^{-1}\left(\frac{G_1 G_2^*}{\left|G_1 G_2^*\right|}\right) \qquad (3.1)$$

The distance displaced can be calculated as shown in equation (3.2) and holds for both the x and y directions:

$$\text{Displacement of object}_{x,y} = (\text{Reference}_{x,y} - \text{Object}_{x,y}) \bmod \text{Image Size} \qquad (3.2)$$

This is illustrated in figure 3.1 where two simple images contain the same object but the position of the object is displaced in the second image in comparison with the first. When the phase correlation surface is calculated using the two images, the result is a peak whose position corresponds to the displacement of the object between the two images.

The cross correlation of two images in the spatial domain is related to phase correlation and can be equated as follows:

$$\begin{aligned} \text{cross correlation} &= g_1 \otimes g_2 = \mathbf{F}^{-1}(G_1 G^*_2) \\ &= \mathbf{F}^{-1}\{(G_1 G^*_2)/|G_1 G^*_2|\} \otimes \mathbf{F}^{-1}\{|G_1 G^*_2|\} \\ \therefore \text{cross correlation} &= P \otimes \mathbf{F}^{-1}\{|G_1 G^*_2|\} \end{aligned}$$

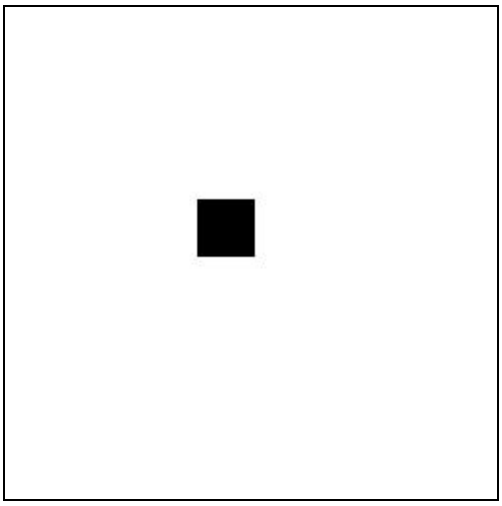


Figure 3.1a: First input image

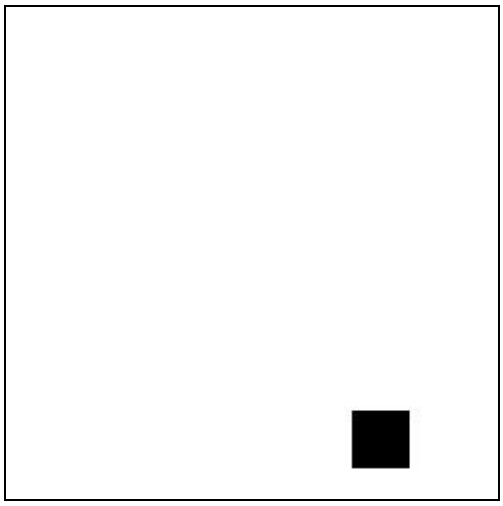


Figure 3.1b: Second input image

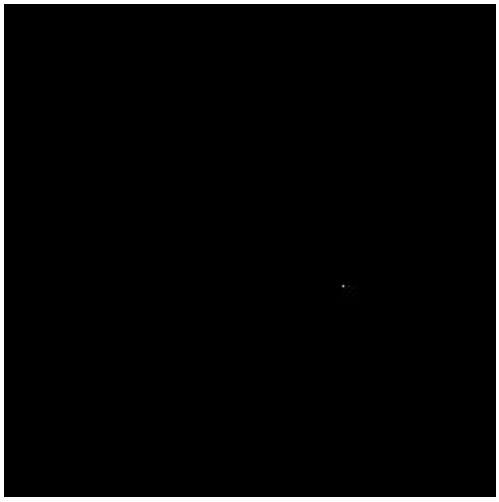


Figure 3.1c: Phase correlation surface

The advantage of the phase correlation method over some other correlation methods is that it is relatively scene-independent and is insensitive to narrow bandwidth noise and convolutional image degradations. Importantly, it is invariant to changes in the intensity of the image caused by scaling or a level shift in intensity so is suitable in environments where the light level is not easily controlled. Although cross correlation performs well in signals which are degraded by white noise, many practical signals are not affected by this type of error. Phase correlation produces a sharper correlation peak than cross-correlation [Horner and Gianino 1984] and is more suited to image degradations which cannot be approximated by white noise. In addition, it performs better in white noise and is superior to cross-correlation in detecting displacements in most situations. Scene independence can be achieved by the "whitening" of the cross-power spectrum.

Pearson et al. [op. cit.] investigated phase correlation more thoroughly by showing that the alignment error and probability of false match can be calculated. This follows from the approximation of a Gaussian distribution to the amplitudes of the incoherent peaks, where the incoherent peaks contain all the power in the image except for a single coherent peak. They also discuss the application of a weighting function in the spatial frequency domain which would act as a filter so that the phase correlation algorithm would be optimal for a range of signal types; from a wide-bandwidth signal degraded by narrow-bandwidth noise, to a narrow-bandwidth signal degraded by wide-bandwidth noise. This is achieved by varying the amount of normalisation of the cross-power spectrum. For instance, if the cross-power spectrum is normalised (phase correlation), there is no weighting function, and the algorithm is optimal for the first signal type mentioned above. However, at the opposite extreme, if the cross-power spectrum is multiplied by a certain weighting function so that the cross-power spectrum is not normalised (cyclic cross correlation), the algorithm is optimal for the second signal type mentioned above.

One cause of possible problems is the situation of geometrical distortion which adds noise to the correlation result. This can be overcome, to some extent, by the use of a low-pass filter which removes high spatial frequencies by setting the unwanted higher spatial frequencies to zero.

3.1.1.1 IMPLEMENTATION

Mindful of the heavy computational requirements for calculating the phase, Pearson et al. [op. cit.] also developed a method for computing the normalised phase which required less computation than that required for computing the cross-power spectrum and dividing by its modulus. Even allowing for modern advances, this is of interest in systems where real-time processing is required under heavy computation requirements. This was achieved by generating a 3-bit index for each pixel position in each image and, for each pixel position, subtracting one index from the other. The resulting value was then used to access a pre-stored phase value. Basically, the three bits were used to divide four 90 degree quadrants (representing the possible phase range of 0 to 360 degrees) into eight 45 degree octants by using a single bit to show if $\Re \geq 0$, $\Im \geq 0$ and $|\Im| \geq |\Re|$. This technique has the disadvantage that the amplitude of the peak is reduced and the displacement error is increased, the amount depending on the size of the image. However, it does mean that phases can be stored using only 1.5 bits/pixel due to the Hermitian property of real images.

Another technique, which was used to simplify the hardware implementation of phase correlation but did not change the amount of computation required for its calculation, was used by Morandi et al. [1986]. Instead of calculating a 2-D Fourier transform, each image was converted to a 1-D form by writing the contents of each consecutive row of a 2-D array to a 1-D array. If phase correlation is then calculated using 1-D Fourier transforms, the displacement of the image is determined. The advantage is that the data is processed in the same order as the data is received from the video input, and that the separate Fourier transforms of the rows and columns is avoided.

Having considered the technique of phase correlation, its applications will now be discussed.

3.1.1.2 PHASE CORRELATION APPLICATIONS

Phase correlation is a powerful tool that can be used on its own to determine translations or in conjunction with other algorithms, for instance to help in the registration of rotated and scaled images or in stereo disparity analysis.

Papadimitriou and Dennis [1994] investigated the use of phase correlation in determining the disparity of stereo images. They generated two images containing identical backgrounds and also an area of interest which was placed at a slightly different location in the second image in comparison with the first image. This area of interest was then rotated slightly which, together with the slight translation, might be expected in stereo images. They used the technique of dividing each image into smaller (overlapping) blocks which were then windowed. The size of the blocks was chosen using the assumption that horizontal disparity was more usual than vertical disparity. Each block then underwent phase correlation and the data was normalised to the second highest peak to suppress the effect of the often dominant (0,0) peak. The positions of the peaks were then used to produce disparity vectors. Although phase correlation was, in this paper, able to generate correct results from slightly rotated images, the amplitude peaks resulting from phase correlating two rotated images were reduced. This is a problem if an object recognition system is to cope successfully with the rotation of objects. The use of phase correlation to determine scale and rotation differences will therefore be discussed in a later section in this chapter.

At this point it is interesting to survey other techniques which could be used in detecting the displacements of objects so that the amount by which an object in one image is displaced in comparison with the same object in another image can be quantified.

3.1.2 Cepstral Analysis

As with the process of phase correlation, cepstral analysis (usually pronounced “kepstral”), [Lee et al. 1988, Childers et al. 1977], is useful for indicating whether an object is located in an image with reference to an initial image. Two techniques will be discussed; the power cepstrum and the complex cepstrum method. The power cepstrum can be used to determine signal arrival times and the complex cepstrum, although also able to determine echo arrival time, is better used to determine the waveform of the signal. Both methods are illustrated [Kemerait and Childers 1972] in figure 3.2. The processes have been applied to many fields including speech, seismology, sonar, radar and hydroacoustics.

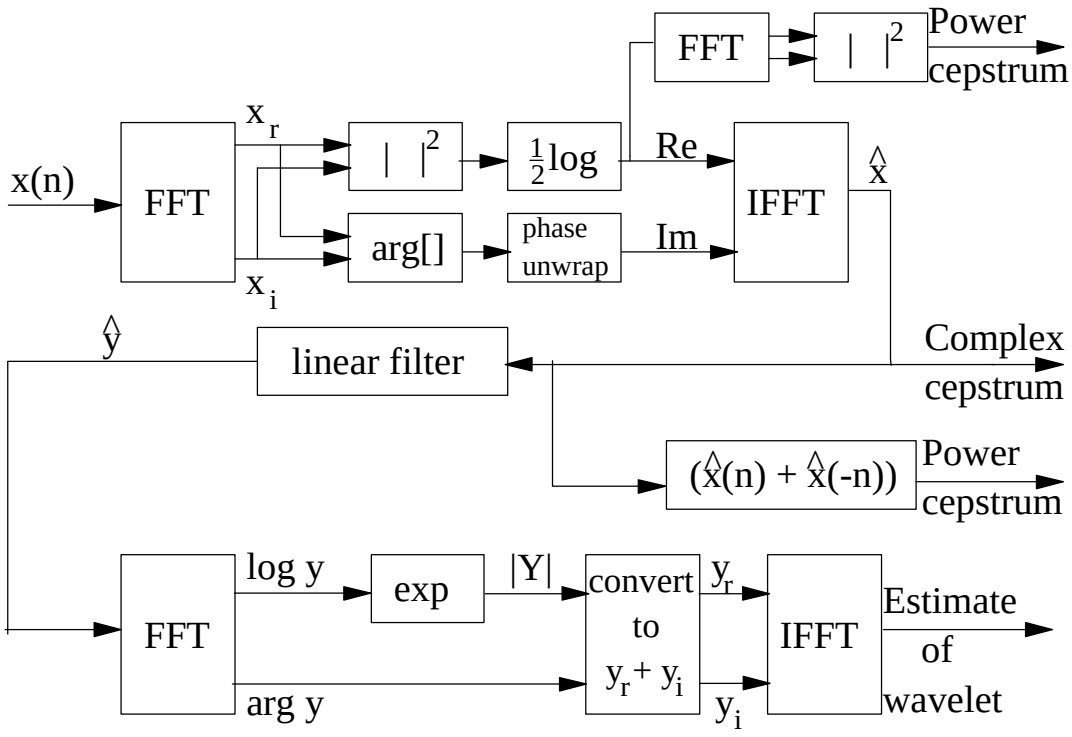


Figure 3.2: Cepstrum calculation block diagram, after Kemerait and Childers [1972]

The reason for the unusual sounding names for processing is due to the result of processing being in the spatial domain since two FTs are used. To reinforce this point, terms which would be used in the spatial-frequency domain have a few letters back-to-front. For instance a cepstrum is, in essence, a spectrum of a spectrum and quefrency refers to frequency. Many more terms were coined by Bogert et al. [1963] but not all are in common usage.

3.1.2.1 POWER CEPSTRUM

The power cepstrum which was first defined by Bogert et al. [ibid.] can be defined as the power spectrum of the logarithm of the power spectrum of a function. This is shown mathematically in equation (3.3):

$$C_p(g(x,y)) = | \mathbf{F}(\ln | G(u,v) |^2) |^2 + \text{echoes} \quad (3.3)$$

where g(x,y) is the superposition of the reference image with another image containing the displaced object and G(u,v) is its FFT. If the object is moved to a new location in the image, a series of 'echoes' or weaker pulses relating to the displacement of the object, and which decrease in amplitude, will appear. In the 2D case, the positions of the echoes will indicate the 2D translation of the object in comparison with the reference image.

The essence of the power cepstra calculation is that an echo in an image, (caused by the translation of the object in comparison with the initial image containing the object), will cause a ripple to appear in the log power spectrum. The power spectrum of this log power spectrum (the power cepstrum), generates a peak corresponding to the frequency of the ripple. Since the cepstrum is in the spatial domain, the peak corresponds to the distance moved by the object in the image. The modulus of the object image should be much smaller than unity so that the echoes rapidly decrease in amplitude and do not obscure the main peak. Mathematically, it can be explained as follows; consider the case of a 1D signal, x(t), containing a signal, s(t), with a single echo:

$$x(t) = s(t) + a_0 s(t-t_0)$$

The power spectrum, $P_x(\omega)$, of x(t) is:

$$P_x(\omega) = P_s(\omega)[1 + a_0^2 + 2a_0 \cos\omega t_0]$$

where $P_s(\omega)$ is the power spectrum of signal s(t).

Then take the logarithm of both sides:

$$\log [P_x(\omega)] = \log [P_s(\omega)] + \log [1 + a_0^2 + 2a_0 \cos\omega t_0]$$

If the amplitude of the echo (a_0) is much less than 1, the expansion of the second logarithm term can be approximated by $2a_0 \cos\omega t_0$ and so a ripple appears in the log power spectrum. However, if the amplitude of the echo is close to 1 then the expansion of the second logarithm term has to include higher order terms and therefore different ripples are added. Peaks corresponding to these echoes are generated from this data by the calculation of the power spectrum. This final result yields the power cepstrum.

The use of the power cepstrum to determine translations in images was used by Lee et al. [op. cit. 1988] to correct shifted and rotated medical images. Before using the power cepstrum, they corrected for *rotational* differences by repeatedly using the shift invariant property of the Fourier transform. This was done by calculating the difference between the reference power spectrum and the object power spectrum. The object power spectrum was then rotated slightly and the comparison repeated. This was repeated until the difference between the two power spectra was at a minimum and then it was assumed that the rotation difference between the two images had been removed. This is certainly a method for removing rotations but is a tedious and computationally expensive way to overcome the problems of rotation. Lee et al. [1989] did, however, make the claim that the power cepstrum method for detecting shifts was a better algorithm to use than phase correlation in cases where there was noise or in images where the dominant feature changed over time. However, no comment was made as to which method was superior if the images were filtered as discussed by Pearson et al. [op. cit.].

3.1.2.2 COMPLEX CEPSTRUM

Having discovered the power cepstrum in the literature, the complex cepstrum was soon of interest as its computation outputs complex values. This interest was due to the way colour information was to be represented, (as described in chapter 4), and ideally retained during processing.

The complex cepstrum is closely related to the power cepstrum, as can be seen from figure 3.2. The calculation of the complex cepstrum has similarities with that of the

power cepstrum. It can be defined, as shown in equation (3.4), as the inverse transform of the complex logarithm of the transform of a function.

$$C_C(g(x,y))=\mathbf{F}^{-1}(\ln(G(u,v))) + \text{echoes} \qquad (3.4)$$

In the resulting two-dimensional array, a peak is obtained which indicates the displacement of the object but in this case a complex number represents the peak. Unlike the power cepstrum, the complex cepstrum retains the phase information and can therefore be used for recovery of the signal.

As before, its calculation can be explained mathematically: again, consider the case of a signal, x(t), containing a single echo:

$$x(t) = s(t) + a_0 s(t-t_0)$$

The Fourier transform of x(t) is:

$$X(\omega) = S(\omega)(1 + a_0 e^{-j\omega t0})$$

If the complex logarithm is taken of both sides:

$$\log[X(\omega)] = \log[S(\omega)] + \log(1 + a_0 e^{-j\omega t0})$$

If $a_0 < 1$ then after the above is inverse Fourier transformed to obtain the complex cepstrum:

$$F^{-1}\{\log[X(\omega)]\} = F^{-1}\{\log[S(\omega)]\} + a_0\delta(t-t_0) -(a_0^2/2)\delta(t-2t_0)...$$

If $a_0 > 1$ then in order to expand the second log term using a convergent series, the log equation is rearranged to give:

$$\log[X(\omega)] = \log[S(\omega)] + \log a_0 e^{-j\omega t0}(1 + 1/a_0 e^{-j\omega t0})$$

$$\log[X(\omega)] = \log[S(\omega)a_0 e^{-j\omega t0}] + \log(1 + 1/a_0 e^{-j\omega t0})$$

After the above is inverse Fourier transformed to obtain the complex cepstrum:

$$F^{-1}\{\log[X(\omega)]\} = F^{-1}\{\log[S(\omega)a_0 e^{-j\omega t0}]\} + (1/a_0)\delta(t+t_0) -(1/2)(1/a_0)^2\delta(t+2t_0)...$$

Skinner and Childers [1976] examined the implementation of complex cepstrum processing and explained where bottlenecks are likely to occur and the time taken for each part of the process.

The complex cepstrum requires a continuous phase curve for the input data and this curve is known as the unwrapped phase. Before unwrapping, the phases are contained as modulo 2π and thus this causes discontinuities. To obtain a real output of the complex

cepstrum (inverse Fourier transform of the log magnitude and phase) for a real input requires the phase to be continuous and odd and the log magnitude to be even. A number of methods have been proposed to obtain unwrapped phase [Childers et al. op. cit., Dudgeon 1977, Tribolet 1975].

Although the phase is retained (which is an important asset as will be shown in the next chapter) using the complex cepstrum, it is more difficult to determine the echo arrival times than by using the power cepstrum [Kemerait and Childers op. cit.]. It was decided, therefore, to concentrate on using the phase correlation technique to determine object displacements.

The ability of the power cepstrum to determine echo arrival times in comparison to the complex cepstrum method can be due to the linear phase contribution of the imaginary part of the logarithm masking the echo delay in the complex cepstrum. However, more detailed discussion of this topic will not be followed further here as, although it is interesting, it is not so directly pertinent to this research. Relevant details are contained in the referenced literature.

3.1.3 FOURIER-MELLIN TRANSFORM

This section investigates the use of the Fourier-Mellin transform as an aid to obtaining invariance to rotation, scale and translation. The constituent parts of the practical implementation of the transform will firstly be discussed. Since the FT has been dealt with in previous sections, the main emphasis here is to describe the log-polar transform. Its use together with the Fourier transform enables the Mellin transform to be calculated. Care must be taken before using the Mellin transform as it assumes the object of interest is at the centre of the image. One approach to finding the centre of the object of interest in order to remove the rotation and scaling is to use moments. This is acceptable if it is possible to separate the object from the clutter in the image in order to calculate its centre of gravity. However, a more general purpose method is to use a Fourier transform to centre the object (and all other data) in an image. This is because the Fourier transform is translation invariant and so no matter where in an image an object occurs, it

has the same power spectrum signature. This method, where a Fourier transform is calculated, then a log-polar coordinate mapping and Fourier transform (or Mellin transform), is called a Fourier-Mellin transform.

3.1.3.1 LOG-POLAR TRANSFORM

As has been mentioned previously in chapter 2, the modulus (or spectrum) of a Fourier transform of an image containing an object will be similar no matter where in the image the object is located. For the case where the background is different and the object is displaced, although there may be different spatial frequency data due to the background, the spectrum of the spatial frequency content due to the object is the same; the phase will be different. The Fourier transform can therefore be thought of as a translation invariant algorithm but it will not overcome the problems associated with the rotation and scaling of an object in an image. These are problems because the rotation and scaling cannot be quantified or removed just by calculating the FT and are caused because the FT is not rotation or scale invariant. If an object is rotated in an image, its FT is a rotated version of the FT of the unrotated image. If the object is scaled, the spatial frequencies due to the object are inversely scaled. This can be overcome by rotating one image with another by small amounts and then comparing the two until an optimal match is found [De Castro and Morandi 1987] but this is not a good solution as it has the disadvantages of being wasteful of time with unnecessary computations being made. However, in this section a technique will be discussed to eliminate the problems caused by variations in scale and rotation. This is the use of the log-polar coordinate transformation, [Wilson and Hodgson 1992, Weiman and Chaikin 1979, Massone et al. 1985, Reitböck and Altmann 1984].

This transform is a way of non-linearly sampling pixels so that those near the centre of an image are sampled more than those further from the centre. This non-linear sampling has an analogy with the human visual system (HVS) in that the sampling points are similar to the distribution of the foveal and extrafoveal areas in the retina. When light enters the eye it is focused on the retina and absorbed by photoreceptors. The receptive field radius increases exponentially with the viewing angle, or radius, from the centre of the fovea.

Each ring of receptors contains the same number of receptive fields. The analogy for the mapping is that the points are mapped from the retinal plane to the cortical plane. The visual cortex which receives the signals from the receptive fields has a uniform structure. This mapping in the HVS can therefore be modelled using the Log-Polar transform which maps Cartesian coordinates to Polar coordinates and then logarithmically scales the radial axis.

In essence, this is a method of sampling an image g(x,y) to produce a new image h(r,θ) in such a way that if an object in g is rotated this causes the components due to the object in the image to be shifted vertically in the transformed image in comparison with a reference image. In a similar manner, if an object in g is scaled so that it is enlarged or reduced, this causes the log-polar transformed image h, to be shifted horizontally in comparison with a reference image. The amount of linear shift on either axis is indicative of the amount of scaling or rotation undergone by the object of interest. A constraint on this procedure is that the object of interest must be at the centre of the sampling or the sampled data will be distorted in comparison with what would be expected. If the object is not at the centre of sampling, it will cause the transformed image to produce a grossly distorted LPT image in comparison with that expected for the correctly centred sampled image. This is dealt with mathematically below.

A pixel position can be described using polar coordinates, given a known origin; so that r represents the distance of the pixel from the origin and θ represents the angle of the pixel position from a preset zero degrees reference. If the r and θ relate to the pixel positions in the transformed image, h, then the sampled pixels in the input image g have positions related to log(r) and θ. For the purpose of standard results in this thesis, the horizontal axis of LPT images was chosen to represent r (scaling) and the vertical axis to represent θ (rotation). Therefore, if an object is scaled by *a* then the sample points describing the same features are (log(ar), θ) = (log(r) + log(a), θ); the effect of rotation is self-explanatory.

To be more specific, if the origin of the coordinates is taken to be at the centre of the image, with x in the horizontal direction and y in the vertical direction, the radius out from the centre of the image is sampled at logarithmically increasing distances, whilst the angle linearly increases. Despite appearances, *both* the r and θ are logarithmically scaled.

The sampling points can be represented by $\log(re^{j\theta})$, and this results in $\log(r) + j\theta$ so that although θ is sampled linearly it has, in fact, together with r, undergone a log-polar transform.

In relation to a square image of size N*N, the following equations (3.5a-b) can be used to determine the sampling positions.

If $r(u) = (N/2)^{**}(u/(N-1))$ where u = 0 .. N-1.

$\phi(v) = 2\pi v/N$ where v = 0 .. N-1.

then $x(u,v) = N/2 + r(u)\cos(\phi(v))$ (3.5a)

$y(u,v) = N/2 + r(u)\sin(\phi(v))$ (3.5b)

where r(u) : radius of the sampling ring

ø(v) : angle of the sample on the ring with respect to the horizontal axis

x(u,v) : horizontal sample position in input image

y(u,v) : vertical sample position in input image

u : horizontal coordinate in new image

v : vertical coordinate in new image

For each exponentially increasing radius from the centre of the image, the u coordinates of the transformed image increase by one, and for each ring, N samples are taken at linearly increasing angles around the ring and this increases the v coordinate one by one. If the sampling resolution is required to be altered so that the output image is not the same size as the input image, the value of N (relating to the required output size) can be changed in the equations calculating r(u) and ø(v); this, in effect, changes the base of the logarithm. The resolution of the sampling was discussed by Juday [1995].

The sampling positions can be seen in figure 3.3. It should be noted that, due to the logarithmic nature of the sampling, the sampling points become sparser for a given radius, or for a given θ, as the radius increases.

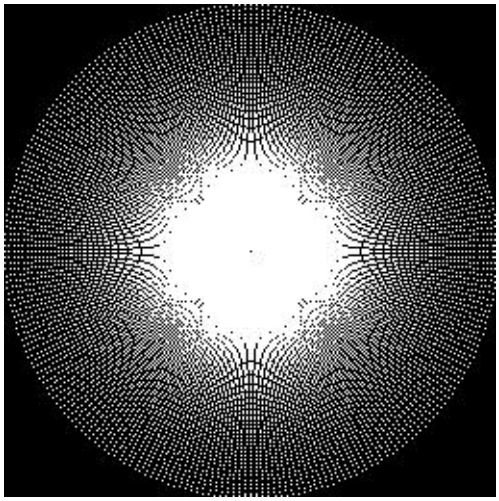


Figure 3.3: Log-polar sampling points

The images illustrated in figure 3.4 illustrate examples of log-polar sampling input images. Figure 3.4a shows an image containing two squares of different sizes and figure 3.4b shows the result of log-polar sampling this image. It can be seen that scaling an object results in a horizontal shift of the components relating to that object in the transformed image. If one of the squares, in this case the smaller one, is rotated by 30°, see figure 3.4c, then the log-polar sampled result is as shown in figure 3.4d. It can be seen that rotation of an object causes the components related to that object to be shifted vertically in the transformed image.

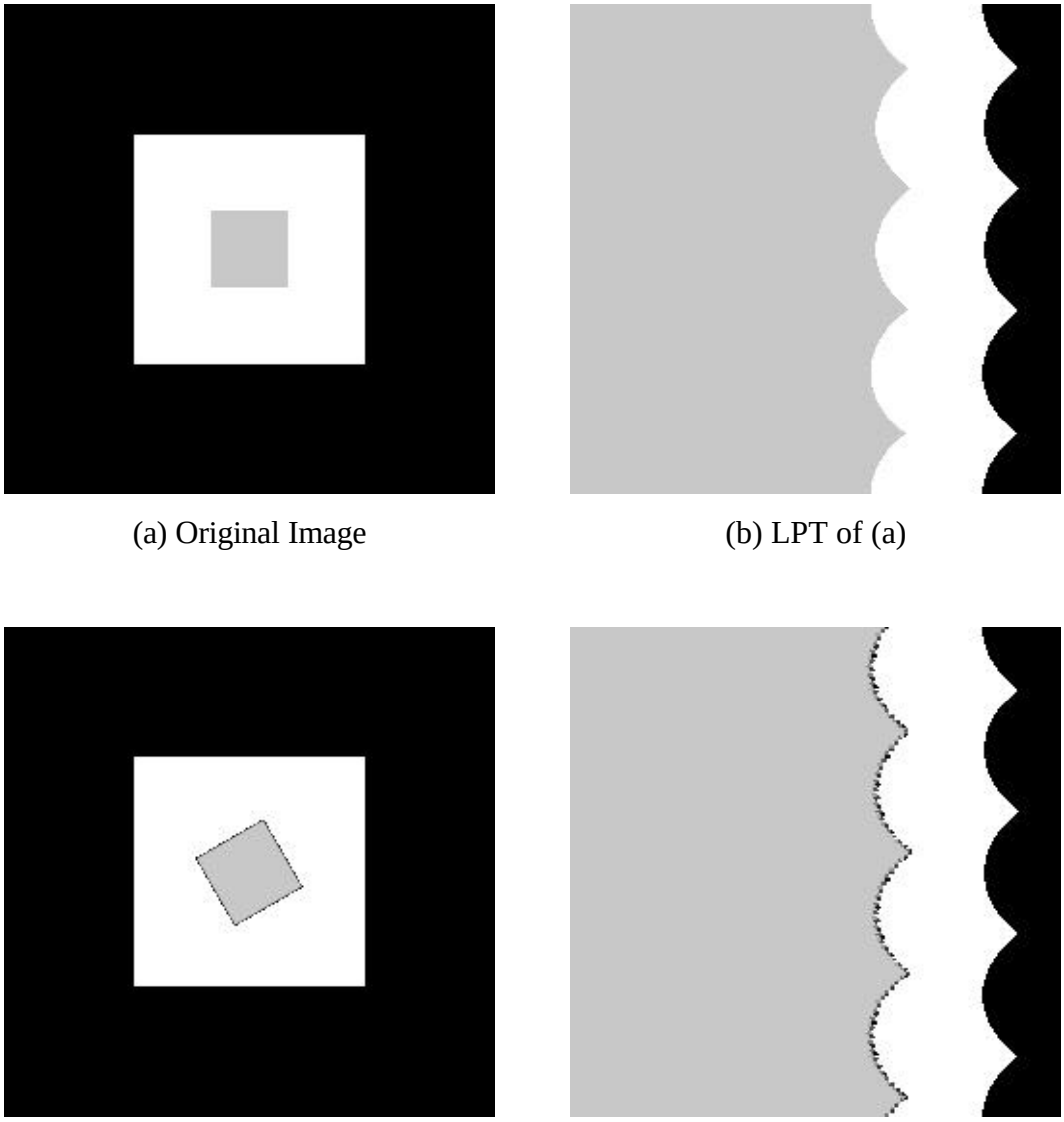
Having described the log-polar sampling of an image, its use in combination with the Fourier transform will now be discussed.

3.1.3.2 MELLIN TRANSFORM

As stated in chapter 2, the Mellin transform [Robbins and Huang 1972] is a scale invariant transform which can be implemented by logarithmically scaling the input image coordinates and calculating a Fourier transform. The result of this is that if two images, containing objects of different scale, are Mellin transformed the result of their power spectra are identical (within an intensity scale factor). This assumes the centre of the object of interest is at the centre of the sampling.

3.1.3.3 FOURIER-MELLIN TRANSFORM

This transform is so useful because it is easily implemented and it is able to remove rotation, scale and translation variances without iterative processes; it has been used many times for this purpose - removal of these variances. However, depending on the situation, it is sometimes more useful to extract information to be able to quantify these variances. The paper by Chen et al. [1994] introduced the idea of using a Fourier-Mellin invariant (FMI) descriptor for an image together with phase correlation to determine the amount of rotation and scaling. The FMI is invariant to translation but represents rotation and scaling as translations. In terms of the practical implementation of the FMT, the FMI is obtained at the output of the LPT, after the first FT.



(a) Original Image (b) LPT of (a)

(c) Original Image (d) LPT of (b)

Figure 3.4: Log Polar coordinate transformation

Phase correlation is then used at that stage to compare the FMI descriptor with a reference FMI descriptor. A variation on this process is used in the research which is presented in the following chapters so further discussion on the use of the FMT and phase correlation can be found there.

A scheme to correlate images for scale and translation invariant pattern recognition was proposed by Altmann and Reitbock [op. cit.]. They discussed the problems due to sampling which mean that at high spatial frequencies, the distance between sample points is too large to be able to resolve specific frequencies. Border effects were also discussed since the DFT assumes data is periodic. If an object is rotated or scaled and a FT and then an LPT are calculated, if the object is rotated such that it shifts out of the upper border, it will appear at the lower border. This conforms with the cyclic property of the DFT. However, for scaling there is a problem as it is not cyclic. Any data that shifts off the right border does not then appear at the left border. To counter this effect, some criteria were given as to the size allowed of the object of interest in an image. This is certainly one method of addressing the problem but it does restrict the object size and it may be that it is not possible in the real world to constrain them so easily without increasing artificially the image size. The problem of sampling is addressed in a later chapter.

3.1.3.3.1 COMPUTATION

The calculation of the Fourier-Mellin transform can be quite intensive due to the requirement of logarithmic calculations for each pixel. However, this problem can be overcome to some extent as will be explained in chapter 6.

3.1.3.3.2 INTERPOLATION

Many image processing algorithms result in values for non-integer pixel positions. The quick and simple answer is to assign the pixel value to the nearest applicable pixel position by rounding up or down the calculated pixel position. However, this is not an ideal answer as errors are introduced so interpolation between pixel positions has been considered. General methods for implementing interpolation can be found in the

literature [Pratt op. cit.] and this allows pixel values to be generated which correspond to sub-pixel resolutions.

3.2 COLOUR

Colour is being used more often in processes which are steps towards the recognition of objects in an image. It is useful, for instance, where objects which look dissimilar in colour but not in intensity need to be recognised. As discussed in a previous chapter, there are many different ways to represent colour and the choice is often influenced by what process is to be performed. The areas in which colour has received most attention are segmentation, edge detection/enhancement and image enhancement but, while some of these tasks may be used to aid an object recognition algorithm, they are not on their own sufficient to recognise an object. Therefore, although there is a lot of literature on colour processes involving segmentation and enhancement there is little use made of colour in the recognition algorithm itself. This has meant that there has been relatively little literature to research before developing the colour object recognition process which is introduced and discussed in subsequent chapters.

The human perception of colour is better described using hue, saturation and intensity than the widespread red, green and blue system. Faugeras [1979] presented a model for human colour vision and showed that the perceptual space defined by the model could be structured as a vector space.

3.2.1 GENERAL APPLICATIONS

This section briefly gives some examples of the various uses of colour made by researchers in the image processing field before describing the use of colour in object recognition.

Colour image segmentation has been studied by many people including Celenk [1991] who used the CIELAB colour space to generate lightness, hue and chroma values which were then used in cluster detection. Healey [1992] uses intensity to generate edges for the segmentation which then uses the colour information. Ohta et al. [1980] proposed

three colour features and compared segmentation using their colour features with seven other colour representations and concluded that their features were effective. It should be noted that colour spaces such as HSI are not ideal for detecting colours using clustering due to the singularities and spurious modes and gaps described by Kender [op. cit.] and it has to be remembered that, although perceptually easy to understand, they are not uniform colour spaces such as those recommended by the CIE in 1976.

A number of researchers have investigated edge detection using colour including Robinson [op. cit.] who proposed various descriptions for colour edges, Cumani [1991] who used the second order derivative to find edges in RGB images and Nevatia [1977] who used the HSI colour representation. Jordan III and Bovik [1991] generated chromatic gradient matching constraints by calculating attributes from each channel of the RGB data whereas Liu and Yan [1994] attempted to enhance the colour edges using luminance and saturation in the CIELUV colour space.

Of particular interest, in the context of the research presented in later chapters, is the paper by Strickland et al. [1987] which shows that the saturation of a colour can be important in enhancing the edges in an image.

3.2.2 Colour and Recognition

Mital and Goh [1993] used the Fourier-Mellin transform to recognise colour objects. They identified a number of steps in the process which are feature extraction, FFT, log-polar transform, FFT and finally correlation with a reference image. However, the only stage where colour is involved is in the feature extraction stage. This involved the extraction of edges using the colour (HSI) information. The actual edge extraction was implemented by smoothing the image, applying an orientation independent Laplacian filter and then smoothing again. The result is then thresholded, differentiated again and finally the edges are detected. This feature extraction process reduces the colour information to a single monochromatic component which is then the input to the Fourier-Mellin transform. There are two main disadvantages to this algorithm; the first is that it only attempts to match objects rather than quantify any of the potential variances that are removed through the processing; the second disadvantage is due to the way the colour

information is treated. This process is able to match colour images to a certain extent since only objects containing a specified band of colours are allowed into the Fourier-Mellin input; it is also able to match similar shaped objects without using the colour information; however, it can not be used to match objects of different colours and also retain information about the colour of both objects. This is a big disadvantage since, using this algorithm, it would require multiple computations of the whole algorithm to match objects of defined colours. As will be seen in chapters 5 and 6, this disadvantage has been overcome using a novel way of representing the colour information.

The use of colour for recognition over a range of lighting conditions was proposed by Berry [op. cit.]. The recognition process used a Hough transform on a red image to find the position of a number of coloured circles. Once the positioning was accomplished, the colour of each circle was found using a set of features (RGB and hue and saturation calculated using different methods [Ohta et al. op. cit., Faugeras op. cit.]) by comparing the values of the features with previously calculated reference features. The features considered were said to be invariant to intensity although this is not entirely true since the accuracy of the saturation value is affected by intensity, [Kender op. cit.]. This algorithm means that two main processes are involved; the first for monochromatic (red only) processing and the second for chromatic processing. This would be improved if only one process was required so that the chromatic processing could be combined with the monochromatic processing.

The colour object recognition process proposed in later chapters is a much neater, more useful system than that demonstrated by Mital and Goh [op. cit.] and enables the colour to be used in recognition in parallel with the recognition of the object shape which is an improvement upon the Berry technique.

It is clear from the evidence in this chapter that much work is still to be done on the area of colour object recognition and the following chapters propose some solutions to problems in this subject area.

4. COLOUR REPRESENTATION

This chapter introduces a novel representation for chromatic information which enables information about the colour content in an image to be described where previously only monochromatic details were able to be used. In addition, the representation has the useful property that it requires no extra processing to be calculated during the recognition algorithms presented in the following chapters. In this chapter, this novel representation is described together with practical considerations when converting to and from the RGB colour space.

The use of colour in object recognition has a very short history and although colour representations can be of use in the recognition process, none make the most of the colour information at the disposal of the recognition algorithm. This can be the fault of the algorithm and/or the colour representation. These two shortcomings are overcome in this chapter and the next. Firstly, the novel colour representation is described before its use is shown in object location in the next chapter.

4.1 IZ REPRESENTATION

The IZ representation has been introduced because the HSI representation itself was not considered compact enough since the traditional method for representing HSI values is to use three separate (real) values. Since some research on monochrome recognition has used the frequency domain it was decided that the use of complex numbers to represent colour had the potential to be exploited. The pixel values of the novel colour representation used in this research can be generated directly from the RGB values to obtain a further three values representing intensity, hue and a variation of saturation. Each pixel is represented by a real number for the intensity, I, and a complex number, Z, which holds the chromatic information. The I remains separate for two reasons. The first is that a complex number cannot represent three quantities and so cannot fully describe a colour pixel. The second reason is that since the I is separate, traditional monochromatic processing can be accomplished if required.

Apart from the method of coding the colour information leading naturally from the use of the HSI space, it was decided that where human interaction is concerned, such as for the

specifying of some recognition criteria, the use of hue, saturation and intensity should be retained rather than other non-perceptual values which are more difficult for a human to describe.

4.1.1 IZ GEOMETRY

The argument of the complex number is directly equivalent to the traditional definition of hue, and the modulus of the complex number is similar to saturation. Although the term *saturation* is used in this chapter, it should be noted that this refers not to the conventional representation of saturation but to the value of |Z|, unless otherwise stated. Calculation of the conventional representation of saturation from the IZ domain is described in more detail in a later section.

The IZ representation is described in relation to the RGB cube since the R, G and B values are often the parameters available in a practical image processing system. In the following discussion, 8-bit values are assumed for the red, green and blue components of a pixel. Each pixel component therefore can have values between 0 and 255. The (intensity) line R = G = B is normal to the Z plane and intercepts the Z plane at its origin. This can be visualised by imagining the Z plane as a flat surface with the RGB cube balanced on the surface, on its black corner, with the grey line vertical. The R, G and B axes, when projected onto the Z plane, are as shown in figure 4.1, and labelled R', G' and B' respectively since the R, G and B axes are not parallel to the Z plane. The real axis of the Z plane was chosen to coincide with R'.

Z values are confined to a hexagon on the complex plane, by the geometry of the RGB colour space; this can be observed by looking down on the Z plane from above the RGB cube. The real axis in the complex plane is coplanar with (but not parallel to) the red direction. Intensity may be regarded as normal to the complex plane, that is, I values are orthogonal to the Z values. Thus in the complex plane, red is situated at 0^o, green is at 120^o and blue at 240^o.

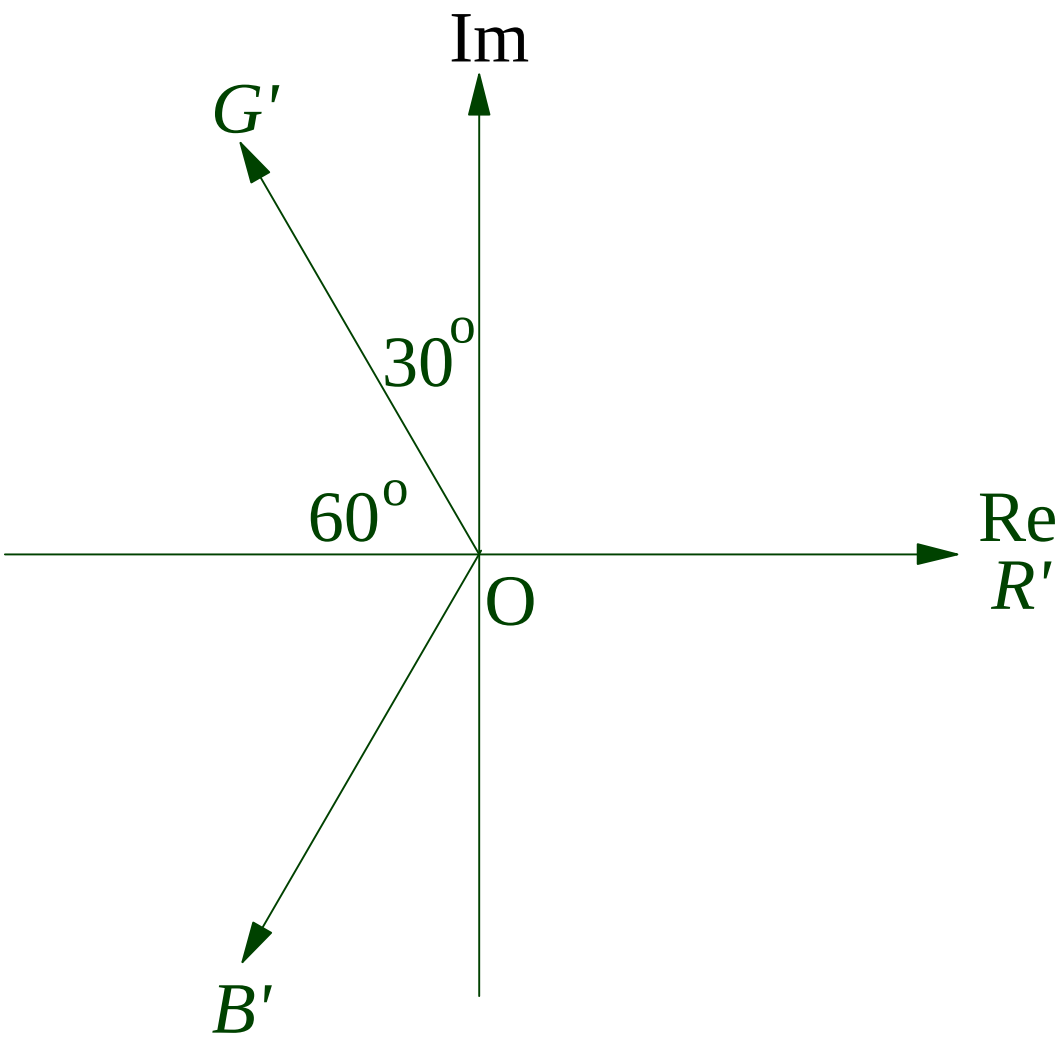


Figure 4.1: The complex Z plane

For any point on the edge of the Z plane hexagon, (where at least one of the R, G or B values is 255), there is only one value of intensity possible for which conversion back to valid RGB space would occur. However, for any other point in the Z plane, there is a range of intensities for which correct conversion could be performed. Figure 4.2 shows that for a single pixel within the hexagon there would be many possible intensities due to the possible combinations of the RGB components. As |Z| (saturation) decreases this range of valid intensities increases. When the point is at the centre of the hexagon on the grey line, (the OW line in figure 4.3), the intensity can have a value anywhere between 0 and 255.

Apart from the advantage of separating the intensity data from the chromatic data, an advantage of IZ space over conventional representations using hue is that there is no hue discontinuity near to red (0^o, 360^o) due to its angular nature.

Although its proposed use is in object recognition which is discussed in the next chapter, this representation is also useful in the visualisation of (complex) Fourier domain data. Usually the modulus of the Fourier transform is displayed separately to its phase. Using the IZ notation however, the phase of each complex number can be displayed in colour, with no discontinuity, and the modulus of each complex number can be displayed as the saturation. This, combined with separate intensity data, provides a colour image which displays both phase and modulus data in one image.

4.1.2 COLOUR CONVERSION

The conversion of colour pixels from RGB to IZ format is relatively simple, although the transformation back from IZ to RGB is complicated by the possibility that at least one of the resulting RGB values could be outside the valid 0 to 255 range.

4.1.2.1 CONVERSION FROM RGB TO IZ

The intensity is calculated using the traditional equation (4.1a). The Z value is computed as shown in equation (4.1b), using the geometry of the R, G and B values with reference to the Z plane.

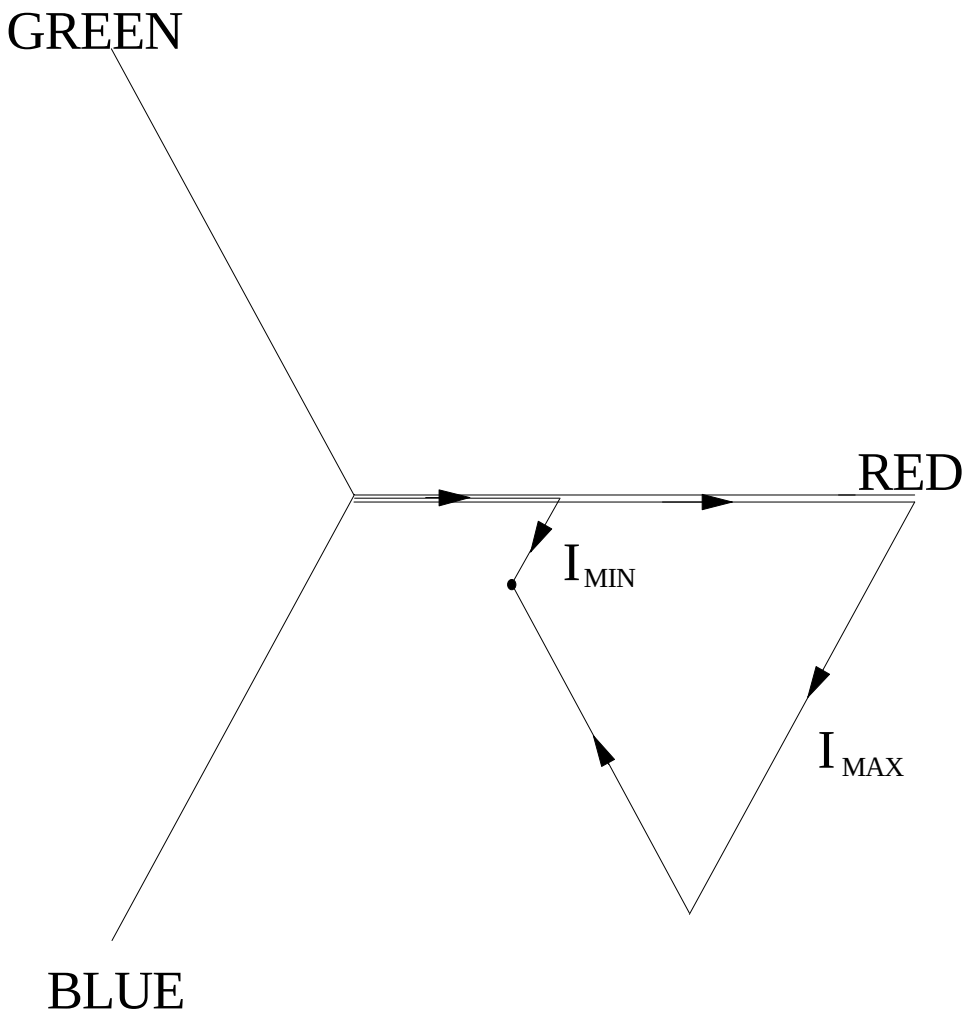


Figure 4.2: Horizontal slice of RGB space perpendicular to the grey line

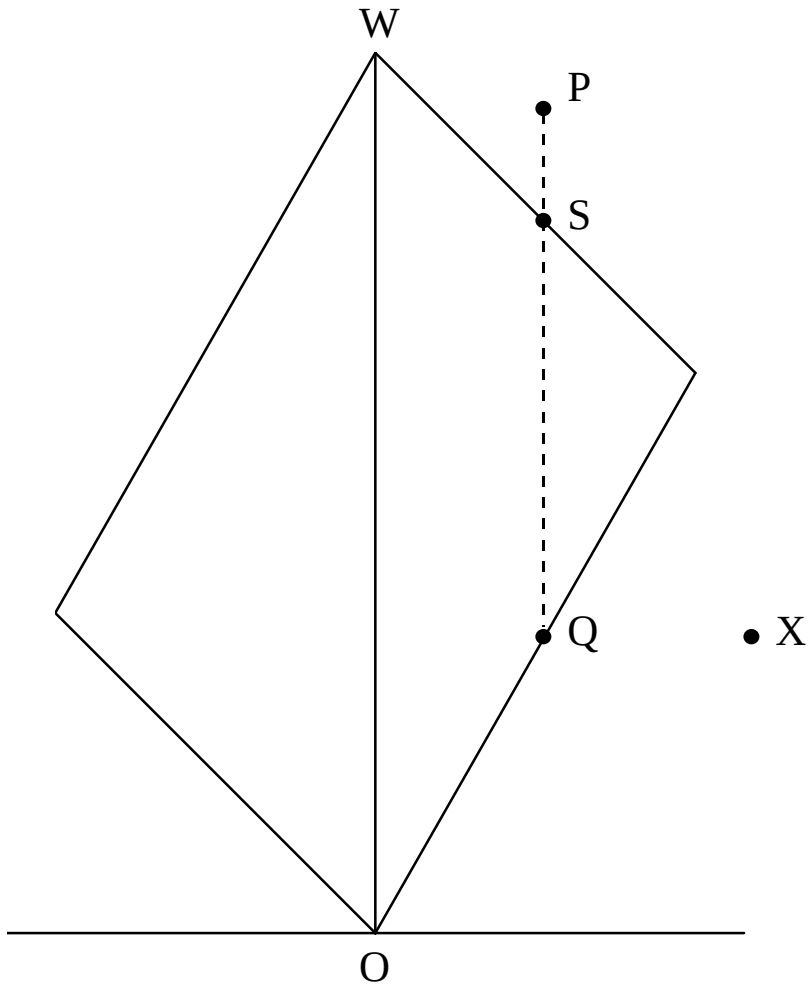


Figure 4.3: Cross-section of RGB space perpendicular to the Z plane

$$I = \frac{r+g+b}{3} \tag{4.1a}$$

$$Z = \cos q\left(r - \frac{g+b}{2} + j\sqrt{3}\frac{g-b}{2} \right) \tag{4.1b}$$

The factor cosθ, which may be omitted, is a scale factor which represents the cosine of the angle between each of the R, G and B axes and the Z plane because the RGB axes are not in the complex plane.

4.1.2.2 CONVERSION FROM IZ TO RGB

If processing has been performed in the IZ domain and it is required to convert back to RGB space, such as for the purpose of displaying the resulting image, the conversion from IZ to RGB is calculated using equations (4.2a -c):

$$r = I + \frac{2}{3}\Re(Z) \tag{4.2a}$$

$$g = I + \frac{\Im(Z)}{\sqrt{3}} - \frac{\Re(Z)}{3} \tag{4.2b}$$

$$b = I - \frac{\Im(Z)}{\sqrt{3}} - \frac{\Re(Z)}{3} \tag{4.2c}$$

where $\Re(Z)$ and $\Im(Z)$ are the real and imaginary parts of Z respectively. It should be noted that after processing has been performed on the IZ data, the recovered RGB values could be outside the valid RGB space. There are two cases where this could happen and which are described below in the next section.

4.1.2.2.1 CONVERSION PROBLEMS

Approaches to the problem of gamut clipping which deal with the different methods which can be used to reduce clipping errors with regard to mapping between different devices are documented in the chapter by Morovic [op. cit. Sangwine and Horne 1998a]. Figure 4.3 shows a cross-section through the RGB cube normal to the Z plane which illustrates the cases that could occur after processing has been performed on the values in the IZ domain. It should be noted that the cross-section varies with the argument of the

Z value. For the point P, which is outside the valid range of RGB values, there is a set of intensities for which its modulus is valid. These are contained within the intensity range Q to S.

The point X, however, has a |Z| value such that there is no intensity value which will produce a point within the valid RGB space. The method chosen to deal with this problem depends upon whether the hue or the saturation is more important; the method described below retains the hue and alters the saturation as little as possible since variations in hue are more easily distinguished than variations in saturation. In addition, the reduction of saturation is guaranteed to result in the RGB values being within the valid RGB space. However, if the hue only was altered, the saturation could still be so large that the RGB values would never be acceptable.

4.1.2.2.1.1 Altering Intensity Only

The value of intensity at S could be chosen as a suitable point in RGB space to represent the invalid value P and is obtained by subtracting equal amounts from each of R, G and B (white or grey subtraction). Equally, if the point P were directly below Q, a valid point could be obtained by adding white. The values of R, G and B required to generate the minimum and maximum intensities are shown graphically in figure 4.4. To calculate these values, equations are shown below which quantify exactly the amount that the R, G and B values need to be altered so that they fall within the RGB colour cube. It can be seen from figure 4.4 that the 120° sectors for which the colour components are zero or 255 are in opposing positions. This has to be taken into account when calculating the minimum and maximum possible intensity.

The following equations determine the values of R, G and B required to preserve the hue and saturation but alter the intensity. The minimum intensity can be obtained using equation (4.3):

For $0° < \theta \leq 120°$ $\quad R = \Re + \frac{\Im}{\sqrt{3}} \quad G = \frac{2\Im}{\sqrt{3}} \quad B = 0$

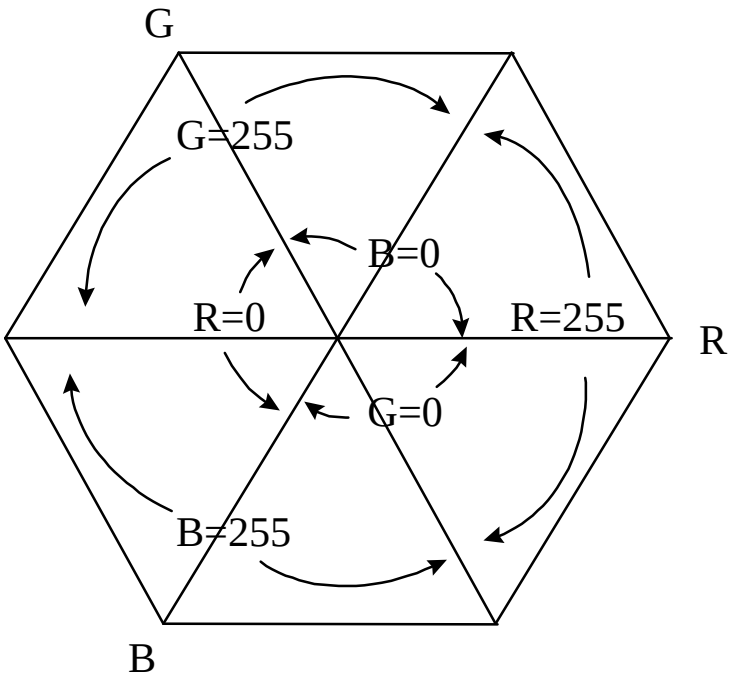


Figure 4.4: Horizontal slice of RGB space perpendicular to the grey line showing values of RGB required to generate minimum and maximum intensities.

$$\begin{array}{lllll} \text{For } 120° < \theta \le 240° & R = 0 & G = -\Re + \frac{\Im}{\sqrt{3}} & B = -\Re - \frac{\Im}{\sqrt{3}} & \\ \text{For } 240° < \theta \le 360° & R = \Re - \frac{\Im}{\sqrt{3}} & G = 0 & B = -\frac{2\Im}{\sqrt{3}} & (4.3) \end{array}$$

If the maximum intensity is required to be calculated then equation (4.4) should be used.

$$\begin{array}{lllll} \text{For } 240° < \theta \le 60° & R = 255 & G = 255 - \Re + \frac{\Im}{\sqrt{3}} & B = 255 - \Re - \frac{\Im}{\sqrt{3}} & \\ \text{For } 60° < \theta \le 180° & R = 255 + \Re - \frac{\Im}{\sqrt{3}} & G = 255 & B = 255 - \frac{2\Im}{\sqrt{3}} & \\ \text{For } 180° < \theta \le 240° & R = 255 + \Re + \frac{\Im}{\sqrt{3}} & G = 255 + \frac{2\Im}{\sqrt{3}} & B = 255 & (4.4) \end{array}$$

It should be obvious that any intensity above the minimum allowed can be generated by adding the same value onto the values of R, G and B calculated using equation (4.3). This is permitted until the maximum intensity is reached.

4.1.2.2.1.2 Altering Intensity And Saturation

On conversion, the range between the maximum and minimum values of R, G and B which represent point X in figure 4.3, exceeds 255; this means that white cannot be added and black cannot be subtracted since this process affects all components equally and the range between the maximum and minimum values of R, G and B would still exceed 255. Therefore, let the new middle value of R, G and B, equal *m*, see equation (4.5), where the maximum, middle and minimum value of the RGB components equal max(R,G,B), mid(R,G,B) and min(R,G,B):

$$m = \frac{255(\text{mid}(R,G,B) - \min(R,G,B))}{\max(R,G,B) - \min(R,G,B)} \qquad (4.5)$$

and then let the new maximum value and new minimum value equal 255 and 0 respectively [Levkowitz and Herman op. cit.]. An alternative equation, but one which requires real numbers is shown in equation (4.6):

$$m = \frac{255 \sin q}{\sin(120° - q)} \qquad (4.6)$$

This method retains the hue angle whilst the saturation is reduced. This is not as arbitrary as methods which involve clipping the RGB values (which alters both hue and saturation) in that only the saturation is reduced and this is done as little as is necessary for successful conversion.

4.1.2.3 HUE

As has already been stated, hue can easily be derived from the Z values, equation (4.7), by calculating the argument of the complex number so that:

$$Hue = \arctan\left(\frac{\frac{\sqrt{3}}{2}(G-B)}{R-\frac{(G+B)}{2}}\right) = \arctan\left(\frac{\sqrt{3}(G-B)}{2R-G-B}\right) \qquad (4.7)$$

This is equal to the traditional definition of hue.

4.1.2.4 SATURATION

As has been previously stated, conventional saturation is slightly different to the value of saturation, |Z|, derived from the Z value. It is thus necessary, to show how the IZ representation relates to HSI. It is shown below how |Z| relates to the traditional equation for saturation, S, which is shown in equation (4.8).

$$S = 1 - \frac{\min(R,G,B)}{I} \qquad (4.8)$$

The representation of the modulus of Z as the saturation is complicated by the fact that the saturation of a pixel may be different for identical Z values; this is due to the saturation of a pixel being dependent not only on its perpendicular distance from the grey line but also on its intensity, as discussed in chapter 2. This means that the intensity for a pixel is needed to calculate S. If this intensity is known, the maximum |Z| value for that intensity can be calculated, and consequently S is known since $S = |Z|/|Z|_{\max}$. Figure 4.5 illustrates the values which are used to calculate the saturation of a pixel. It must be remembered that the cross-section shown in figure 4.5 will be different depending on the hue value.

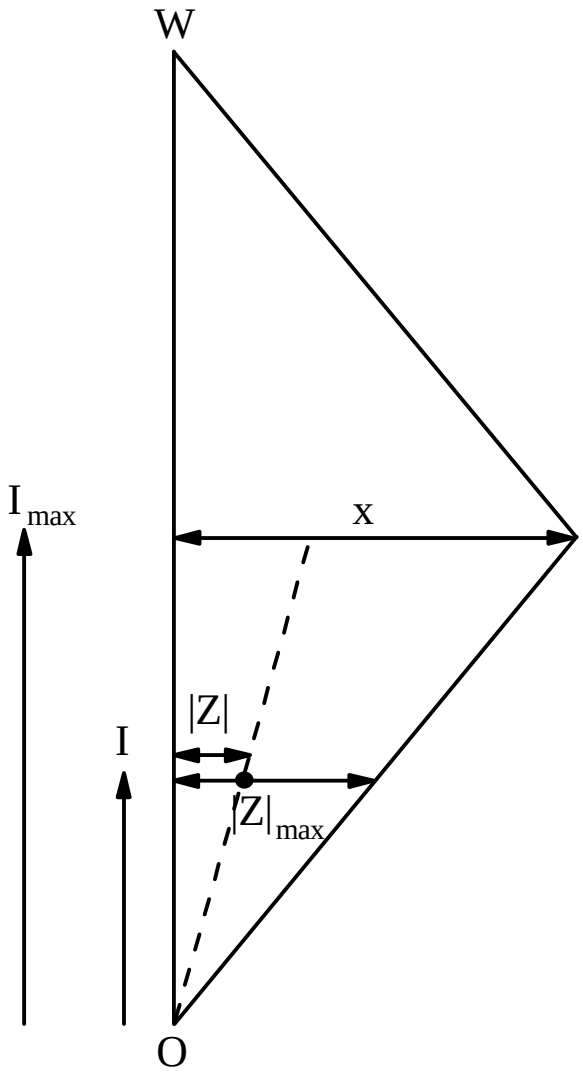


Figure 4.5: Saturation diagram

4.1.2.4.1 OBTAINING SATURATION FROM IZ VALUES

The equation shown below, equation (4.9), states a simplified expression for calculating conventional saturation from the IZ domain. The values of intensity are unscaled.

$$S = \frac{2|Z|\cos(60° - \Theta)}{K} \qquad (4.9)$$

where $K = I$, $I \leq 382{\cdot}5$; $K = 765-I$, otherwise.

where $\theta = \arg(Z)$ and

$\Theta = |\theta|$, $-60° < \theta \leq 60°$

$\Theta = |\theta-120°|$, $60° < \theta \leq 180°$

$\Theta = |\theta+120°|$, $-180° < \theta \leq -60°$

The use of K is due to the geometry of the colour space as can be seen in figure 4.5. The value of θ has to be constrained within 120 degree segments for the equation to be valid. Each of the three segments have red, green or blue as the maximum value. For instance, in the range $-60° < \theta \leq 60°$, red is set at the maximum value of 255. This is explained further in the next section which illustrates how equation (4.9) was obtained. Having done this, equation (4.9) is proved to be equal to the conventional method of calculating saturation.

Equation (4.10) shows how saturation can be calculated, where *x* is the maximum possible vector length for a given hue as shown in figure 4.5.

$$S = \frac{|Z|}{|Z|_{max}} \quad \text{where} \quad \frac{|Z|_{max}}{x} = \frac{I}{I_{max}}$$

$$\therefore S = \frac{|Z|}{I} \cdot \frac{I_{max}}{x} \qquad (4.10)$$

If saturation is to be calculated and only the real I and complex Z values are available, I_{max} and x must be obtained. This is done with the aid of figure 4.6. The hue determines the maximum allowable value of x and consequently the value of I_{max}.

For the largest value of x, the lowest possible (unscaled) value of I_{max} is 255 and the highest possible value is 510. For an arbitrary value of θ in figure 4.6 where θ can vary between 0 and 60 degrees, the unscaled intensity at the point x is equal to 255 + L where:

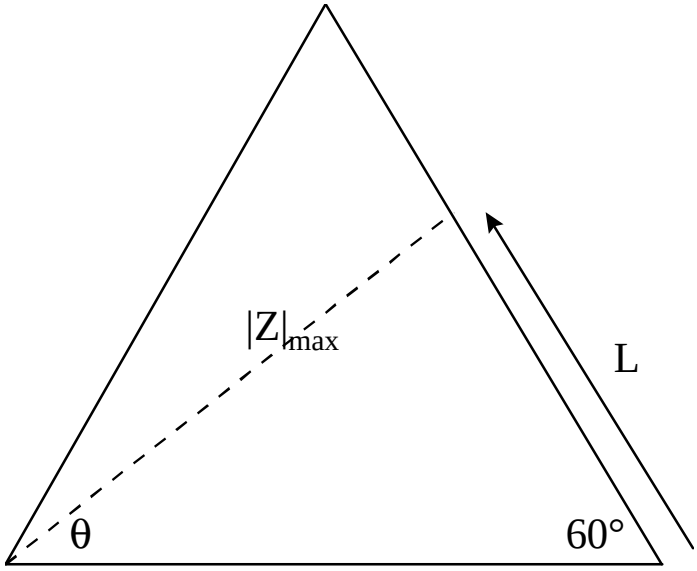


Figure 4.6: Triangle segment from hexagon.

$$L = \frac{255\sin q}{\sin(120° - q)}$$

while

$$x = \frac{255\sin 60°}{\sin(120° - q)} = \frac{255\sqrt{3}}{2\sin(120° - q)}$$

These equations can now be substituted into equation (4.10) and manipulated to obtain equation (4.9):

$$S = \frac{|Z|}{I} \cdot \left(255 + \frac{255\sin q}{\sin(120° - q)}\right) \cdot \left(\frac{2\sin(120° - q)}{255\sqrt{3}}\right)$$

$$S = \frac{|Z|}{I} \cdot \frac{2}{\sqrt{3}} (\sin(120° - q) + \sin q)$$

$$S = \frac{|Z|}{I} \cdot \frac{2}{\sqrt{3}} (2\sin 60° \cos(60° - q))$$

$$S = \frac{|Z|}{I} \cdot \frac{2}{\sqrt{3}} \left(\sqrt{3}\cos(60° - q)\right)$$

$$S = \frac{2|Z|}{I}\cos(60° - q)$$

4.1.2.4.2 Proof of the IZ Saturation Equation

It is necessary to show how the |Z| value is related to the traditional value for saturation since |Z| is one of the values which could be used in characterising a colour.

Proof of equation (4.9) such that $S = \frac{2|Z|\cos(60° - q)}{K} = 1 - \frac{3\min(RGB)}{R + G + B}$ is as follows:

$S = \frac{2|Z|\cos(60° - q)}{K}$ = $\frac{2|Z|}{K}\left(\frac{1}{2}\cos q + \frac{\sqrt{3}}{2}\sin q\right)$ by the double angle formula.

Let $Z = \Re + j\Im$.

$\cos q = \frac{\Re}{\sqrt{\Re^2 + \Im^2}}$ and $\sin q = \frac{\Im}{\sqrt{\Re^2 + \Im^2}}$

(Remember that θ is constrained so that $-60° < \theta \leq 60°$. The above line shows the equations for cosθ and sinθ assuming that θ is above 0°. If θ is below 0° then sinθ is altered since $Z = \Re - j\Im$).

Then substitute for cosθ and sinθ and hence:

$$S = \frac{2}{R+G+B} \cdot \sqrt{\Re^2 + \Im^2} \cdot \left[\frac{\Re}{2\sqrt{\Re^2 + \Im^2}} + \frac{\sqrt{3}\Im}{2\sqrt{\Re^2 + \Im^2}} \right]$$

Cancelling the numerator and denominator we obtain:

$$S = \frac{1}{R+G+B} \cdot \left[\Re + \sqrt{3}\Im \right]$$

Substituting for $\Re$ and $\Im$ using equation (4.1b) this gives:

$$S = \frac{1}{R+G+B} \cdot \left[R - \frac{G}{2} - \frac{B}{2} + \frac{3G}{2} - \frac{3B}{2} \right]$$

and obtain by simplification:

$$S = 1 - \frac{3B}{R+G+B}$$

Thus the expression for saturation is obtained since the hue value is assigned to a value between 0° and 60° when starting the proof and therefore B must be the minimum value. If the proof was followed through with $-60° < \theta \leq 0°$ the minimum value results in G. This proof works for all values of θ assuming the constraints previously mentioned. Due to these constraints, if 120° has to be added to or subtracted from the hue, the values of R, G and B in relation to $\Re$ and $\Im$ are also rotated around which is how different minimum components are generated.

The IZ representation has been published, [Thornton and Sangwine 1995], in a paper which also illustrates its use in an object location algorithm. Having recognised the use of a complex coding scheme for colour, McCabe et al. [1997] have used the Yuv colour space rather than the HSI representation to generate complex numbers where the real component is related to u (red-green) and the imaginary component is related to v (yellow-blue). Using this coding, the extraction of colour features using linear filters in what they call the spatio-chromatic Fourier transform domain has been successfully implemented.

This chapter has introduced the novel IZ complex colour representation and demonstrated how to convert between RGB space and the IZ values. It has also explained the problems that could be encountered in converting back to valid RGB space

from processed IZ values. The IZ values have also been related to the HSI colour space; although hue and intensity are easily obtained from the IZ values, the traditional value of saturation is slightly harder to obtain so an equation for saturation from the IZ values has been proposed and proved. The use of this colour representation in an object location scheme is now discussed in the next chapter.

5. COLOUR OBJECT LOCATION

This chapter builds upon the knowledge of the IZ colour representation, which was described in the previous chapter, to enable the location in an image of multiple identical objects which differ only in colour by using a single reference image containing a known coloured object.

5.1 FOURIER TRANSFORM DISPLAY

It is worth mentioning at this point how Fourier transform data is displayed and how the spatial-frequency data is arranged in an image. The decision as to how to display Fourier transformed data requires some understanding as to the content of the data since it is made up of complex (not real) numbers and the DC (or zero frequency) value has to be dealt with as explained below.

5.1.1 DISPLAY OF FOURIER TRANSFORM SPECTRUM

To display the spectrum, for each pixel the square is taken of both the real and imaginary parts of the complex number and then added. To then represent transformed images using an 8-bit number it is necessary to alter the data so that it can fit in the range 0 .. 255. This is done using logarithmic compression, equation (5.1). This is because the DC, or zero frequency, term which appears in the top left corner of the transformed image, is the sum (or average depending on where the division is performed on the forward or inverse FT) of all the pixels in the original image and contains a much greater value in comparison with other pixel values.

$$\text{Compressed value} = 255*\log(1+|F(u,v)|)/\log(1+|F(0,0)|) \qquad (5.1)$$

If the image were simply scaled linearly to be within 0 .. 255 to fit the pixel values into 8-bits, because the DC term is originally so large in comparison with the other pixel values, the other pixel values would be zero.

5.1.2 DISPLAY OF FOURIER TRANSFORM PHASE

The display of the phase data is not as straightforward as it at first appears. The problem is that phase can vary between 0° and 360° where values near those two extreme angles represent similar phases. So the option of scaling the phase data so that it is constrained within 0 to 255 does not give an easily interpreted picture of phase values since black and white will represent very similar phases. This problem can be overcome by displaying phase data using different hues.

In this way it is possible to combine both the magnitude and phase data so that they can be displayed at the same time; intensity is used to display the magnitude and hue displays the phase.

5.2 OBJECT LOCATION

In the case of Fourier transforms, a significant advantage of the colour representation is that transformation of the chromatic image into the frequency domain requires only one Fourier transform on a complex image (two-dimensional array of complex numbers) whereas using a traditional RGB model requires three[5.1] Fourier transforms of real images to be calculated in order to process the chromatic information. The resulting spectrum contains more information about the image than would be contained in the FFT of the intensity data.

In this chapter, colour images described by complex (Z) pixels have been used for determining the location of an object. This method of coding the colour information results in an increase in knowledge about an image without a consequent increase in computation.

Two established frequency domain techniques for locating objects; Phase Correlation, and the Cepstrum procedure, have been described in chapter 3 and the advantages of combining the colour representation and these location techniques are illustrated in this chapter. It is shown that, using the novel colour representation, both methods are able to

[5.1] The computational load can be reduced by using the Hartley transform (HT) [Hartley 1942, Bracewell 1983, Bracewell 1986] or the separable properties of the Fourier transform (FT) so that three HTs or one FT and one HT would be required.

produce an intensity peak in either a correlation surface or a spatial domain surface corresponding to the displacement of an object in comparison with a reference image. However, this peak is also able to indicate the colour of the displaced object. If the intensity image were used in the location procedure, the object would be found but it would not be possible to estimate its colour. In addition, the use of colour aids in locating objects where the intensity of the object may be the same as the background or where the image contains a cluttered background; for instance an object of a certain colour may be required to be located but there may be other different coloured but similarly shaped objects also contained in the image; this is overcome through the use of the complex pixel representation.

5.2.1 PHASE CORRELATION

The technique of phase correlation has been discussed in chapter 3 using monochrome inputs. This section will detail its use in conjunction with the IZ representation introduced in chapter 4. Although the phase correlation equation has previously been stated, for ease of reference it will be reiterated here in equation (5.2) along with the equation detailing the relationship between the position of the object in each image and the position of the peak in the phase correlation surface, equation (5.3).

$$P = F^{-1}\left(\frac{G_1 G_2^{*}}{\left|G_1 G_2^{*}\right|}\right) \qquad (5.2)$$

$$\text{Displacement of object}_{x,y} = (\text{Reference}_{x,y} - \text{Object}_{x,y}) \bmod \text{Image Size} \qquad (5.3)$$

5.2.1.1 USING THE IZ COLOUR REPRESENTATION

The method that was used by Kuglin and Hines [op. cit.] for the phase correlation of intensity images can be used for phase correlation using the IZ representation. Of interest here is the use of the chromatic information so, specifically, the Z data will be used. However, because a complex number containing colour information is used, the result of the phase correlation can discriminate between similar shaped objects of different colours. The phase correlation surface is represented by complex numbers for

each pixel position, the magnitude of each indicates whether objects have been matched while the argument of each indicates hue differences between the two images undergoing object location. Since the modulus of Z is an indication of the saturation of a pixel colour, the matching of objects is accomplished using this 'saturation' indication. Supporting this use of saturation, it was observed by Strickland et al. [op. cit.] that the saturation component often contains more high frequency energy than the intensity data and is therefore of use in image enhancement.

The idea of matching single objects in each image, see figures 5.1 to 5.3, can be extended to correlating a single object in one (reference) image and multiple similar objects in the second image, see figures 5.4 to 5.6 and table 5.1. Phase correlation will still detect the location of the objects and the difference in colour between the reference object and those in the second image. This is shown in the equations shown below for the 1D discrete case; it can be extended to the 2D discrete case.

5.2.1.1.1 THEORY

This section illustrates by means of equations how the phase correlation works for retaining colour information about an object. The mathematics below considers a continuous-time 1D case rather than a 2D case as it is easier to understand; however, it is simple to apply to the 2D case.

Let signal g_1 contain nothing except a coloured object of single hue and saturation, which has a width W1 starting at time D1 represented by a series of complex numbers of magnitude S and argument θ. Its Fourier transform (G_1) is:

$$G_1(u) = \frac{1}{\sqrt{N}} \int_{x=D1}^{D1+W1} S e^{jq} e^{\left(-j2pux/N\right)} dx$$

$$G_1(u) = \frac{S e^{jq}}{\sqrt{N}} \int_{x=D1}^{D1+W1} e^{-j2pux/N} dx$$

$$G_1(u) = \frac{S e^{jq}}{\sqrt{N}} \left[e^{\frac{-j2pu}{N}(D1+W1)} - e^{\frac{-j2pu}{N}D1} \right]$$

Let signal g_2 contain two objects of similar shape but different colours and saturations to the object in image g_1, starting at different times D2 and D3 and of widths W2 and W3

and described by different magnitudes T and U and different arguments ϕ and α such that:

$$G_2(u) = \frac{T e^{jf}}{\sqrt{N}}\left[e^{\frac{-j2pu}{N}(D2+W2)} - e^{\frac{-j2pu}{N}D2} \right] + \frac{U e^{ja}}{\sqrt{N}}\left[e^{\frac{-j2pu}{N}(D3+W3)} - e^{\frac{-j2pu}{N}D3} \right]$$

To calculate the phase correlation, first take the conjugate of G_2:

$$G_2^*(u) = \frac{T e^{-jf}}{\sqrt{N}}\left[e^{\frac{j2pu}{N}(D2+W2)} - e^{\frac{j2pu}{N}D2} \right] + \frac{U e^{-ja}}{\sqrt{N}}\left[e^{\frac{j2pu}{N}(D3+W3)} - e^{\frac{j2pu}{N}D3} \right]$$

then multiply G_1 by G_2^*:

$$G_1(u)G_2^*(u) = \frac{S e^{jq} T e^{-jf}}{N}\left[e^{\frac{-j2pu}{N}(D_1+W_1-(D_2+W_2))} - e^{\frac{-j2pu}{N}(D_1+W_1-D_2)} - e^{\frac{-j2pu}{N}(D_1-(D_2+W_2))} + e^{\frac{-j2pu}{N}(D_1-D_2)} \right]$$

$$+ \frac{S e^{jq} U e^{-ja}}{N}\left[e^{\frac{-j2pu}{N}(D_1+W_1-(D_3+W_3))} - e^{\frac{-j2pu}{N}(D_1+W_1-D_3)} - e^{\frac{-j2pu}{N}(D_1-(D_3+W_3))} + e^{\frac{-j2pu}{N}(D_1-D_3)} \right]$$

If the objects in the signals have the same shape the values of W1, W2 and W3 are identical and the above equation simplifies to give:

$$G_1(u)G_2^*(u) = \frac{S T e^{j(q-f)}}{N}\left[2e^{\frac{-j2pu}{N}(D_1-D_2)} - e^{\frac{-j2pu}{N}(D_1-D_2+W_1)} - e^{\frac{-j2pu}{N}(D_1-D_2-W_2)} \right]$$

$$+ \frac{S U e^{j(q-a)}}{N}\left[2e^{\frac{-j2pu}{N}(D_1-D_3)} - e^{\frac{-j2pu}{N}(D_1-D_3+W_3)} - e^{\frac{-j2pu}{N}(D_1-D_3-W_3)} \right]$$

After normalising the values and inverse Fourier transforming to obtain phase correlation, the exponentials in the above equation appear as impulses whose positions are dependent on the translation of the objects between signals g_1 and g_2. In addition, the phase information due to the arguments of the input complex numbers is retained so it is possible to distinguish between objects of different hues but identical shapes.

5.2.1.1.2 RESULTS

The effect of the above result can be seen in the figures and table in this chapter. The following figures illustrate object detection using phase correlation of (complex) colour images. Figures 5.1 and 5.2 are colour images containing 256×256 pixels. Each contains

objects which in this example are squares of size 30×30 pixels. The origin is assumed to be at the top left position. Figure 5.1 shows the reference image which contains a square starting at x,y position (100,100) and containing 0° hue (red) while the image to be correlated with the reference image, figure 5.2, contains two squares of hue 60° (yellow) and 240° (blue) and starting position (50,150) and (180,210) respectively. According to theory, the peaks in the phase correlation surface should appear at (176,146) and (50,206) with hues of 120° (green) and 300° (magenta) respectively which is the case. Figure 5.3 shows the positions of the peaks although the argument of each peak can not be seen from the figure.

Having illustrated the simple case of a single object in each image, the situation of multiple objects is now investigated. Figure 5.4 shows the reference image containing a single object. Figure 5.5 shows the object image containing multiple objects of identical shape to the reference object but of differing colours. Figure 5.6 illustrates the phase correlation surface which results from phase correlating the images shown in figures 5.4 and 5.5. As can be seen, a number of peaks are generated corresponding to the displacement of the objects in the second image in comparison with the reference image. Each of these peaks represents a complex number and the argument of each of the complex numbers is shown in table 5.1 and compared with the values expected. Table 5.1 shows the position and height of each of the peaks (which are as expected) and also what hue is expected and found at each peak position. The hue at each peak position is, in practice, not identical to the expected hue although it is very close. This is due to the interaction of the correlation peaks for each square.

The next three figures illustrate object detection with images containing more complicated details. Figures 5.7 and 5.8 are colour images containing 256×256 pixels. Each contains an object which is a square of size 10×10 pixels. Figure 5.7 shows the reference image which contains a yellow (hue = 60°) square starting at (x=20, y=10) on an arbitrary background. Figure 5.8 shows the object image containing a red (hue = 0°) square starting at (90,60) on the same background. When phase correlation is performed on the two images, a peak is obtained which correlates with the difference in position of the square in the object image in comparison with the square in the reference image.

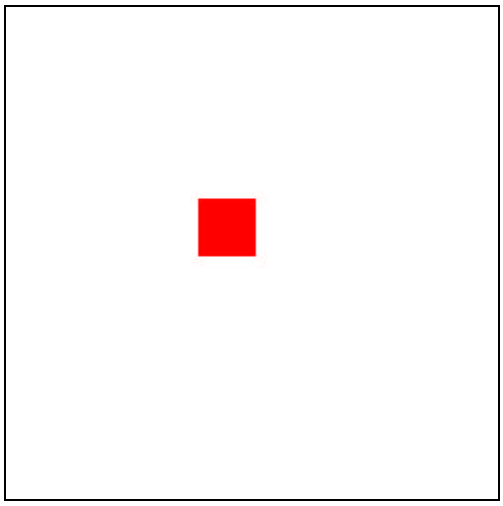


Figure 5.1 First input image

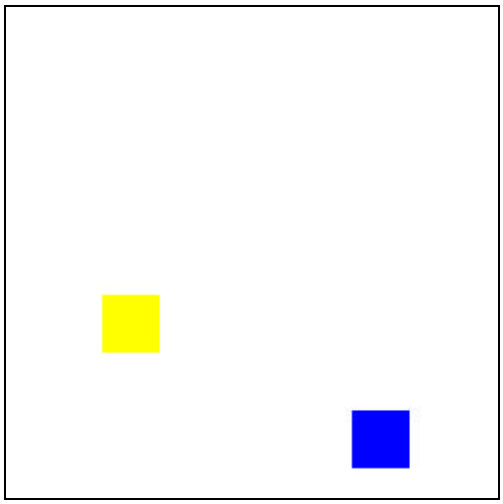


Figure 5.2 Second input image

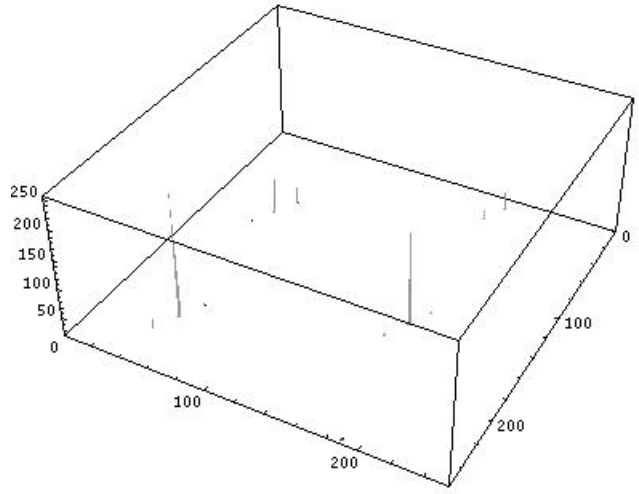


Figure 5.3: Phase correlation surface from figures 5.1 and 5.2

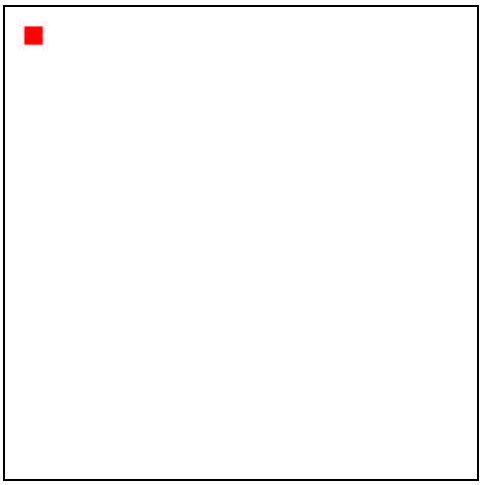


Figure 5.4: First input image

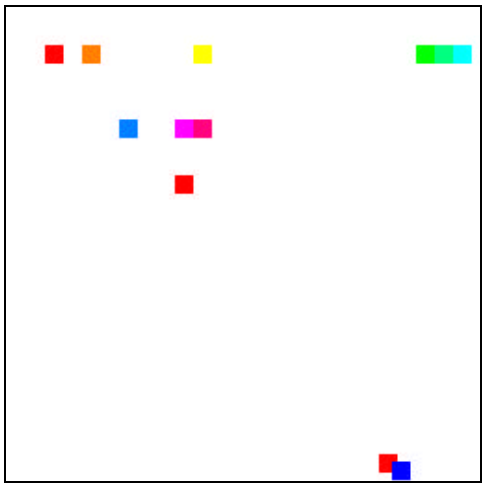


Figure 5.5: Second input image

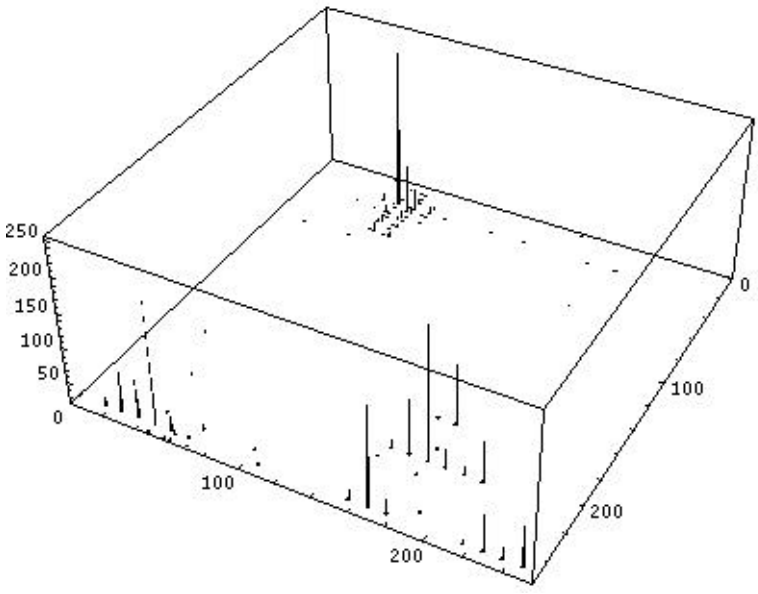


Figure 5.6: Phase correlation surface from figures 5.4 and 5.5

Peak Brightness	Peak Position (x, y)	Peak Argument	Expected Argument
255	(59, 22)	125.4°	120°
231	(166, 246)	299.5°	300°
216	(176, 206)	61.4°	60°
215	(46, 246)	243.2°	240°
138	(206, 206)	152.4°	150°
124	(226, 246)	330.1°	330°
110	(176, 176)	355.9°	0°
107	(246, 246)	2.9°	0°
104	(166, 206)	26.0°	30°
96	(26, 246)	167.0°	180°
95	(66, 26)	2.7°	0°
93	(36, 246)	205.0°	210°

Table 5.1 Phase correlation results from figures 5.4 and 5.5.

Figure 5.7: Yellow Square

Figure 5.8: Red Square

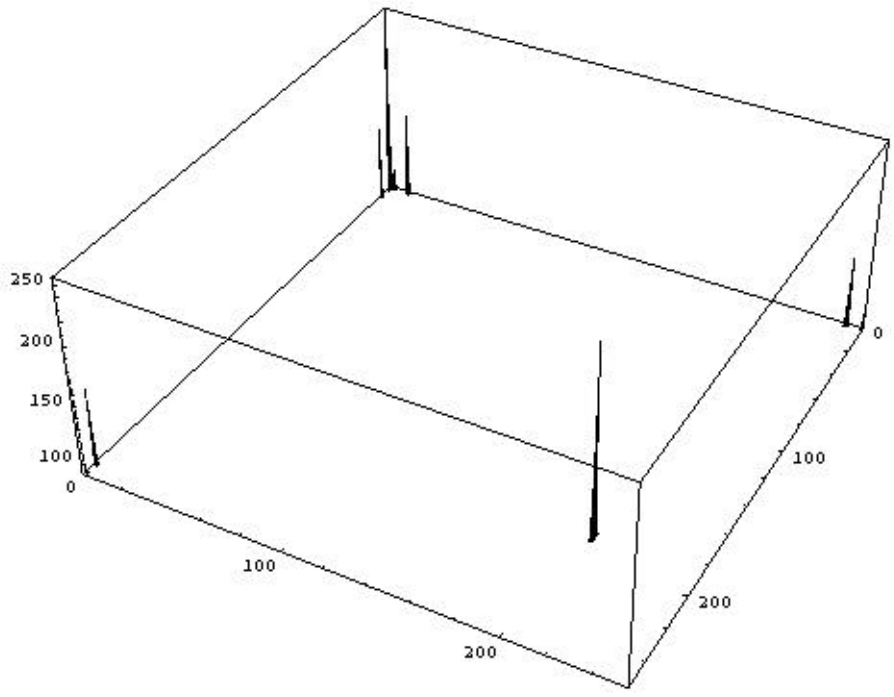


Figure 5.9: Phase correlation surface from figures 5.7 and 5.8

The peak is illustrated in figure 5.9 and occurs at (186,206) which is as expected from equation (5.3).

Having located the peak corresponding to the displacement of the object, the argument of the complex number representing the peak gives an angle whose value corresponds to the difference in colour between the square in the reference image and that in the object image. In the case of the images shown in figures 5.7 and 5.8, the argument of the peak in figure 5.9 is 60° which means that the object in the second image contains a colour which is 60° less than yellow, i.e. red. It should be noted that peaks exists around the origin due to the matching backgrounds which have not changed position in figures 5.7 and 5.8.

The next three figures illustrate object detection using phase correlation of (complex) colour images. Figures 5.10 and 5.11 are both colour images containing 512×512 pixels. Each image contains objects of different colours and varying sizes against a multicoloured background where the origin is assumed to be in the top left position. Before the capture of the second image, the red car, (whose reference position is (275,366)), is replaced by a similar, yellow, car (whose reference position is (451,357)), in a different position. When phase correlation is performed on the two images, a peak is obtained which correlates with the distance moved by the shape of the car in the object image in comparison with the reference image. The peak is illustrated in figure 5.12 and occurs at about (338,8) which is as expected. There are other correlation peaks near to the origin due to the non-translation of the background and other objects in the image. Having located the peak corresponding to the displacement of the object, the argument of the complex number representing the peak gives an angle whose value corresponds to the difference in colour between the car of interest in the reference image and that in the object image. In the case of the images shown in figures 5.10 and 5.11, the argument of the peak is around 300° corresponding to the difference in colour between the red (hue = 0º) and yellow (hue = 60º) cars. More results can be found in appendix B and a selection of the code that has been used in this research is shown in appendix C.

Figure 5.10: Red Car

Figure 5.11: Yellow Car

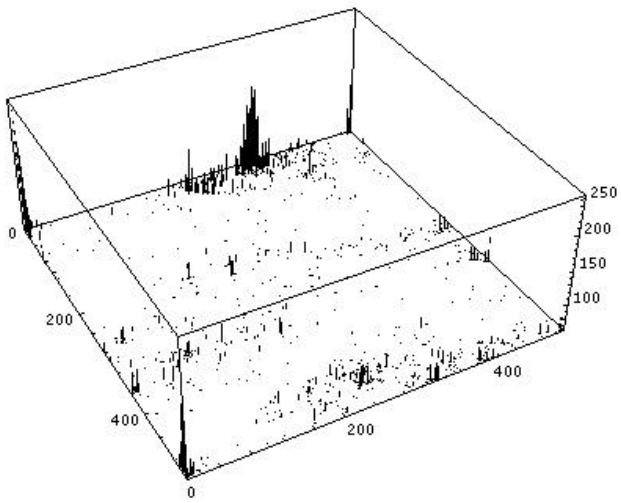


Figure 5.12: Phase correlation surface for figure 5.10 and 5.11

5.2.2 CEPSTRUM TECHNIQUE

Two cepstrum processes resulting in the power cepstra and the complex cepstra were introduced in chapter 3. As with the use of phase correlation, the IZ representation can take advantage of the use of complex numbers in cepstral processing.

5.2.2.1 USING THE IZ COLOUR REPRESENTATION

Of the two cepstrum methods, the complex cepstrum technique appears to be the more useful because not only is there a peak corresponding to the displacement of an object in comparison with an object in a reference image, but the peak is represented by a complex number whose argument can be used to find the colour of the located object. The phase information (and therefore the hue data) is kept when using the complex cepstrum and can therefore be used to discriminate between objects of different colours. However, this only applies to the situation where the background is zero and contains nothing. If this is not the case, the background obscures the phase data and the colour information is lost. Despite this, it is interesting to investigate the interaction between two objects. As stated in chapter 3, the complex cepstrum is generated by taking the logarithm of the FT of a single image and then calculating the FT of the result. The single input image typically contains a desired signal plus some echoes. In the case of object displacement, the single input image can be generated by adding a reference image to an object image. However, the ability of the complex cepstrum to generate peaks corresponding to echoes (indicating object displacement) *with* accurate phases (for colour) depends on the magnitude of the input objects of interest. It appears that the accuracy of the colour detection is dependent on the magnitude of the object image being half that of the reference image. So this method suffers from disadvantages in comparison with phase correlation for the detection of coloured objects.

5.2.3 DISCUSSION OF OBJECT LOCATION USING THE IZ REPRESENTATION

This chapter has shown how the representation of colour pixel values in complex form leads to generalisations of established monochrome frequency domain techniques in the

case of colour images. Using this representation with Fourier transforms and in particular with phase correlation has resulted in the great advantage of retaining the colour information about an object so that it is possible to discriminate between multiple identically shaped objects. The recognition is based on the saturation component while the hue can be used to further aid the recognition process. Apart from calculating the hue from the complex number representing the correlation peak, all this is done without the requirement for extra processing in comparison with recognising intensity only objects. Formerly, three Fourier transforms would have been required to convert the colour information into the frequency domain. However, if the object to be recognised has multicoloured features this could cause problems in the hue recognition, although not in the phase correlation. This is because if a multicoloured object is displaced in one image in comparison with a reference image, each different colour describing the object will be displaced by the same amount. This means that the hue description generated at the correlation peak cannot describe the hue of the object correctly (by giving a hue value in relation to the reference object) unless all the colours of the object are different by the same amount from those in the object in the reference image.

For the same amount of processing as for monochrome images, the use of this complex form has resulted in the evident and useful advantage that the colour of the object is detectable. In addition, this representation is suitable for use in overcoming problems due to variances in scale and rotation, as will be demonstrated in the next chapter.

The method of phase correlation using the IZ representation for object location has been adopted in further processing which is described in the next few chapters rather than the method involving the use of the complex cepstrum for reasons detailed below. No special pre-processing is required before phase correlation (apart from the usual windowing requirement). In comparison, the complex cepstrum requires that the reference and object images are added together and that the magnitude of the echoes is known since the resultant pulses and their positions are different depending on whether the magnitude of the input echo is larger or smaller than unity. The results of this research have been published by Thornton and Sangwine [op. cit. 1995]. This paper and others by Thornton and Sangwine which have been published are shown in appendix D.

The above techniques for object location are applicable when an object has been translated within an image, but problems arise when the object is rotated or scaled in comparison to a reference. This is the point when the log-polar transform is useful and its use and limitations are discussed in the next chapter.

5.3 COLOUR FILTERING

Although many methods have been developed in the spatial domain for filtering coloured images, its application in the spatial-frequency domain has been largely ignored. Certainly, the research that has been published in this area tends to split colour into its separate components such as R, G and B rather than treat it as a whole. The paper by McCabe et al. [op. cit.] which was briefly introduced in chapter 4 not only encodes colour data (slightly differently) in a complex number but also uses Fourier transforms of the complex images to develop what they call *spatial frequency gratings*. These gratings are generated using the magnitudes of the real and imaginary values at specified spatial frequency positions to allow chromatic filtering. This idea could be extended to the IZ representation so that instead of using the uv components from the Yuv space, the Z plane could be the basis for the colour filtering.

6. COLOUR OBJECT RECOGNITION USING THE IZ REPRESENTATION

The knowledge of the rotation, scaling and translation of an object in comparison with a reference object can be important in the recognition process. This knowledge can help to pinpoint the precise placement of objects; for instance, an automated system which is required to pick up objects requires quantitative knowledge as to where the object is, how far away it is and at what orientation it is. This information can be gained through comparison with a reference image.

The research which is described below uses Fourier transforms, a log-polar coordinate transformation and phase correlation together with the complex number representation for colour to determine these variances and recognise coloured objects.

The Fourier transform can be thought of as a translation invariant algorithm but it will not overcome problems associated with the scaling and rotation of an object in an image. One method to remove these variations is the use of the Fourier-Mellin transform (FMT) which was introduced in chapter 3. This procedure consists of a Fourier transform then a log-polar transform of the spectrum followed by another Fourier transform. The first FT removes translation variance since the spectrum of an object will be similar no matter where the object is located in the image. The log-polar transform converts rotation and scaling to translations which are then made invariant by the second Fourier transform. To achieve recognition, the result is then correlated with another image which has undergone the same process.

6.1 THE FOURIER-MELLIN TRANSFORM AND IZ COLOUR SPACE

Conventionally, the FMT has been used to generate a pattern to be matched against some reference pattern. However, the use of the FMT in a pattern matching system as shown in figure 6.1 suffers from the disadvantage that it does not make the best use of the information available since the result will only determine if there is a similar object in both images.

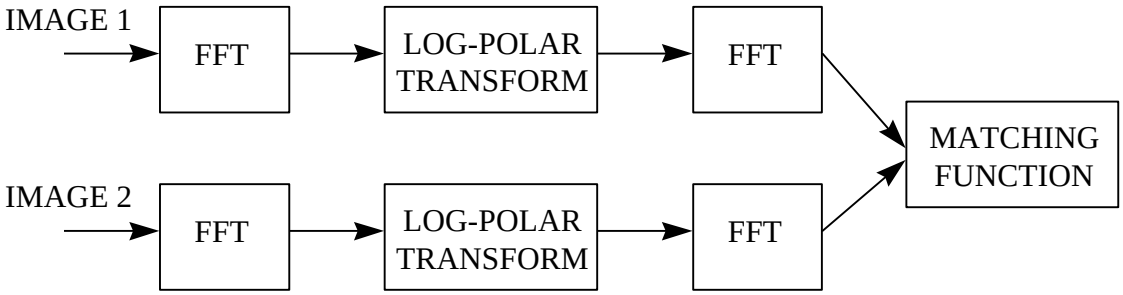


Figure 6.1: FMT pattern matching.

It would be more useful to be able to quantify the scaling, rotation and translation and indeed, determine the colour of the matching object. The process of phase correlation which together with the use of the IZ representation has been described in the previous chapter, has therefore been introduced and is the main novel feature of the work reported in this section. This processing is inspired by the Fourier-Mellin transform, but is able to quantify the rotation, scaling and translation and recognise the colours of the objects. The block diagram of the system is shown in figure 6.2 and will now be discussed.

6.1.1 Removing Geometric Variances

The diagram shown in figure 6.2 below illustrates the use of FMT descriptors. The amount of scaling and rotation between the two images can be determined by combining Fourier transforms, phase correlation, the IZ space and the log-polar coordinate transformation.

Chen et al. [op. cit.] used phase correlation and the Fourier-Mellin transform to determine the translation, rotation and scaling but no reference was made to discriminating between coloured objects. Interestingly, only half the spectrum was used as input to the log-polar mapping. This was because they were processing real-only input images and the resulting spectrum therefore results in symmetry so that only half the spectrum is required to be able to describe the whole spectrum. However, using only half the spectrum results in an inability to resolve the rotational difference between the corresponding objects in the two images after the Fourier-Mellin transform. The rotational angle, θ, which is calculated could actually be θ or $\theta + 180°$. Chen et al. [ibid] overcome this problem by re-scaling the object image by the (now) known scaling difference so that the two objects in each image are of the same size. The object image is then rotated by θ and then by $\theta + 180°$ to obtain two images. Phase correlation is calculated for each rotated image with the reference image and the rotation which results in the highest correlation is assumed to be the correct rotation. Having found both the rotation and scaling values it is now a simple matter to calculate the amount of translation of the object in one image in comparison with the other image.

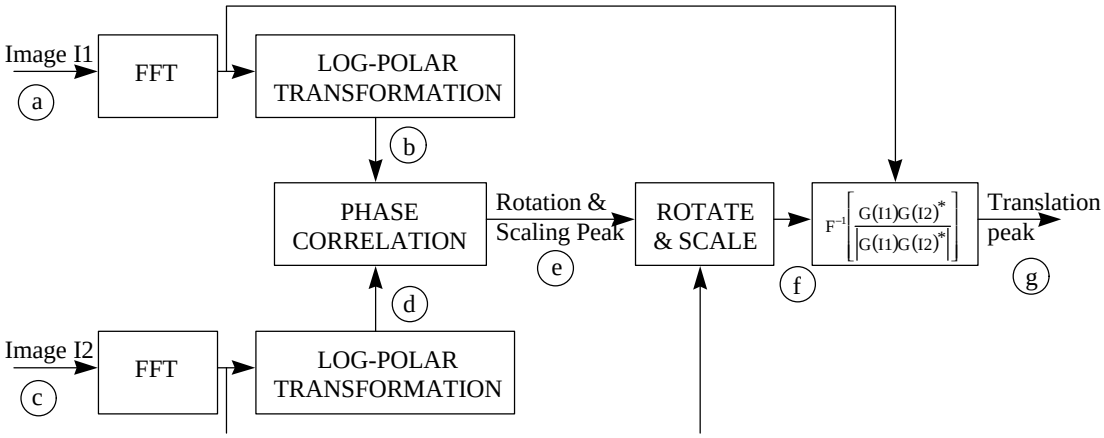


Figure 6.2: Translation, Scaling and Rotation Invariance System

The underlying idea of using phase correlation and the Fourier-Mellin transform to quantify the three variances of translation, rotation and scaling is of benefit in a correlation system but the way it is implemented by Chen et al. [ibid] is not as useful as the method proposed below, [Thornton and Sangwine 1996]. This is because it uses more phase correlation calculations than necessary to determine the rotational variance due to the use of half of the spectra and more importantly it does not investigate the use of colour in the recognition process. In addition, the oversampling and then filtering of the log-polar mapped pixels which is presented later in this chapter to reduce some of the errors associated with the non-linear log-polar sampling is not used.

The log-polar coordinate transformation has been described and is a method of sampling an image in such a way that if an object is rotated this causes the corresponding features in the log-polar transformed image to be shifted up or down in comparison with a reference image. In a similar manner, if an object is scaled this causes the log-polar transformed image to move right or left in comparison with a reference image. The amount of shift on either axis is indicative of the amount of scaling or rotation undergone by the object of interest. A constraint on the complex log-polar transform procedure is that the object of interest must be near the centre of the image. However, if a Fourier transform is calculated before the coordinate transformation this constraint is overcome, since the data is inherently centred in the spectrum. Thus, by applying a log-polar transformation to the spectrum, the requirement to locate the centre of the object of interest is avoided. (It should be remembered that the rotation and scaling of an object in an image causes rotation and inverse scaling of the components of the spectrum due to the object.)

The block diagram of figure 6.2 illustrates the processes involved; the letters within circles in the diagram indicate images which appear as outputs in figure 6.3. Some of the processing which is required to implement the block diagram can be performed before the capture of the object image. The reference image can be previously captured and stored, and its FT, Log-Polar transform and the FT required for phase correlation can be calculated. This will save processing time when object images are to be compared to a reference image.

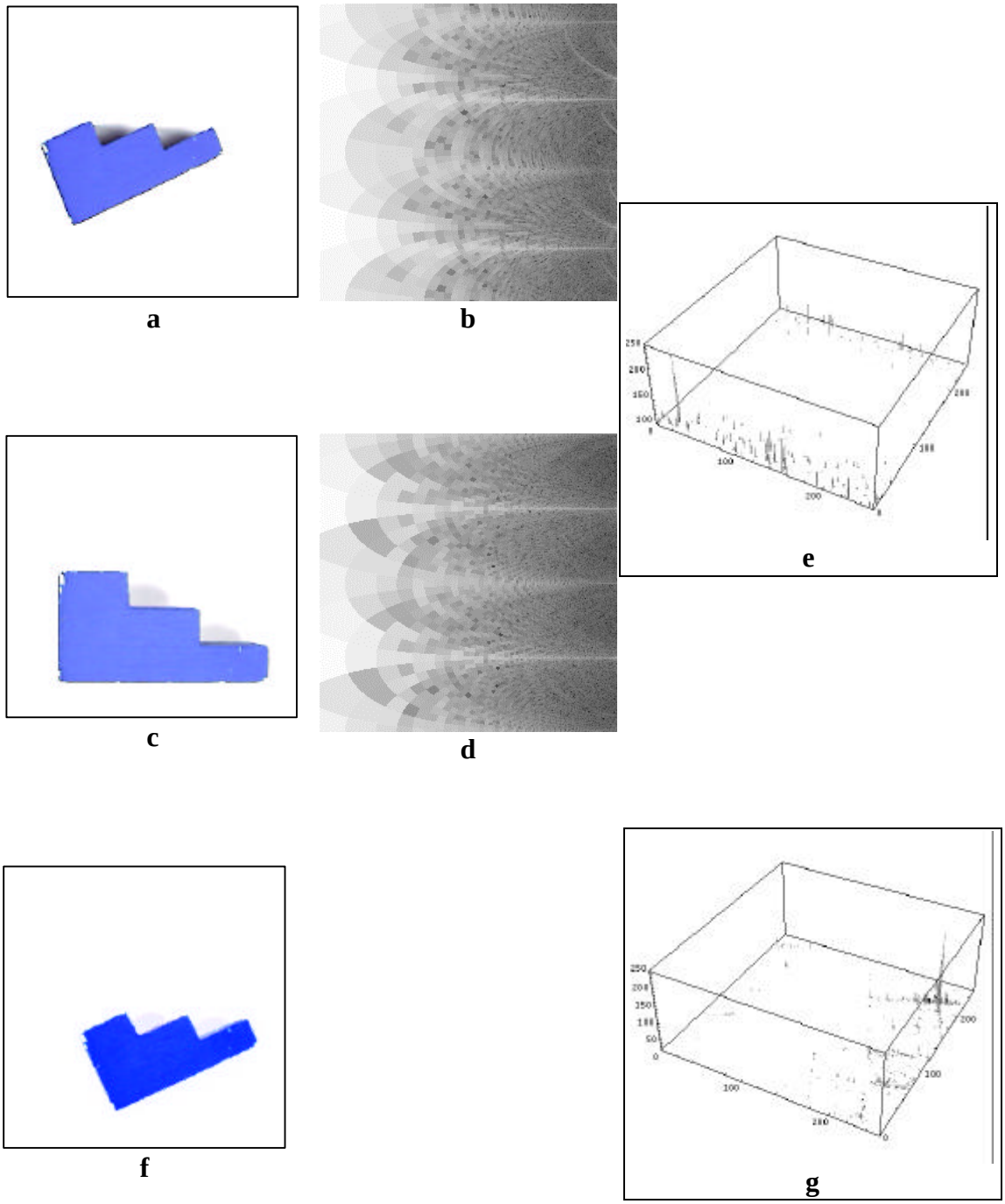


Figure 6.3: Outputs from the stages in block diagram shown in figure 6.2

As can be seen in figure 6.2, after each of the two images has undergone a Fourier transform and log-polar coordinate transformation, phase correlation of these two transformed images is calculated. Information about the position of the phase correlation peak can then be used to alter the object image so that scaling and rotation variance is removed. This is a more exact method than the technique proposed by Lee et al. [op. cit. 1988] to remove rotational differences which involves iteratively rotating the spectrum by small angles until the difference between the spectrum and the spectrum of the reference image is at its lowest value. Once the scaling and rotation differences have been removed, either from the input object image or its FT, the translation of the object can be found by phase correlation between the original reference image and the corrected object image and, as discussed in the previous chapter, the colour of the object can be found. The outputs of these processes are shown in figure 6.3, where each letter indicates at which point in figure 6.2 the output was obtained. Additional images relating to processes shown in figure 6.2 are shown in figures B.4a - B.4f in appendix B.

If the object image, called I2 in figure 6.2, is corrected at the stage before the first FT, it must be rotationally corrected by the amount specified to have the same orientation as the object in reference image I1. This is true if the correction is performed after the first FT. However, it should be noted that the situation for correcting scaling is different depending at what stage the image is corrected. For the case where the object is larger in image I2 than in image I1, if the image is to be corrected before the first FT it must be scaled by a value smaller than one; however, if the correction is calculated after the first FT it must be scaled by a value larger than one.

6.1.1.1 RELATIONSHIP OF PHASE CORRELATION PEAK TO VALUES OF ROTATION AND SCALING

To be able to calculate the amount of rotation and scaling of an object in comparison with a reference, the relationship between the position of the phase correlation peak and the corresponding rotation and scaling value must be known. These equations can be derived using knowledge of the calculation of the sampling points and the equation relating the phase correlation peak and the position of the input objects.

The case of rotation will be dealt with first; if an object is unrotated and unscaled then a clockwise rotation, θ, of that object will result in the LPT signature becoming shifted down by 255×θ/360 mod 360. If phase correlation is then calculated on the two LPT images then the peak will appear shifted down by that much. Correspondingly, if the peak position is known the rotation is known since θ equals $360 \times y_{peak}/255$.

If identical objects in each image are scaled but not rotated then recalling equation (3.5a) gives equation (6.1):

$$X = X_C + 128^{\frac{Xout}{255}} \qquad (6.1)$$

where X = Sampling position = Sampling centre + radius from centre of sampling

X_C = Sampling centre

X_{OUT} = Position in output of LPT array

Therefore the position of an object in the spatial frequency image from the centre of sampling is obtained by rearranging equation (6.1) to give equation (6.2).

$$X_{OUT} = \frac{255\log(X - X_C)}{\log 128} = 255\log_{128}(X - X_C) \qquad (6.2)$$

where $X - X_C$ = radius from sampling centre

From equation (3.2), the horizontal peak position, $X_{peak} = r_{reference} - r_{object} \bmod 256$. Therefore, if the size of the reference object is smaller than the size of the object in the second image the equation requires the addition of 256. However, if the reference object is smaller in the second image there is no need to add 256. This results in two equations for relating the horizontal position of the phase correlation peak to the scaling. Equation (6.3) is to be used if $N/2 \leq X_{peak} < 256$. However, if $0 \leq X_{peak} < N/2$ then equation (6.4) should be used since N is not required to be added.

$$N - X_{peak} = R_O - R_R$$

$$N - X_{peak} = 255\log_{N/2}(R_O) - 255\log_{N/2}(R_R)$$

$$\frac{N - X_{peak}}{255} = \log_{\frac{N}{2}}\left(\frac{R_O}{R_R}\right)$$

$$\frac{R_O}{R_R} = \left(\frac{N}{2}\right)^{\left(\frac{N - Xpeak}{255}\right)} \qquad N/2 \le X_{peak} < 256 \qquad (6.3)$$

$$\frac{R_O}{R_R} = \left(\frac{N}{2}\right)^{\left(\frac{-Xpeak}{255}\right)} \qquad 0 \le X_{peak} < N/2 \qquad (6.4)$$

where R_O = point on unrotated object

R_R = corresponding point on unrotated object

N = size of image

X_{peak} = horizontal position of phase correlation peak

It should be noted that R_O and R_R refer to points in the image which is the input to the LPT and therefore by referring to the system block diagram of figure 6.2 it can be seen that R_O and R_R are points from the output of the first FT. This result means that the scaling calculated using equations (6.3) or (6.4) are applicable to scaling in the spatial frequency domain. To obtain the scaling in the spatial domain the inverse of the result from equations (6.3) or (6.4) should be calculated such that the results are equal to the value of the (size of the reference object / size of the object in second image).

6.1.1.2 WINDOWING

The issue of windowing the LPT before calculating the second Fourier transform is interesting because, unusually, it is not necessary to window in both dimensions. This should be apparent if it is known how the output of the LPT is formed (as has been discussed). Assuming the vertical axis of the LPT output represents the angular sampling of the FT and the horizontal axis of the LPT output represents radial sampling of the FT it can be understood that the vertical axis should not require windowing as it is continuous; assuming integer angular increases, the angles of 359° and 0° are neighbours but the samples taken at an angle of 359° appear horizontally at the bottom of the sampled image and the samples taken at an angle of 0° appear horizontally at the top of the sampled image.

Unfortunately, the horizontal axis does not have that advantage and therefore some method must be used to remove the discontinuities at the vertical edges. If the input to

the LPT is the output of the first Fourier transform, the output of the LPT will contain the DC and low spatial frequency (usually high intensity) terms to the left edge whilst the right edge will contain the high spatial frequency (usually low intensity) terms. This output can either be windowed horizontally or another method that can be used is to differentiate the output. This will retain the edges of the pattern in the LPT but will remove the flat intensities over the image so the brightness caused by the low spatial frequency terms on the left side of the image is removed. One advantage of this method is that data about the image at low and high spatial frequencies is kept thus improving the recognition procedure.

6.1.1.3 RESULTS

Figures 6.3a and 6.3c show example inputs to the system. Each spatial image is Fourier transformed and a log-polar coordinate transform applied, the outputs of which are shown in figures 6.3b and 6.3d. These outputs are then phase correlated so that the rotation and scaling difference between one image and the next can be found from the correlation peak (figure 6.3e). In this case the peak occurs at (14,18) which corresponds to a rotation of 25° and a scale change of about 0.77:

Rotation: $18 \times 360/255 = 25°$

Scale: $128^{(-14/255)} = 0.77$

In the above situation the object appears 0.77 times smaller in the spatial frequency image but 1/0.77 times larger in the spatial image, as can be seen from figure 6.3c. Using this information, one of the spatial images is corrected for rotation and scaling (figure 6.3f) and the result of this is phase correlated with the untouched spatial input. The resultant correlation peak, shown in figure 6.3g, indicates the translation of the reference object with the object in the second spatial image. In addition, the colour of the second object is found by calculating the argument of the complex correlation peak.

This procedure has been shown to successfully obtain the translation, rotation and scaling data necessary to provide information to practical object recognition systems. If more than one similar object but of different colour were in the second image the system is able to discern them; this has been tested but is not shown here.

6.2 PROBLEMS WITH THE FOURIER-MELLIN TRANSFORM

The Fourier-Mellin transform is a method for making images rotation, scale and translation invariant. It can be implemented in either optical or digital form. However, if the transform, or the use of its descriptors, is to be performed digitally then there are improvements which can be made to the processing to enhance the result. This section suggests that the novel use of a spatially variant filter to modify the output of the Fourier transform improves the output of the Fourier-Mellin transform, [Thornton and Sangwine 1997]. The motive for filtering the input to the LPT is to remove the possibility of aliasing caused by the LPT sampling of an image and consequently improve the peak in the subsequent phase correlation surface caused by the difference in rotation and scaling between the two input images.

6.2.1 LOG-POLAR TRANSFORM SAMPLING

The LPT samples the output of the FT (the spectrum of the original image) by mapping pixel coordinates from cartesian to polar form. The mathematics behind complex logarithmic mapping is shown in equation (6.5), although the base of the logarithm depends on what size of image is to be sampled.

If $Z = x + jy$

then $Z = r \times \exp(j\theta)$

Let $W = \log(Z)$

$$\therefore W = \log(r) + j\theta \qquad (6.5)$$

where Z: Input image

x: horizontal coordinate for the pixel

y: vertical coordinate for the pixel

W: Output image

r: $|Z|$

θ: $\arg(Z)$

Sampling is achieved by mapping pixels, for each pixel in the output image, from the input to the output image. The centre of the LPT sampling is the zero frequency term in the FT. The output of the FT is sampled with exponentially increasing radius and linearly increasing angle. The sample points are shown in white in figure 6.4.

When using a normal LPT, at each radius a circle is sampled the same number of times so that there are N sample points equally spaced apart on the circle; this means that as the radius is increased more pixels on the circle are not sampled and therefore ignored. So for an image of size 256 x 256, pixels start to be missed at a radius of 41 pixels. The LPT therefore samples the FT non-uniformly and this can produce aliasing errors due to undersampling at high radius values. Filtering is therefore required due to the increasing sparsity of the sampling points. If a filter is introduced between the FT and the LPT this aliasing effect is removed. At low radii it is not necessary to filter the image because the pixels are oversampled during the LPT, but as the radius increases, the distance between successive samples on a circumference increases and the pixels must be filtered to prevent aliasing. The filtering of the FT therefore should be spatially variable.
A spatially variable filter at the input to the LPT, which filters *circumferentially*, has been introduced. Due to the decreasing amount of oversampling as the radius increases, the effect of the filter is variable and has more effect at greater radii. Chen et al. [ibid] did not use this in their recognition system and this system is therefore more robust in determining the rotation and scaling and consequently the translation values.

6.2.2 Implementation of Spatially Variable Filtering

The basic idea behind the implementation of this process is that the FT is oversampled by the LPT which is then filtered and resampled to obtain the final image. Filtering is achieved by sampling the FT in the same way as the LPT but at an increased resolution such that at the outer radius, no pixels on the circumference are missed during sampling. The block diagram explaining this process is illustrated in the lower half of figure 6.5.

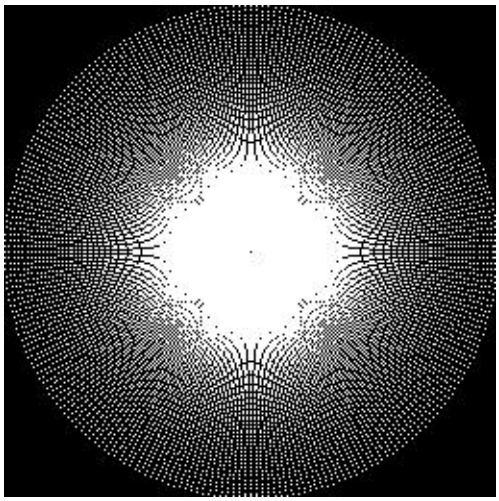


Figure 6.4: Sample points of the LPT

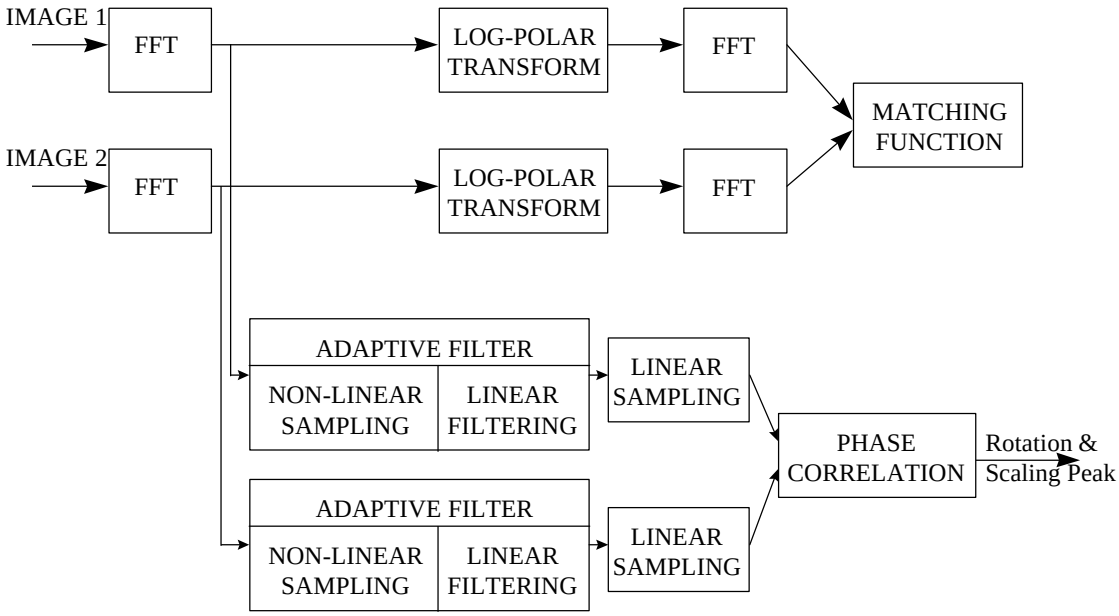


Figure 6.5: Adaptive filtering for Log-Polar Transformation

The top half shows the conventional FMT with a matching function at its output as a comparison. The figure can be compared with the system shown in figure 6.2 and it can be seen that the proposed spatially variable filtering affects one stage of the object recognition system shown in figure 6.2.

The number of samples is the same for each radius so that at lower radii, the circumference is oversampled. This is shown in figure 6.6 where the new sample points are shown in white.

The value of each pixel which is to be put in an array for filtering is found using bilinear interpolation [Pratt op. cit.]. The sampled pixels are put into a rectangular array, so that the radius represents the horizontal direction and the angle (circumferential samples) represents the vertical direction. For a 256*256 image, the greatest sampling circumference contains almost 805 pixels and the array therefore has a size of 256 by 805 pixels.
This array can then be filtered by convolving using a 1D mask so that each output pixel is an average of its vertical neighbours (i.e. its neighbours along the circumference before sampling). Figure 6.7a represents the sampling grid for the LPT on the input image and figure 6.7b shows the mapped samples from the input where at low radii the pixels have been oversampled along the circumference and at the largest radii each pixel on the circumference is sampled once only. This oversampled array is then convolved with a suitable 1D mask and the result is then downsampled to the correct array size as can be seen in figure 6.7c. Those pixels nearer to the left side of the array which were nearer to the zero frequency term and therefore have smaller radii are less affected by the filtering. This is due to the oversampling at smaller radii so that the value of each pixel after convolution is unchanged. In contrast, the outputs of those pixels which were not oversampled will depend upon the neighbouring pixels. The final output of the filtered LPT (size 256*256) is obtained by sampling every π lines, using bilinear interpolation, in the filtered array.

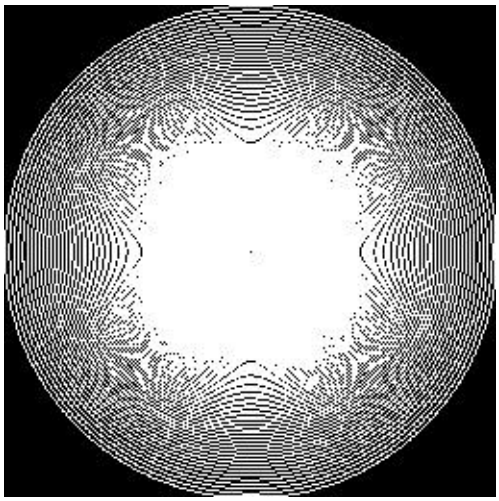


Figure 6.6: New sample points of the LPT

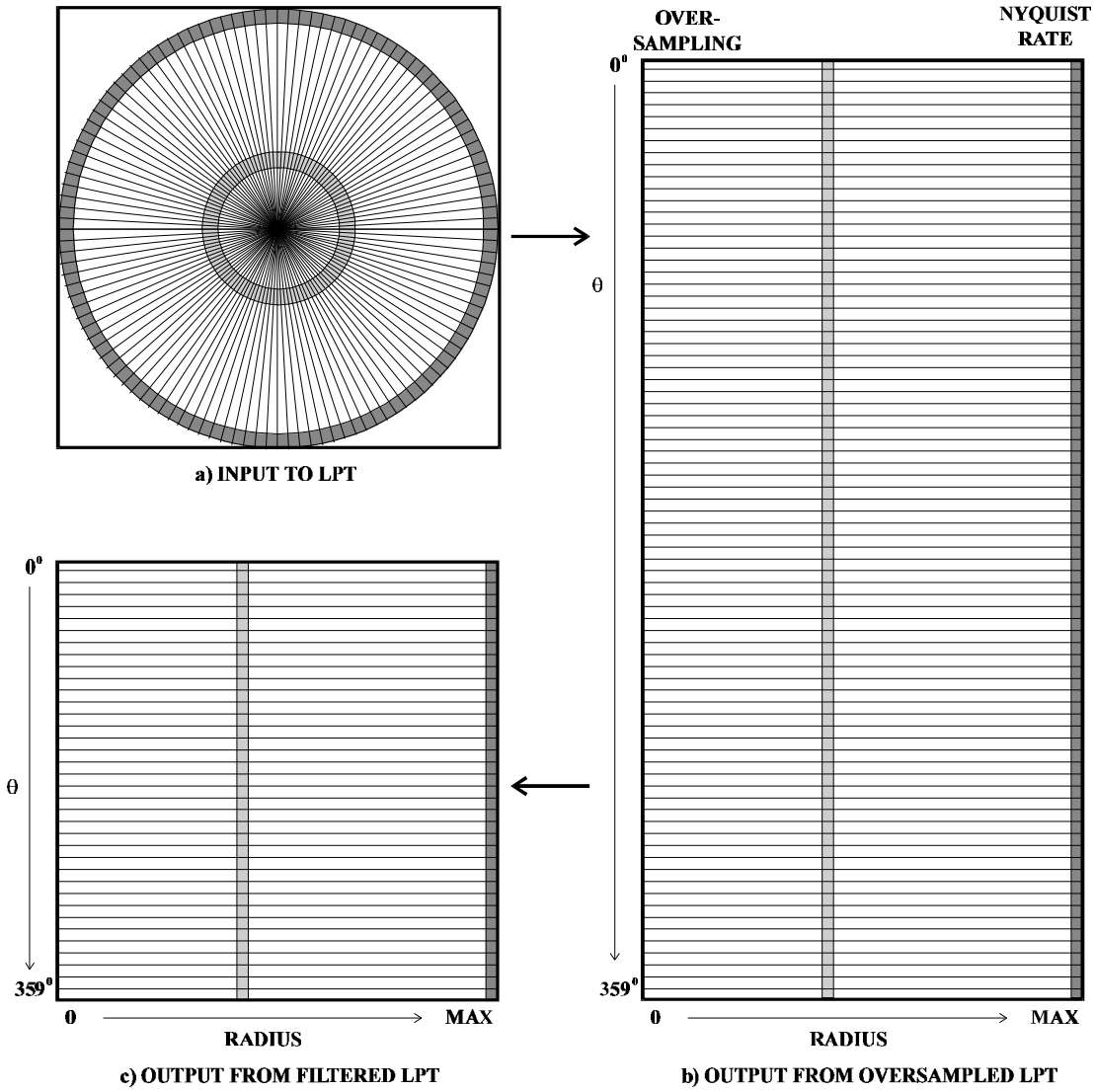


Figure 6.7: The oversampling process for filtering the input to the LPT

For a 256*256 image, the LPT can be obtained by 256*256 look-up-table (LUT) calculations, where the LUT contains the coordinate locations to be sampled. The optimised computation for the filtering and LPT process requires 256*805 LUT calculations and 256*256 [3*1] convolutions.

The method described above deals with aliasing caused by the distance between consecutive samples of a circumference. The other possible cause of aliasing is from the distance between radial sample points. A similar method as described above has also been used for sampling in the radial direction at an increased resolution so that filtering takes place along the radial direction.

Having explained the methodology behind the spatially adaptive filter, the next section contains some results to demonstrate the effectiveness of the filtering strategy.

6.2.3 RESULTS

The spatially adaptive filtering along the circumference of each sampling radius improved the peak in the phase correlation surface and also removed the aliasing effect. It may not be necessary to filter along both the circumferential and radial directions as this requirement depends upon the input FT image. However, filtering in both directions removes the possibility of aliasing along the circumferential and radial directions caused by the LPT undersampling an image. Testing has shown improvements of up to 13 dB in desired phase correlation peaks. Figures 6.8 and 6.9 have false origin offsets of 40 in the magnitude of the data so that the result is easier to interpret. Figure 6.8 shows the result of phase correlating the LPT of the FT of a simple image with the LPT of the FT of a rotated copy of that image; figure 6.9 illustrates the effect of adaptive filtering of the LPT on the phase correlation surface. Generated peaks are caused by rotation and scaling. It can clearly be seen that the peaks resulting from phase correlating the filtered LPTs are more easily identifiable and contain much less spurious data. The cause of the four peaks (instead of one) is because the input image which was processed was symmetrical about four 90 degree axes.

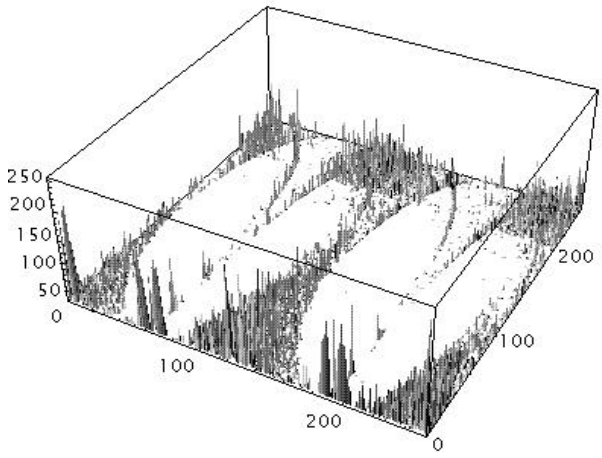


Figure 6.8: Phase correlation surface, no filtering

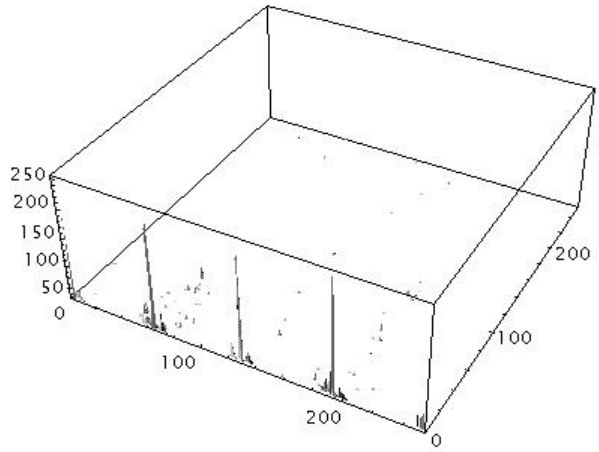


Figure 6.9: Phase correlation surface, with filtering

6.2.3.1 FILTERING APPROACHES

The result shown in figure 6.9 was generated by applying an adaptive filter to the LPT samples in the circumferential direction only so that a filter mask was applied in the vertical dimension in figure 6.7b. This filtering would be applicable, for instance, in a situation where an object has spatial frequencies such that it produces lines radiating from the centre (DC value) of the spatial-frequency image. At higher spatial-frequencies, the sampling without filtering could miss this data.

The application of the filter is not limited to filtering along the circumference and, depending on the spatial-frequency content of the image to be log-polar sampled, it may be desirable to also filter in the radial direction. However, under normal log-polar sampling, fewer pixels are ignored in a radial direction than in a circumferential direction.

As stated above, where necessary, bilinear interpolation is used during the pixel sampling in all the results. As would be expected, if interpolation is not used when mapping the pixels the improvement in the required peak is more apparent.

A number of test images, figures B.5a - B.5g were photographed and can be seen in appendix B. Phase correlation was performed on various pairs of images to generate a peak relating to the amount of rotation and scaling, as has been discussed previously. The processing was performed with adaptive filtering of both the circumferential and radial axes and without filtering. The results are shown in figures B.6a - B.6t in appendix B and in quantitative form in table 6.1. Additional results are shown in table 6.2 to indicate the use of filtering in the circumferential and/or radial directions. It can be seen that in most cases, the filtering improved the signal-to-noise ratio of the desired peak.

6.2.4 DISCUSSION

The research presented in this chapter has built upon the research of the previous two chapters and enables the quantification of rotation, scaling and translation between a reference object and another arbitrary image containing the object. The results show that

	Interpolation			
	No Filtering		Adaptive Filtering	
Test Case	Peak Value	SNR	Peak Value	SNR
A - B	255	21.9	255	22.1
A - C	255	19.3	255	18.7
A - D	221	16.4	210	17.4
A - E	255	28.8	255	29.2
A - F	255	22.8	255	23.2
A - G	250	16.3	246	16.6
B - D	255	34.0	255	34.3
B - E	237	18.2	255	19.7
B - G	233	15.0	234	14.9
C - G	237	15.1	246	15.2

Table 6.1: Adaptive filtering compared with no filtering

	No Interpolation			
Test Case	Peak	Peak Sequence	Filter Type	SNR
A - B	255	255,231,205,192,159,159,159,157,156,145	NONE	18.8
A - B	255	255,186,185,175,174,119,117,116,115,113	C	21.2
A - C	255	255,217,186,180,180,176,175,169,167,158	NONE	17.8
A - C	255	255,198,197,191,190,169,168,166,159,157	R	18.0
A - D	184	255,212,205,204,199,184,178,177,175,169	NONE	14.0
A - D	149	255,168,149,128,125,120,120,120,120,119	B	12.9
A - E	255	255,135,119,99,98,91,83,80,80,72	NONE	25.3
A - E	255	255,118,108,97,90,89,82,66,65,59	C	26.9
A - F	255	255,170,153,117,114,104,97,93,81,80	NONE	19.1
A - F	183	255,183,178,147,130,126,121,86,79,77	B	21.4
A - G	116	255,240,231,167,141,140,136,133,132,130	NONE	11.9
A - G	182	255,244,236,182,152,149,145,141,140,139	R	15.3
B - D	255	255,128,30,26,24,23,22,22,21,21	NONE	34.9
B - D	255	255,122,40,36,36,30,27,20,20,19	B	35.5
B - E	255	255,219,200,191,187,170,168,168,166,165	NONE	16.9
B - E	255	255,183,177,152,152,150,147,144,143,141	B	18.9
B - G	200	255,208,205,205,205,204,203,203,201,200	NONE	12.9
B - G	230	255,230,227,194,191,190,190,189,188,186	C	14.8
C - G	145	255,202,196,172,171,165,162,161,161,160	NONE	12.1
C - G	219	255,229,225,219,207,203,202,197,194,193	C	14.2

C - Circumferential filtering
R - Radial filtering
B - Circumferential and radial filtering

Table 6.2: Best filtering result compared with no filtering

it is possible to do this without having to perform iterative calculations to determine these values. It is also possible to determine if the object is of the desired colour due to the colour representation which is used.

This chapter has also explained the need to filter an image by varying amounts, which depends on the spatial position of the filter on the image, before log-polar transformation. The implementation of the filter ensures that pixels are filtered along each circumference which is to be sampled, and that the amount of filtering depends upon the radius of the circumference from the centre of sampling. The implementation of the filter requires little extra complexity in comparison with the log-polar transform, while being able to achieve circumferential filtering of the FT by mapping pixel coordinates and convolving a 1D mask with a square image. The results which have been obtained show that filtering in this way improves the desired phase correlation peak.

7. THE USE OF QUATERNIONS IN CORRELATION

The previous chapters have described the use of complex numbers rather than real numbers in recognising coloured objects using phase correlation. However, although the results are interesting and significant, this use of complex numbers can be thought of as one step along a path which could lead to the development of the use of a set of numbers so that a single FT of multiple signals of an image can be calculated, the image processed and then inverse Fourier transformed. This is where the set of numbers known as *quaternions* is of interest. It should be of obvious benefit if colour processing operations are able to be performed by treating colour information (comprising three distinct values as opposed to the two using the Z value from the IZ representation and ignoring the intensity) as a single entity for each pixel in an image rather than repeatedly using a processing algorithm to operate on the separate colour components of each pixel.

7.1 QUATERNION NUMBERS

Quaternion numbers were first introduced by Hamilton [1866] and have since been the preserve mainly of mathematicians. Their Cartesian form is (a + ib + jc + kd) where a, b, c and d are real numbers and i, j and k are complex operators. Multiplication of quaternion numbers is not commutative and the properties of the three operators are:

$i^2 = j^2 = k^2 = ijk = -1$

$ij = k \quad jk = i \quad ki = j$

$ji = -k \quad kj = -i \quad ik = -j$

In addition, the quaternion conjugate of (a + ib + jc + kd) is (a - ib - jc - kd) and its modulus is $\sqrt{(a^2 + b^2 + c^2 + d^2)}$.

As can be seen from the format of the quaternion, it is possible to represent completely a single colour pixel using a quaternion number. With that in mind, together with knowledge of the research presented in previous chapters, work was undertaken by Sangwine into developing a discrete quaternion FT.

7.2 QUATERNION FOURIER TRANSFORMS

Ell [1993], while investigating the stability of two-dimensional linear systems, developed Hamilton's work to show that quaternion Fourier transforms are possible. Equations 7.1a and 7.1b show the forward and inverse QFT.

$$H[jw,kn] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-jwt} h(t,t) e^{-kut}\, dtdt \qquad (7.1a)$$

$$h(t,t) = \frac{1}{4p^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{jwt} H[jw,kn] e^{kut}\, dudw \qquad (7.1b)$$

Sangwine [1996, 1997] introduced a 2D discrete quaternion Fourier transform (QDFT) based on the work by Ell which, as with the 2D DFT, is separable and thus can be calculated using row-column separation; it is also possible to implement a fast algorithm (QFFT). The QDFT and inverse QDFT are shown in equations 7.2a and 7.2b.

$$F[u,v] = \frac{1}{\sqrt{MN}}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} e^{\frac{-j2pxu}{M}} f(x,y) e^{\frac{-k2pyv}{N}} \qquad (7.2a)$$

$$f(x,y) = \frac{1}{\sqrt{MN}}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} e^{\frac{j2pxu}{M}} F[u,v] e^{\frac{k2pyv}{N}} \qquad (7.2b)$$

As can be seen, this is very similar to the 2D DFT shown in equation (2.2). However, the basis functions contain orthogonal imaginary components which leads to independence of the variables in the frequency domain, unlike the result from the 2D complex FT.

7.2.1 COLOUR DQFTS

Sangwine introduced the idea of calculating the QDFT of colour images by placing the RGB components into the three imaginary components of a quaternion number and setting the real part to zero. This was then processed by applying a filter in the form of a frequency domain mask to the QDFT values and inverse transforming. The resulting RGB values were taken from the three imaginary components into which they had originally been placed and the result produced a filtered version of the original data as expected. So this experiment showed that it is possible to convolve coloured images represented using quaternions.

7.2.2 QUATERNION PHASE CORRELATION

Having seen that colour can be successfully coded in a quaternion number so that Fourier domain processing can be achieved it was decided to try the phase correlation of two colour images which contained a similar object in each image. This was done for each pixel by putting the R, G and B components in the i, j and k components of the quaternion number. Each quaternion image then underwent a QDFT and one was conjugated. They were then multiplied together, divided by their magnitudes and inverse transformed using the same method used in the complex case discussed in chapter 5, except that the rules for quaternion mathematics were observed. The result produced a peak whose position corresponds to the amount of translation in one image of an object in relation to its position in the second image but there were also other peaks generated. The testing of phase correlation has currently only been undertaken using simple images, so it is possible to see that the other unwanted peaks are related to the desired peak. Unfortunately, the colour information is not able to be retained so that although this process uses the RGB colour data to produce a correct peak it did not result in the RGB values stored in the imaginary components having the right value at the position of the peak. Currently, it does not seem that the useful property of correlation is applicable using quaternion Fourier transforms; however, given the four correlation peaks generated with a definite symmetry, it may be possible to overcome this problem[7.1] but this will take further research and is one of the subjects covered in the last chapter.
The quaternion representation and the use of phase correlation with the IZ representation are covered in Sangwine and Horne [1998c].

The next chapter discusses the research which has been presented in this and the previous chapters.

.

[7.1] Sangwine, in a personal communication, says that he and Ell now know the definition for the QDFT is incorrect.

8. DISCUSSION & FURTHER RESEARCH

The research undertaken by the author and which is presented in this thesis builds upon previously published work by others in the fields of image processing, colour and vision. The aim of the research was to enable the recognition of coloured objects and the discrimination between similarly shaped objects containing dissimilar colours. In particular, this was to be accomplished using frequency domain processing.

8.1 DISCUSSION

The state of the research area before the start of this research was that phase correlation had been used to determine the location of monochromatic signals. This had then been used in conjunction with the Fourier-Mellin transform to allow the quantification of the amount of translation, scaling and rotation an object had undergone in an image in comparison with a reference image containing the object. However, as a result of the new research presented in this thesis, a method has been developed for encoding colour in such a way that even after Fourier transforming an encoded image, the hue information from the object in the reference image and the image to be matched is able to be retained during phase correlation. In addition, it has been shown that the phase correlation surface for the scaling and rotation quantification, generated using phase correlation and the Fourier-Mellin transform can be much improved through the use of a spatially variable filter applied during the log-polar sampling of the Fourier transform spectrum.

This colour encoding which has been developed during this research has been called the IZ representation and it is based on the HSI method for describing colour. Its relationship with the HSI colour space has been demonstrated mathematically and methods for overcoming clipping problems when converting to and from the RGB space have been proposed and implemented successfully.

When phase correlation is undertaken using this encoded colour image, it uses the saturation component to recognise the shape of the object and the hue is used to

determine the colour of the object. In this way, it is possible to accurately determine the location and colour of an object. However, this colour discrimination is possible only when the reference object and the object to be matched contain a single hue; this is due to the nature of phase correlation. If an object contains only one colour and it is translated and the reference image contains only the object, the position of the phase correlation peak is due only to the translation of that object and its hue is due only to the object. However, if the reference object or the object in the image to be matched contain multiple hues, the hue resulting from the phase correlation peak is due to a combination of all the hues in the reference object and all the hues in the matching object. For this reason, colour discrimination fails if the object contains more than one hue.

The processes of obtaining the Fourier transform spectrum, log-polar sampling the spectrum and then calculating another Fourier transform spectrum which are involved in the Fourier-Mellin transform were altered by Chen et al. [op. cit.] so that phase correlation is included in the recognition process. The problem with the Fourier-Mellin technique on its own is that information on the quantification of the translation, scaling and rotation of the object is discarded. The involvement of phase correlation allows this information to be calculated. However, Chen et al. [ibid] did not use colour in their process and only operated on half the spectrum which required extra calculations later to determine the rotational angle. The processing developed here uses the Z values of the IZ representation and also the whole spectrum to reduce confusion over the rotational value at later stages of the algorithm. In addition, one of the processes involved, log-polar sampling of the Fourier spectrum of the input image, has been altered to include filtering. The amount of filtering is dependent upon the distance between pixel samples and this process has been shown to be of value in reducing the noise in the phase correlation surface, which is computed after the log-polar sampling, and also in increasing the amplitude of the phase correlation peak. This filtering can be applied to take account of the potential aliasing in both scaling and rotation data caused by the log-polar sampling. Unfortunately, this filtering does require more computation to be performed, but much of the sampling can be done with the use of a look-up table to avoid unnecessary calculations.

In summary, this research has proposed a new colour representation which has enabled the location of coloured objects using Fourier techniques, and it has also made possible more accurate determination of the amount of rotation and scaling of an object in one image in comparison with an object in a reference image by the application of a spatially variable filter.

This research has developed a method for colour object recognition and improved on an existing monochromatic recognition method. However, the requirement of other methods to encode colour has led naturally from the IZ colour representation to quaternions. Although phase correlation was unsuccessful using quaternions since multiple peaks were generated and the colour information was not kept, there is enough promise in the use of quaternions with other processing techniques to warrant much more investigation.

The next section discusses the implications for further research resulting from the research undertaken in this thesis.

8.2 FURTHER WORK

If the chromatic information is represented using the complex number, Z, and phase correlation is calculated, the hue information on the object is only able to be determined if the peak representing the amount of translation of the object between the two images is generated from objects containing a single hue. This is unlikely to be the case in every practical situation so it would be useful for further investigation to be undertaken to determine the accuracy of the hue values generated by the phase correlation peak when the object to be matched contains varying amounts of different hues. The research has not involved the use of wavelets which are a relatively new area of interest. It would be interesting to use wavelets in object recognition and to determine the applicability of the IZ representation to the colour processing of wavelets.

Although more extensive testing of the spatially adaptive filtering could be undertaken to determine accurately when to apply circumferential and/or radial filtering and so reduce unnecessary computations, the main drive for further work must lie with the coding of colour pixel data so that a colour image can be processed as a whole rather than as separate components. The research presented here has demonstrated a step in this direction so that the chromatic information is able to be encoded and also processed successfully using Fourier techniques. However, the use of the IZ representation still does not allow the completely holistic processing of colour images. It is with the use of the quaternion representation that the possibility of encoding a colour pixel (containing three components) as a single quaternion number has been shown to be feasible.

Further research is needed to determine if it is possible to correlate quaternion images either by using a different formulation of the quaternion Fourier transform or by altering the quaternion conjugation during the correlation calculation. Currently, quaternion convolution is able to be calculated successfully but it would be enormously useful if the correlation problem could be overcome.

As stated in a previous chapter, McCabe et al. [op. cit.] built upon research on the IZ representation to use the Yuv colour space to generate complex numbers which are then filtered in a spatio-chromatic Fourier transform domain to extract desired features. This is very promising work and an area which can be applied to the quaternion representation of colour images. Again, this will require much study and understanding of the issues underlying the processing.

As can be appreciated, a large area of research has been opened up due to the use of the IZ representation and the object recognition process which should result in greater understanding of the use of colour in recognition algorithms involving Fourier techniques. It should now be apparent that the separate processing of the three components in a colour image is not the best option when it is possible to code the components of each colour pixel in a single quaternion number and process the quaternion values to obtain the desired results.

REFERENCES

[Altmann and Reitböck 1984] J Altmann and H J P Reitböck, "A Fast Correlation Method for Scale- and Translation-Invariant Pattern Recognition", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.PAMI-6(1), pp.46-57, 1984.

[Ashbrook 1993] A P Ashbrook, "2-Dimensional Object Recognition (Literature Survey)", *ESG Memorandum 93/8, Dept. of Electronic and Electrical Engineering, University of Sheffield*, 1993.

[Berry 1987] D T Berry, "Colour recognition using spectral signatures", *Pattern Recognition Letters*, Vol.6, pp.69-75, 1987.

[Bogert et al. 1963] B P Bogert, M J Healy and J W Tukey, "The quefrequency alanysis of time series for echoes: cepstrum, pseudo-autocovariance, cross-cepstrum, and saphe cracking", *Proc. Symp. Time Series Analysis*, M.Rosenblatt, Ed. New York: Wiley, Chap.15, pp.209-243, 1963.

[Bracewell 1983] R M Bracewell, "The Discrete Hartley Transform", *Journal of the Optical Society of America*, Vol.73(12), pp.1832-1835, 1983.

[Bracewell 1986] R M Bracewell, "*The Hartley Transform*", Oxford University Press, Oxford, England, 1986.

[Brigham 1988] E O Brigham, "*The Fast Fourier Transform and its Applications*", Prentice-Hall, Englewood Cliffs, New Jersey, USA, p.45, 1988.

[Bulthoff and Edelman 1992] H H Bulthoff and S Edelman, "Psychophysical Support for a Two-dimensional View Interpolation Theory of Object Recognition", *Proc. Natl. Acad. Sci. USA.*, Vol.89, pp.60-64, 1992.

[Casasent and Psaltis 1976] D Casasent and D Psaltis, "Position, rotation and scale invariant optical correlation", *Applied Optics*, Vol.15(7), pp.1795-1799, 1976.

[Castleman 1996] K R Castleman, "*Digital Image Processing*", Prentice-Hall, Englewood Cliffs, New Jersey, USA, p.25, 1996.

[Celenk 1991] M Celenk, "Colour image segmentation by clustering", *IEE Proceedings*, Vol.138(5), pp.368-376, 1991.

[Childers et al. 1977] D G Childers, D P Skinner and R C Kemerait, "The Cepstrum: A Guide to Processing", *IEEE Proceedings*, Vol.65(10), pp.1428-1443, 1977.

[Chen et al. 1994] Q-S Chen, M Defrise and F Deconinck, "Symmetric Phase-only Matched Filtering of Fourier-Mellin Transforms for Image Registration and Recognition", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.PAMI-16(12), pp.1156-1168, 1994.

[Connolly and Leung 1995] C Connolly and T W W Leung, "Industrial Colour Inspection by Video Camera", *IEE 5th Int. Conf. on Image Processing and its Applications*, Heriot-Watt University, Edinburgh, UK, July 4-6 1995, Vol.410, pp.672-676, Institution of Electrical Engineers, London, 1995.

[Cooley and Tukey 1965] J W Cooley and J W Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series", *Math. Comp.*, Vol.19(90), pp.297-301, 1965.

[Cumani 1991] A Cumani, "Edge detection in Multispectral Images", *CVGIP: Graphical Models and Image Processing*, Vol.53(1), pp.40-51, 1991.

[De Castro and Morandi 1987] E De Castro and C Morandi, “Registration of Translated and Rotated Images using Finite Fourier Transforms”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.PAMI-9(5), pp.700-703, 1987.

[Dudgeon 1977] D E Dudgeon, “The Computation of Two-Dimensional Cepstra”, *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol.ASSP-25(6), pp.476-484, 1977.

[Ell 1993] T A Ell, “Quaternion-fourier transforms for analysis of 2-dimensional linear time-invariant partial-differential systems”, *Proc. 32nd IEEE Conf. on Decision and Control, San Antonio, TX, USA, 15-17 December 1993*, IEEE, Control Systems Society, Vol.1-4, pp.1830-1841, 1993.

[Faugeras 1979] O D Faugeras, “Digital Color Image Processing within the Framework of a Human Visual Model”, *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol.ASSP-27(4), pp.380-393, 1979.

[Foley et al. 1990] J D Foley, A Van Dam, S K Feiner and J F Hughes, “*Computer Graphics: Principles and Practice*”, Addison-Wesley Publishing Company, p.165, 1990.

[Frey and Palus 1993] H Frey and H Palus, “Sensor Calibration for Video-Colorimetry”, *Workshop on Design Methodologies for Microelectronics and Signal Processing*, Gliwice-Cracow, Poland, 20-23 Oct. 1993, pp.109-113, 1993.

[Gonzalez and Woods 1992] R C Gonzalez and R E Woods, “Digital Image Processing”, Addison-Wesley Publishing Company, 1992.

[Grace and Spann 1991] A Grace and M Spann, “A comparison between Fourier-Mellin descriptors and moment based features for invariant object recognition using neural networks”, *Pattern Recognition Letters*, Vol.12, pp.635-643, 1991.

[Hamilton 1866] W R Hamilton, "*Elements of Quaternions*", Longmans, Green and Co., London, 1866.

[Hartley 1942] R V L Hartley, "A more symmetrical Fourier Analysis Applied to Transmission Problems", *Proceedings of the IRE*, Vol.30, pp.144-150, 1942.

[Hayes at al. 1980] M H Hayes, J S Lim and A V Oppenheim, "Signal Reconstruction from Phase or Magnitude", *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol.ASSP-28(6), pp.672-680, 1980.

[Healey 1992] G Healey, "Segmenting Images using normalized color", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol.SMC-22(1), pp.64-73, 1992.

[Horner and Gianino 1984] J L Horner and P D Gianino, "Phase-only matched filtering", *Applied Optics*, Vol.23(6), pp.812-816, 1984.

[Jordan III and Bovik 1991] J R Jordan III and A C Bovik, "Using Chromatic Information in Edge-Based Stereo Correspondence", *CVGIP: Image Understanding*, Vol.54(1), pp.98-118, 1991.

[Juday 1995] R D Juday, "Log-polar dappled target", *Optics Letters*, Vol.20(21), pp.2234-2236, 1995.

[Judd and Wyszecki 1975] D B Judd and G Wyszecki, "*Color in Business, Science and Industry*", John Wiley & Sons, New York, USA, p.388, 1975.

[Kemerait and Childers 1972] R C Kemerait and D G Childers, "Signal Detection and Extraction by Cepstrum Techniques", *IEEE Transactions on Information Theory*, Vol.IF-18(6), pp.745-759, 1972.

[Kender 1976] J R Kender, “Saturation, Hue, and Normalized Color: Calculation, Digitization Effects, and Use”, *Tech. Report, Dept. Computer Science, Carnegie-Mellon University*, 1976.

[Kuglin and Hines 1975] C D Kuglin and D C Hines, “The Phase Correlation Image Alignment Method”, *Proceedings 1975 Int. Conf. Cybernetics and Society*, pp.163-165, 1975.

[Lee et al. 1988] D-J Lee, T F Krile and S Mitra, “Power cepstrum and spectrum techniques applied to image registration”, *Applied Optics*, Vol.26(6), pp.1099-1106, 1988.

[Lee et al. 1989] D-J Lee, S Mitra and T F Krile, “Analysis of sequential complex images, using feature extraction and two-dimensional cepstrum techniques”, *Journal of the Optical Society of America A*, Vol.6(6), pp.863-870, 1989.

[Levkowitz and Herman 1993] H Levkowitz and G T Herman, “GLHS: A Generalized Lightness, Hue, and Saturation Color Model*”, CVGIP: Graphical Models and Image Processing*, Vol.55(4), pp.271-285, 1993.

[Li 1992] Y Li, “Reforming the theory of invariant moments for pattern recognition”, *Pattern Recognition*, Vol.25, pp.723-730, 1992.

[Liu and Yan 1994] N Liu and H Yan, “Colour Image edge enhancement by two-channel process”, *Electronics Letters*, Vol.30(12), pp.939-940, 1994.

[Massone et al. 1985] L Massone, G Sandini and V Tagliasco, “‘Form-Invariant’ Topological Mapping Strategy for 2D Shape Recognition”, *Computer Vision, Graphics, and Image Processing*, Vol.30, pp.169-188, 1985.

[McCabe et al. 1997] A McCabe, T Caelli, G West and A Reeves, “Encoding and Processing Spatio-Chromatic Image Information using Complex Fourier Transform Methods”, *Technical Report 97-3, Curtin University of Technology, Perth, Western Australia*, 1997.

[Mital and Goh 1993] D P Mital and W L Goh, “An automatic rotation invariant recognition technique for colour objects and patterns”, *Journal of Microcomputer Applications*, Vol.16, pp.61-69, 1993.

[Moorhead II and Zhu 1995] R J Moorhead II and Z Zhu, “Signal Processing Aspects of Scientific Visualization”, *IEEE Signal Processing Magazine*, Vol.12, No.5, pp.20-41, 1995.

[Morandi et al. 1986] C Morandi, F Piazza and R Capancioni, “Robust 1-D equivalent of Phase-correlation Image Registration Algorithm”, *Electronics Letters*, Vol.22(7), pp.386-388, 1986.

[Nevatia 1977] R Nevatia, “A Color Edge Detector and its use in Scene Segmentation”, *IEEE Transactions on Systems, Man and Cybernetics*, Vol.SMC-7, pp.820-826, 1977.

[Ohta et al. 1980] Y-I Ohta, T Kanade and T Sakai, “Color Information for Region Segmentation”, *Computer Graphics and Image Processing*, Vol.13, pp.222-241, 1980.

[Oppenheim and Lim 1981] A V Oppenheim and J S Lim, “The importance of phase in signals”, *Proceedings of the IEEE*, Vol.69(5), pp.529-541, 1981.

[Papadimitriou and Dennis 1994] D V Papadimitriou and T J Dennis, “Stereo disparity analysis using phase correlation”, *Electronics Letters*, Vol.30(18), pp.1475-1477, 1994.

[Pearson et al. 1977] J J Pearson, D C Hines Jr, S Golosman and C D Kuglin, "Video-rate image correlation processor", *SPIE Application of Digital Image Processing*, Vol.119, pp.197-205, 1977.

[Pittner et al. 1993] S Pittner, J Schneid and C W Ueberhuber, "Wavelet Literature Survey", *Technical Universiy of Vienna, Vienna, Austria*, 1993.

[Pratt 1991] W K Pratt, "*Digital Image Processing*", John Wiley and Sons, New York, USA, pp.310-320 and pp.548-553, 1991.

[Reitböck and Altmann 1984] H J Reitböck and J Altmann, "A model for size and rotation invariant pattern processing in the visual system", *Biological Cybernetics*, Vol.51, pp.113-121, 1984.

[Robbins and Huang 1972] G M Robbins and T S Huang, "Inverse Filtering for Linear Shift-Invariant Imaging Systems", *IEEE Proceedings*, Vol.60(7), pp.862-872, 1972.

[Robinson 1976] G S Robinson, "Color edge detection", *Proc. SPIE Symposium on Advances in Image Transmission Techniques*, Vol.87, San Diego, CA., August 1976.

[Sangwine and Horne 1998a] J Morovic, "Colour management for the graphic arts". In S J Sangwine and R E N Horne, editors, "*The Handbook of Colour Image Processing*", chapter 15, pp.332-357, Chapman and Hall, London, 1998.

[Sangwine and Horne 1998b] S J Sangwine and R E N Horne, editors, "*The Handbook of Colour Image Processing*", Chapman and Hall, London, 1998.

[Sangwine and Horne 1998c] S J Sangwine and A L Thornton, "Frequency domain methods". In S J Sangwine and R E N Horne, editors, "*The Handbook of Colour Image Processing*", chapter 12, pp.228-241, Chapman and Hall, London, 1998.

[Sangwine 1996] S J Sangwine, “Fourier transforms of colour images using quaternion or hypercomplex, numbers”, *Electronics Letters*, Vol.32(21), pp.1979-1980, 1996.

[Sangwine 1997] S J Sangwine, “The discrete quaternion fourier transform”, *IEE 6th Int. Conf. on Image Processing and its Applications*, Dublin, Ireland, Vol.443, pp.790-793, 1997.

[Schalkoff 1989] R J Schalkoff, “*Digital Image Processing and Computer Vision*”, John Wiley & Sons, New York, USA, p.284, 1989.

[Skinner and Childers 1976] D P Skinner and D G Childers, “Real-Time Composite Signal Decomposition”, *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol.ASSP-24, pp.267-270, 1976.

[Smith 1978] A R Smith, “Color Gamut Transform Pairs”, *Computer Graphics*, Vol.12(3), pp.12-19, 1978.

[Strickland et al. 1987] R N Strickland, C-S Kim and W F McDonnell, “Digital Color Image Enhancement based on the Saturation Component”, *Optical Engineering*, Vol.26(7), pp.609-616, 1987.

[Tadmor and Tolhurst 1993] Y Tadmor and D J Tolhurst, “Short Communication: Both the Phase and the Amplitude Spectrum may Determine the Appearance of Natural Images”, *Vision Research*, Vol.33(1), pp.141-145, 1993.

[Thornton and Sangwine 1995] A L Thornton and S J Sangwine, “Colour Object Location using Complex Coding in the Frequency Domain”, *IEE 5th Int. Conf. on Image Processing and its Applications*, Edinburgh, UK, Vol.410, pp.820-824, 1995.

[Thornton and Sangwine 1996] A L Thornton and S J Sangwine, "Colour Object Recognition using Phase Correlation of Log-Polar Transformed Fourier Spectra", *Int. Workshop on Image and Signal Processing '96,* Manchester, UK, pp.615-618, 1996.

[Thornton and Sangwine 1997] A L Thornton and S J Sangwine, "Log-Polar Sampling Incorporating a Novel Spatially Variant Filter to Improve Object Recognition", *IEE 6th Int. Conf. on Image Processing and its Applications*, Dublin, Ireland, No.443, Vol.2, pp.776-779, 1997.

[Tompsett 1979] M F Tompsett, "Video-Signal Generation", printed in "Electronic Imaging", edited by T P McLean and P Schagen, chapter 3, pp. 55-101, 1979.

[Tribolet 1975] J M Tribolet, "A new phase unwrapping Algorithm", *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol.ASSP-25(2), pp.170-177, 1975.

[Watson 1993] K Watson, "Processing Remote Sensing Images using the 2-D FFT - Noise Reduction and other Applications", *Geophysics,* Vol.58(6), pp.835-852, 1993.

[Weiman and Chaikin 1979] C F R Weiman and G Chaikin, "Logarithmic Spiral Grids for Image Processing and Display", *Computer Graphics and Image Processing*, Vol.11, pp.197-226, 1979.

[Wilson and Hodgson 1992] J C Wilson and R M Hodgson, "A Pattern recognition system based on models of aspects of the human visual system", *IEE 4th Int. Conf. on Image Processing and its Applications,* pp.258-261, 1992.

[Woods 1996] J Woods, "Invariant Pattern Recognition: A Review", *Pattern Recognition,* Vol.29, No. 1, pp.1-17, 1996.

[Wyszecki and Stiles 1982] G Wyszecki and W S Stiles, "*Color Science: Concepts and Methods, Quantitative Data and Formulae*", John Wiley & Sons, New York, USA, p.137, 1982.

[Young 1807] T Young, "On the Theory of Light and Colours", *Lectures in Natural Philosophy*, Vol.2, printed for Joseph Johnson, St. Paul's Church Yard, by William Savage, London, 1807, p. 613. An account of some cases of the production of colours, ibid, p.634.

BIBLIOGRAPHY

Brigham, EO, "*The Fast Fourier Transform and its Applications*", Prentice Hall, Englewood Cliffs, New Jersey, 1988.

Bracewell, RN, "*The Fourier transform and its applications*", McGraw-Hill, USA, 1986.

Castleman, KR, "*Digital Image Processing*", Prentice Hall, Englewood Cliffs, New Jersey, 1996.

Foley, JD, van Dam, A, Feiner, S, Hughes, J, "*Computer Graphics: Principles and Practice*", Addison-Wesley Publishing Company, 1990.

R C Gonzalez and R E Woods, "*Digital Image Processing*", Addison-Wesley Publishing Company, 1992.

Hunt, RWG, "*Measuring Colour*", Ellis Horwood Ltd, Chichester, UK, 1991.

Hutson, GH, Shepherd, PJ, Brice, WSJ, "*Colour Television: system principles, engineering practice, and applied technology*", McGraw-Hill, Maidenhead, England, 1990.

Judd, DB, Wyszecki, G, "*Color in Business, Science and Industry*", John Wiley & Sons, New York, USA, 1975.

Kuc, R, "*Introduction to digital signal processing*", McGraw-Hill, Maidenhead, England, 1988.

McLean, TP, Schagen, P, "*Electronic Imaging*", Academic Press, London, UK, 1979.

Pratt, WK, "*Digital Image Processing*", John Wiley & Sons, USA, 1991.

Proakis, JG, Manolakis, DG, "*Introduction to digital signal processing*", Macmillan Publishing Company, New York, 1989.

Rosenfeld, A, Kak, AC, "*Digital Picture Processing: Vol.1*", Academic Press, London, UK, 1982.

Schalkoff, RJ, "*Digital Image Processing and Computer Vision*", John Wiley & Sons, New York, USA, 1989.

Sonka, M, Hlavac, V, Boyle, R, "*Image Processing, analysis and machine vision*", Chapman & Hall, London, UK, 1993.

Steward, EG, "*Fourier Optics: An Introduction*", Ellis Horwood Ltd, Chichester, UK, 1987.

Watkinson, J, "*Introduction to Digital Video*", Butterworth-Heinemann Ltd, Oxford, UK, 1994.

Wright, WD, "*The Measurement of Colour*", Adam Hilger Ltd, London, UK, 1969.

Wyszecki, G, Stiles, WS, "*Color Science: Concepts and Methods, Quantitative Data and Formulae*", John Wiley & Sons, New York, USA, 1982.

APPENDIX A - THE CIE XYZ SYSTEM

In 1931 the CIE defined a standard colorimetric system named the XYZ system which was the result of subjecting many people under strict conditions to colour matching experiments. They were required to match a colour by using a mixture of red (700 nm), green (546.1 nm) and blue (435.8 nm) wavelengths, R, G and B, to create an identical matching colour. The experiments resulted in a set of $\bar{r}$, $\bar{g}$ and $\bar{b}$ colour matching functions which can be used as weighting values determine how much of the R, G and B wavelengths are required to generate a colour. As can be seen from figure A.1, the $\bar{r}$ and $\bar{g}$ values are sometimes negative. To achieve negative values, instead of adding the red or green to the colour 'mixture' it was added to the colour matching reference.

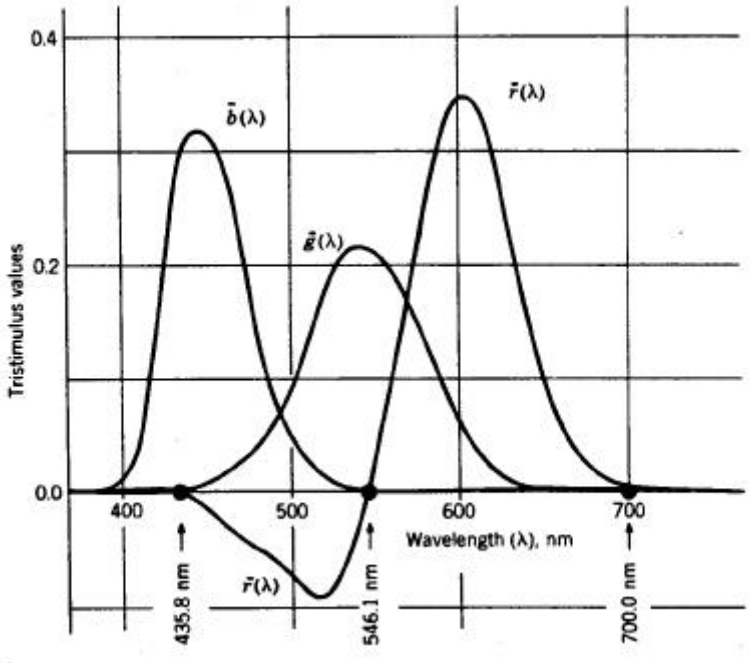


Figure A.1: RGB tristimulus system, from Judd and Wyszecki [1975]

The amounts of R, G and B needed to match any colour can be calculated, (equation A.1a-c), using the spectral energy distribution of the colour, P, and a constant k,

whose value depends upon the spectral energy distribution of the light source chosen as the standard for white. These amounts are denoted *R*, *G* and *B*.

$$R = k\int P(l)\,\bar{r}\,dl \qquad \text{(A.1a)}$$

$$G = k\int P(l)\,\bar{g}\,dl \qquad \text{(A.1b)}$$

$$B = k\int P(l)\,\bar{b}\,dl \qquad \text{(A.1c)}$$

and the chromaticity coordinates can be expressed as shown in equation (A.2a-c).

$$r = \frac{R}{R+G+B} \qquad \text{(A.2a)}$$

$$g = \frac{G}{R+G+B} \qquad \text{(A.2b)}$$

$$b = \frac{B}{R+G+B} \qquad \text{(A.2c)}$$

The luminance of a colour matched using *R*, *G* and *B* is equal to 1.00*R* + 4.5907*G* + 0.0601*B*.

To avoid the use of negative chromaticity values the CIE introduced the new XYZ system. The *X*, *Y* and *Z* tristimulus values can be calculated using the *R*, *G* and *B* tristimulus values, equation (A.3a-c).

$$X = 0.49R + 0.31G + 0.20B \qquad \text{(A.3a)}$$

$$Y = 0.17697R + 0.81240G + 0.01063B \qquad \text{(A.3b)}$$

$$Z = 0.00R + 0.01G + 0.99B \qquad \text{(A.3c)}$$

It should be noted that the ratios of *R*, *G* and *B* for *Y* are (intentionally) identical with the ratios generating luminance using *R*, *G* and *B*.

New colour matching functions $\bar{x}$, $\bar{y}$ and $\bar{z}$ are generated from the $\bar{r}$, $\bar{g}$ and $\bar{b}$ values using equation (A.4a-c).

$$\bar{x} = 0.49\bar{r} + 0.31\bar{g} + 0.2\bar{b} \qquad \text{(A.4a)}$$

$$\bar{y} = 0.17697\bar{r} + 0.81240\bar{g} + 0.01063\bar{b} \qquad \text{(A.4b)}$$

$$\bar{z} = 0.00\bar{r} + 0.01\bar{g} + 0.99\bar{b} \qquad \text{(A.4c)}$$

The colour matching functions using the XYZ system are shown in figure A.2.

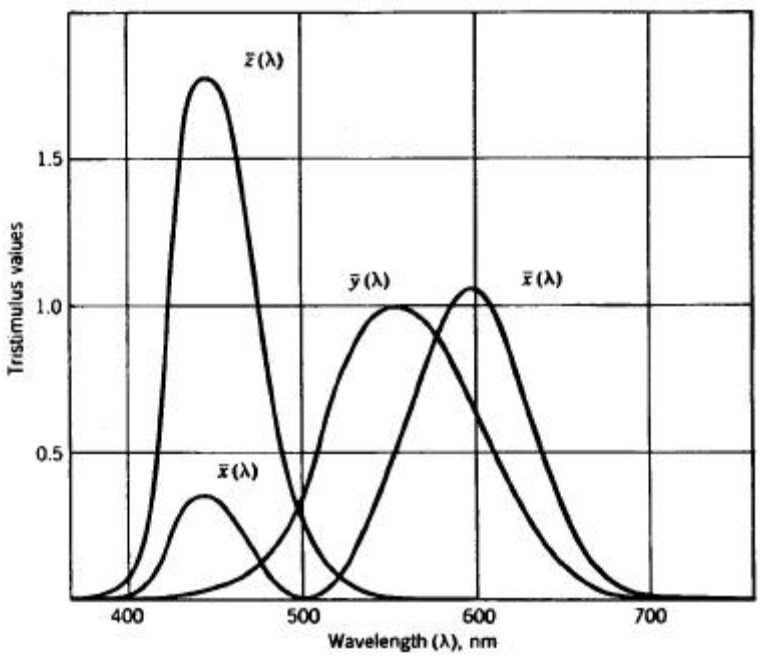


Figure A.2: XYZ tristimulus system, from Judd and Wyszecki [1975]

The amounts of the XYZ primaries needed to match a colour are denoted *X*, *Y* and *Z*. These values are related to the colour matching values using the spectral energy distribution of the colour P(λ) and a constant k, whose value depends upon the spectral energy distribution of the light source chosen as the standard for white, equations A.5a-c.

$$X = k\int P(l)\,\bar{x}\,dl \qquad \text{(A.5a)}$$

$$Y = k\int P(l)\,\bar{y}\,dl \qquad \text{(A.5b)}$$

$$Z = k\int P(l)\,\bar{z}\,dl \qquad \text{(A.5c)}$$

From these *X*, *Y* and *Z* values the chromaticity coordinates x, y and z can be determined using equations (A.6a-c).

$$x = \frac{X}{X+Y+Z} = \frac{0.49r + 0.31g + 0.2b}{0.66697r + 1.13240g + 1.20063b} \qquad \text{(A.6a)}$$

$$y = \frac{Y}{X+Y+Z} = \frac{0.17697r + 0.81240g + 0.01063b}{0.66697r + 1.13240g + 1.20063b} \qquad \text{(A.6b)}$$

$$z = \frac{Z}{X + Y + Z} = \frac{0.00r + 0.01g + 0.99b}{0.66697r + 1.13240g + 1.20063b} \qquad \text{(A.6c)}$$

Since x + y + z = 1 only two of the variables need to be stated in order to determine the third. This result enables the two values x and y to be plotted on a 2D chromaticity diagram as shown in figure A.3.

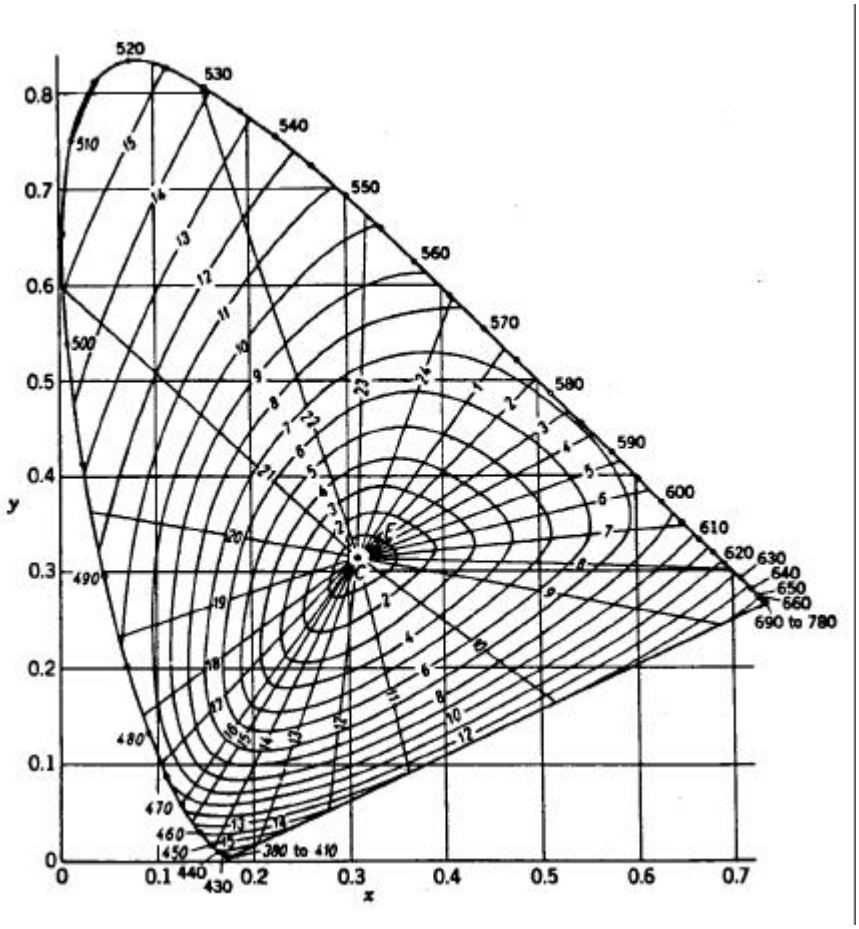


Figure A.3: CIE 1931 chromaticity diagram, from Judd and Wyszecki [1975]

The lines radiating from the centre, neutral point indicate constant hue and the ovals around the neutral point indicate constant saturation. Although this system provides a standard for colour representation it can be seen from figure A.3 that the distribution of colours is very non-uniform. In addition, although Y corresponds to luminance, the *X* and *Z* do not correspond to any perceptual attributes. Thus, although it is important to understand the importance of the CIE standardisation and the reasons why it is necessary, it is more usual to find colour processing performed on other colour representations.

APPENDIX B - EXPERIMENTAL RESULTS

The figures below provide additional examples of testing to supplement the results in the main part of the thesis.

Figures B.1-B.3 illustrate the phase-correlation and cross-correlation of some input images. The arguments of the complex value at each correlation peak is easily seen from the cross-correlation phase image. The argument of the peak is the same when using phase correlation but the phase image is more difficult to interpret.

Figures B.4a - B.4f contain images to enhance the understanding of the processes involved in the block diagram of figure 6.2.

Figure B.5a - B.5g contain test images which were used as inputs to a system which determined the amount of translation, rotation and scaling an object had undergone in relation to a reference object. Figures B.6a - B.6t show the result of using adaptive filtering, introduced in chapter 6, before calculating the phase correlation of images which have undergone a Fourier transform and then a log-polar transform.

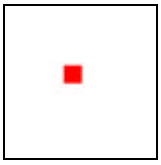


Figure B.1a: First input image (reduced)

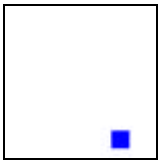


Figure B.1b: Second input image(reduced)

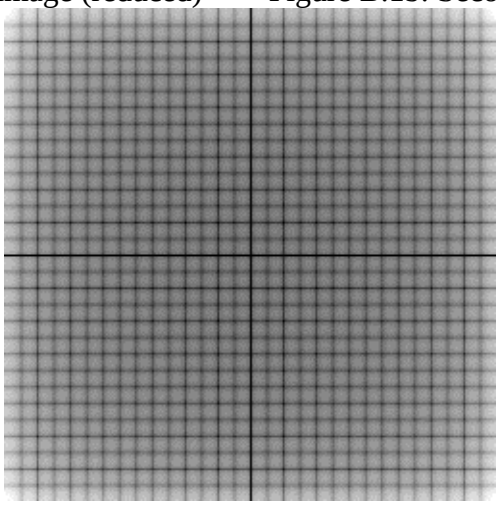
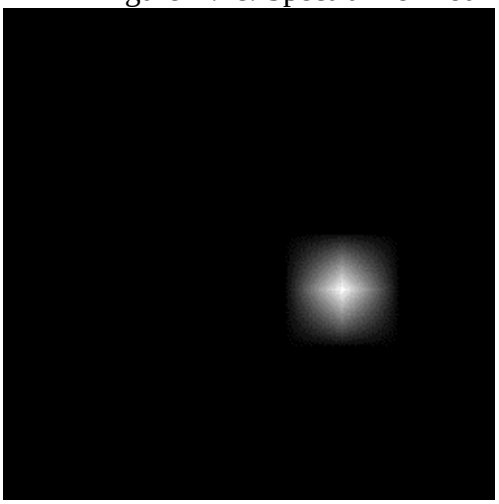


Figure B.1c: Spectrum of Fourier transform of figures B.1a and B.1b



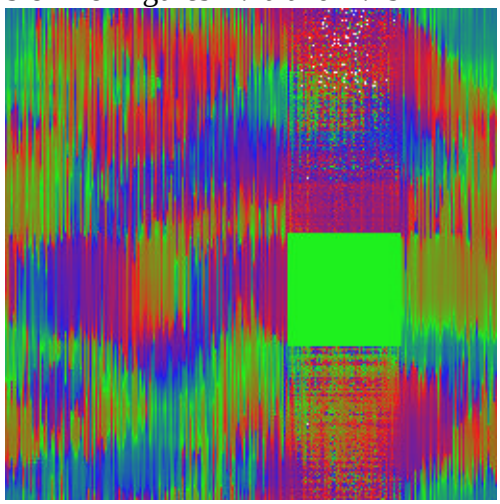
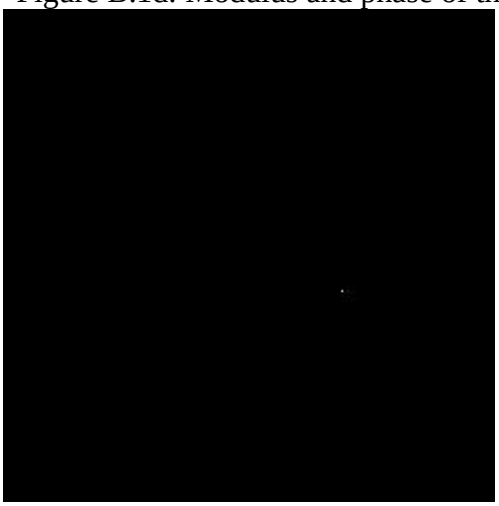


Figure B.1d: Modulus and phase of the cross-correlation of figures B.1a and B.1b



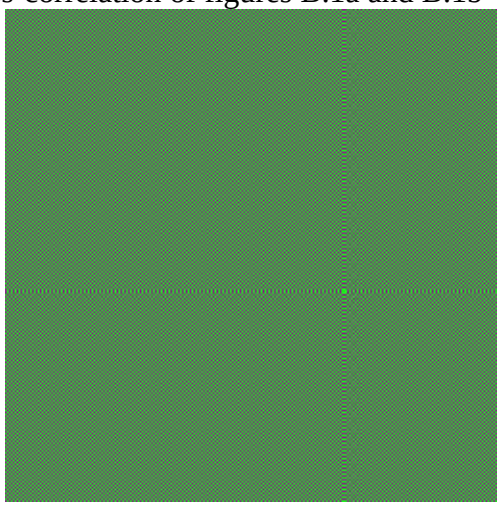


Figure B.1e: Modulus and phase of the phase-correlation of figures B.1a and B.1b

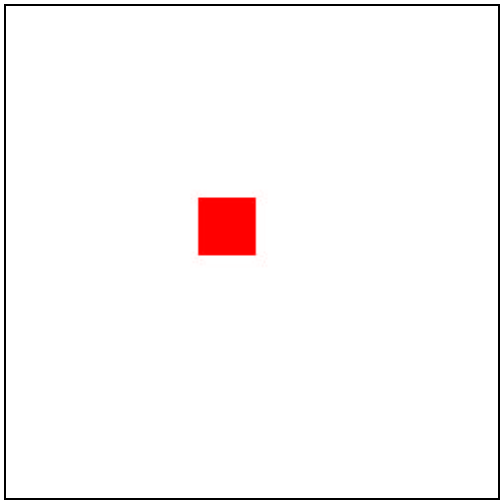


Figure B.2a: First input image

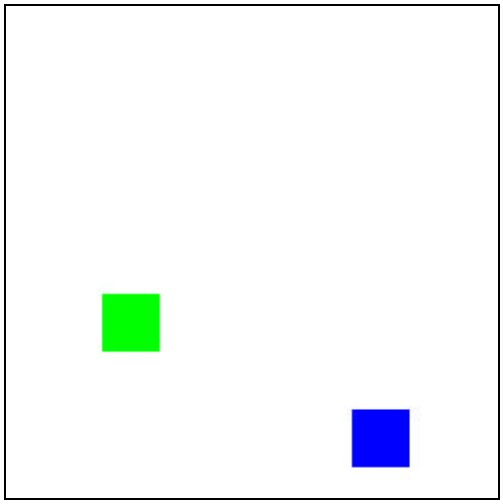
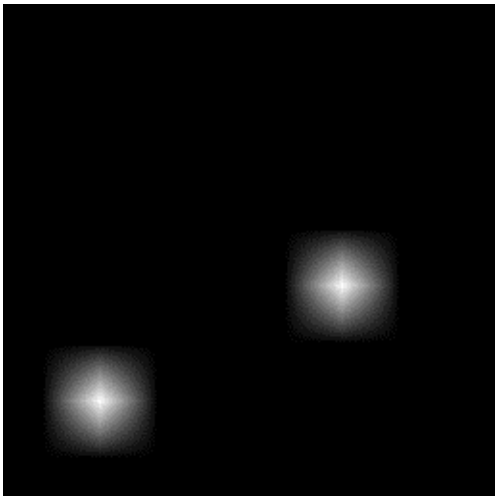


Figure B.2b: Second input image



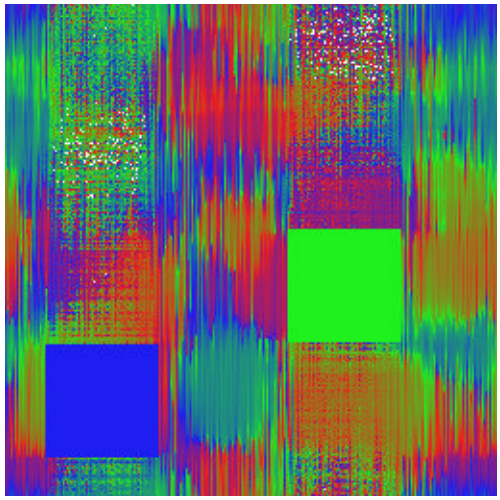
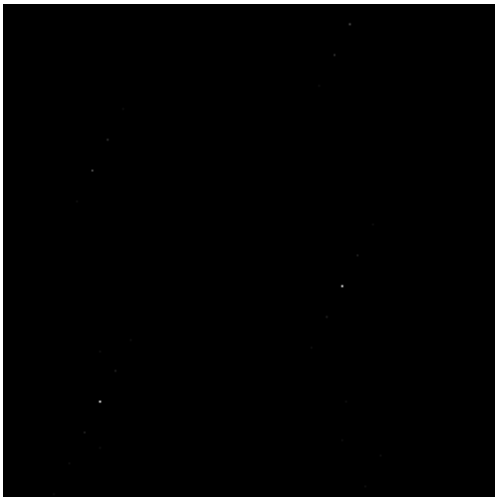


Figure B.2c: Modulus and phase of the cross-correlation of figures B.2a and B.2b



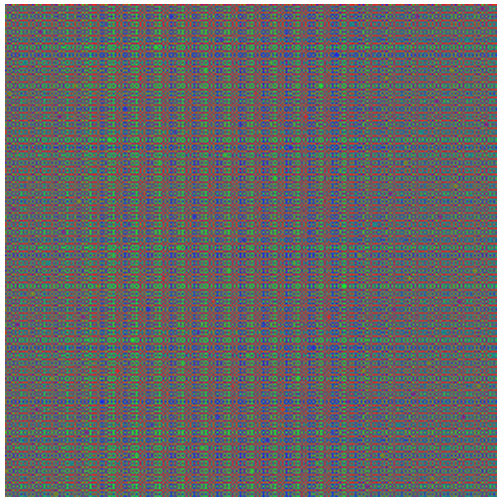


Figure B.2d: Modulus and phase of the phase-correlation of figures B.2a and B.2b

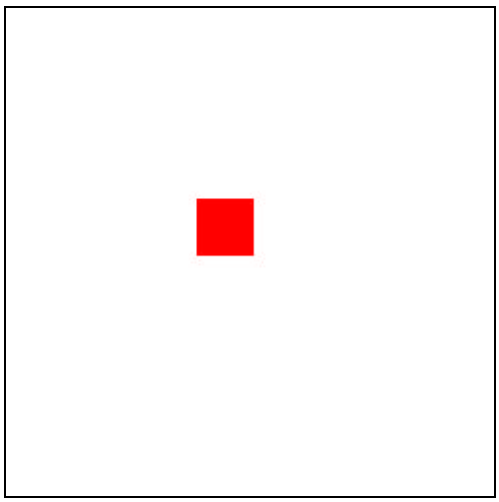


Figure B.3a: First input image

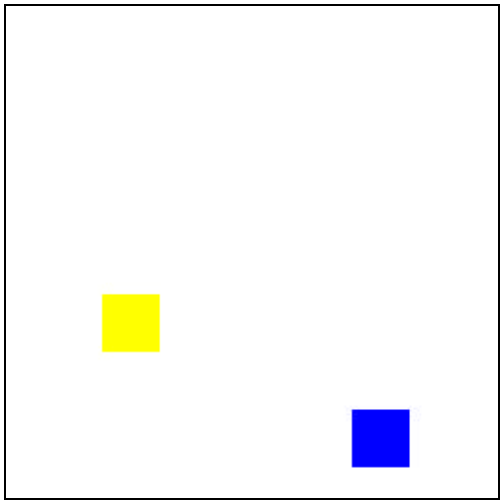
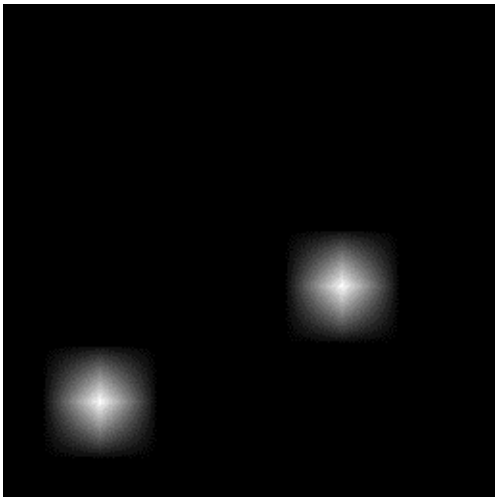


Figure B.3b: Second input image



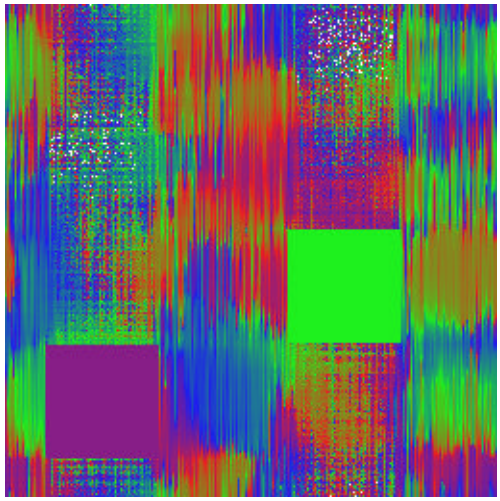


Figure B.3c: Modulus and phase of the cross-correlation of figures B.3a and B.3b

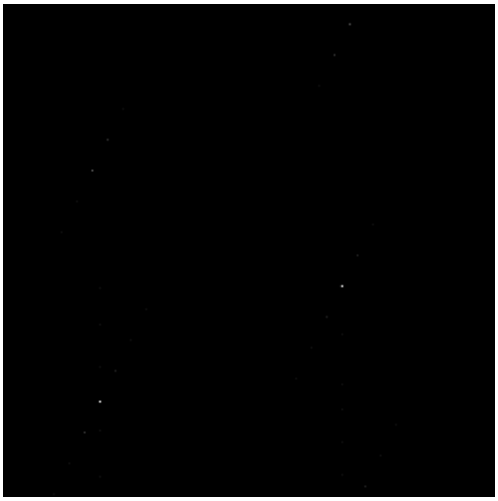


Figure B.3d: Modulus and phase of the phase-correlation of figures B.3a and B.3b

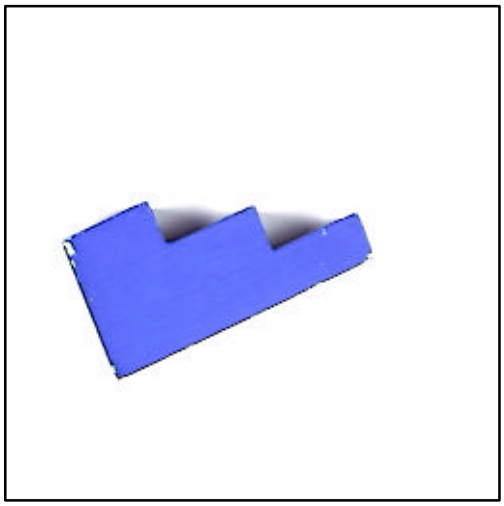


Figure B.4a: Input image A

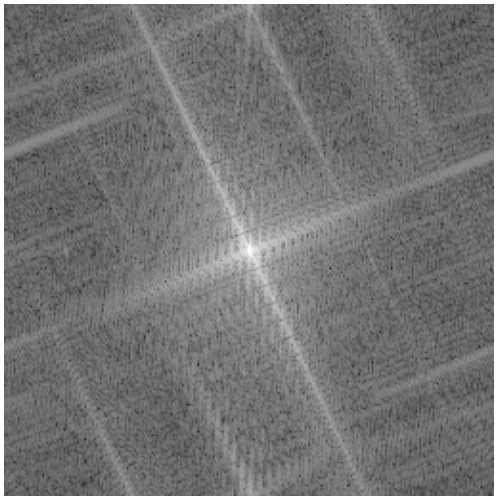


Figure B.4b: Fourier transform of A

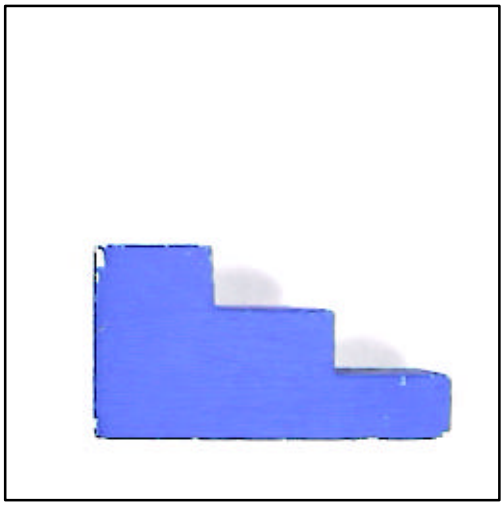


Figure B.4c: Input image B

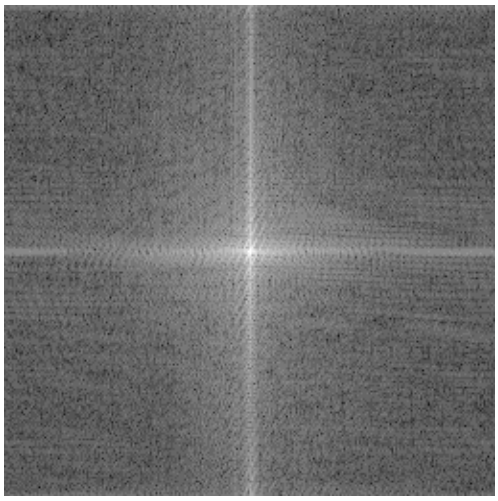


Figure B.4d: Fourier transform of B

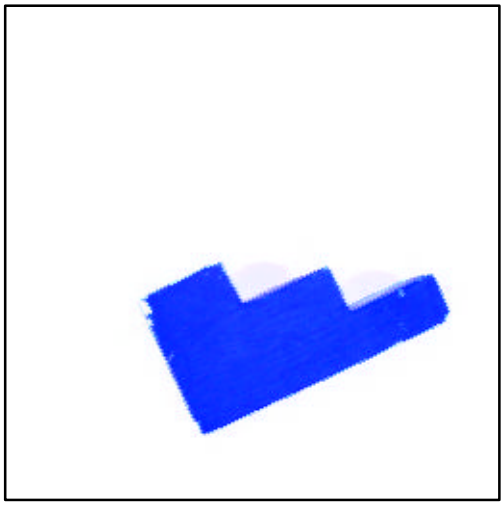


Figure B.4e: Input image C

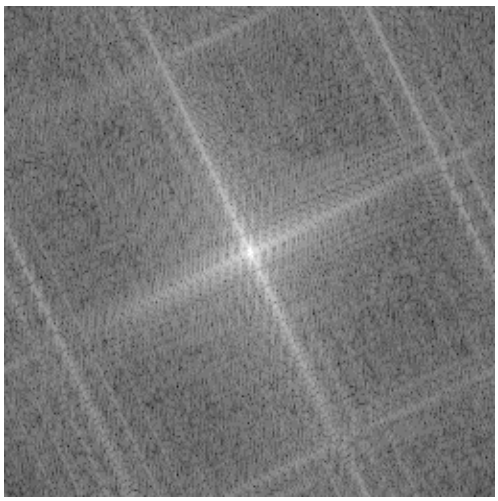


Figure B.4f: Fourier transform of C

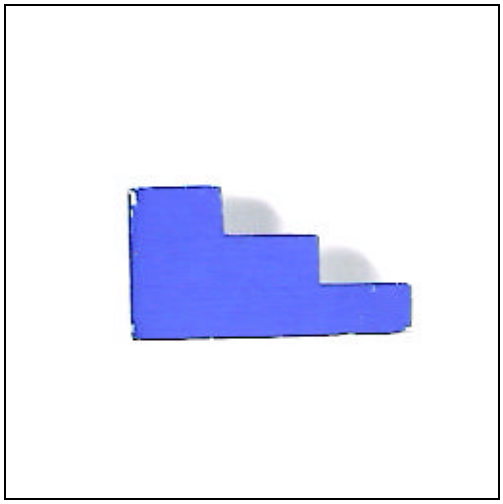


Figure B.5a: Test image A

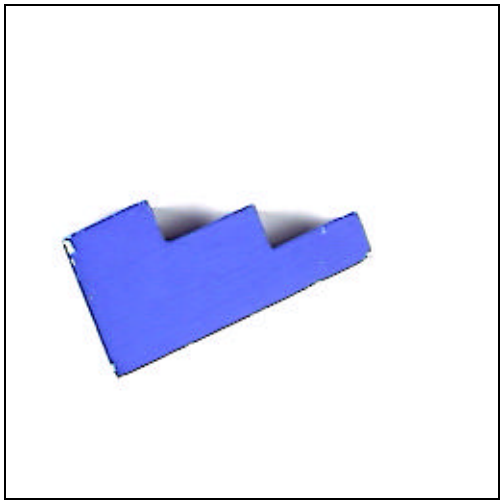


Figure B.5b: Test image B

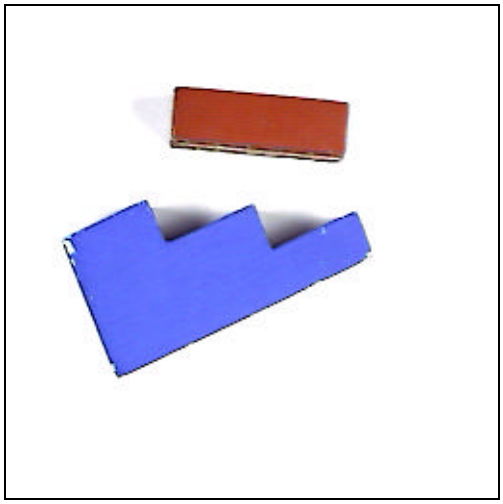


Figure B.5c: Test image C

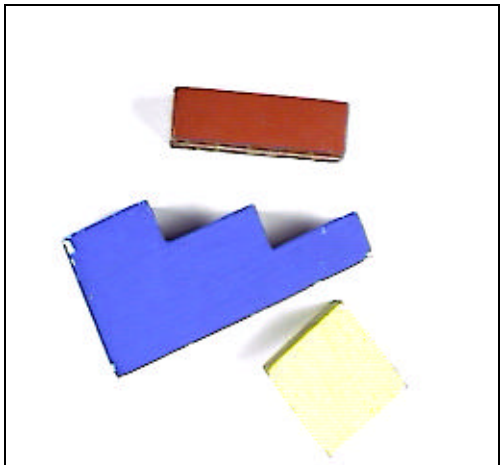


Figure B.5d: Test image D

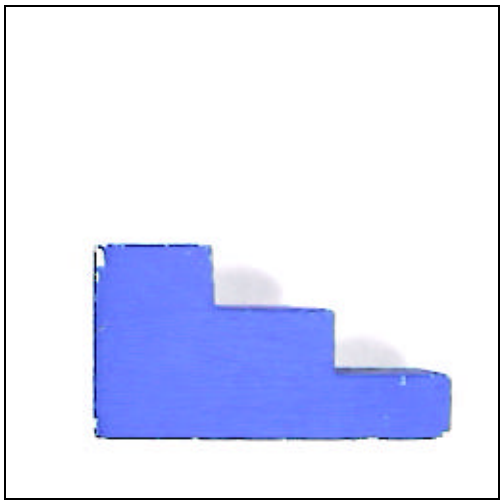


Figure B.5e: Test image E

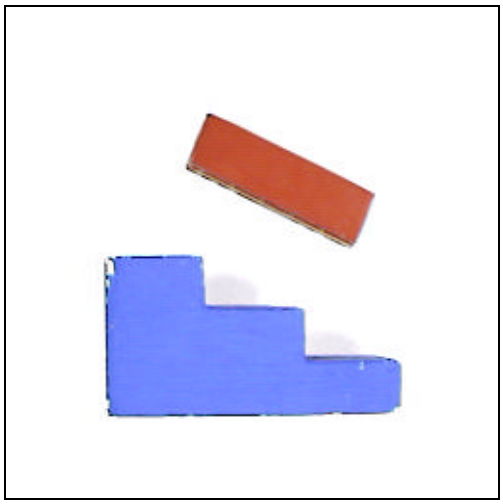


Figure B.5f: Test image F

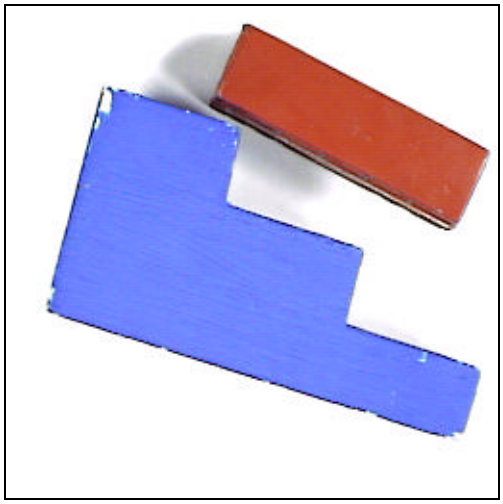
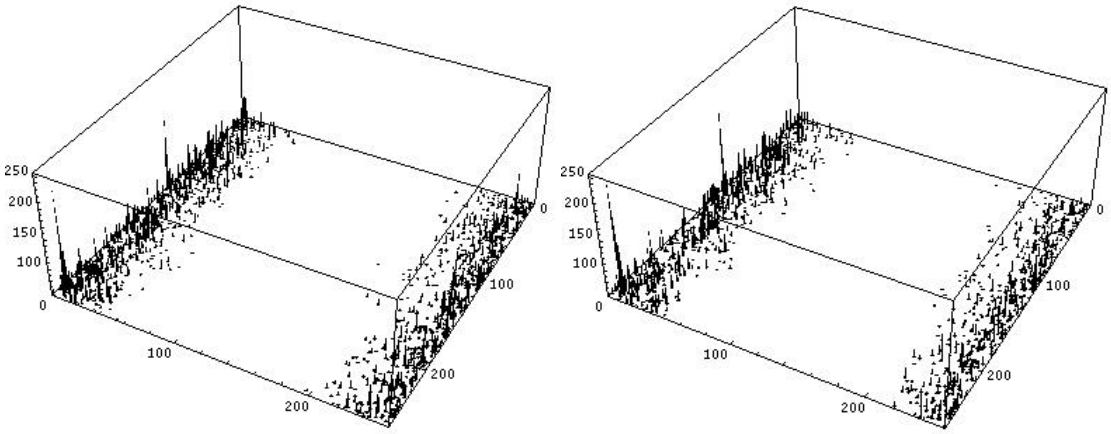
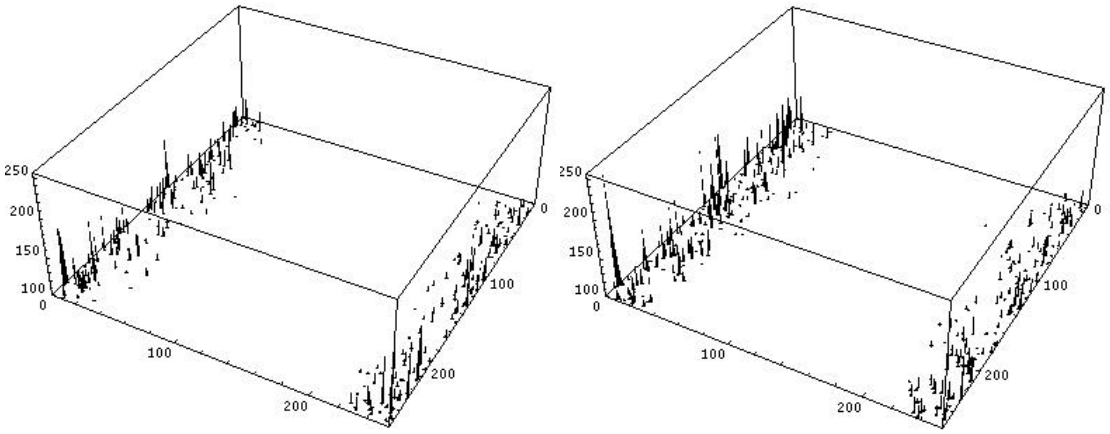


Figure B.5g: Test image G



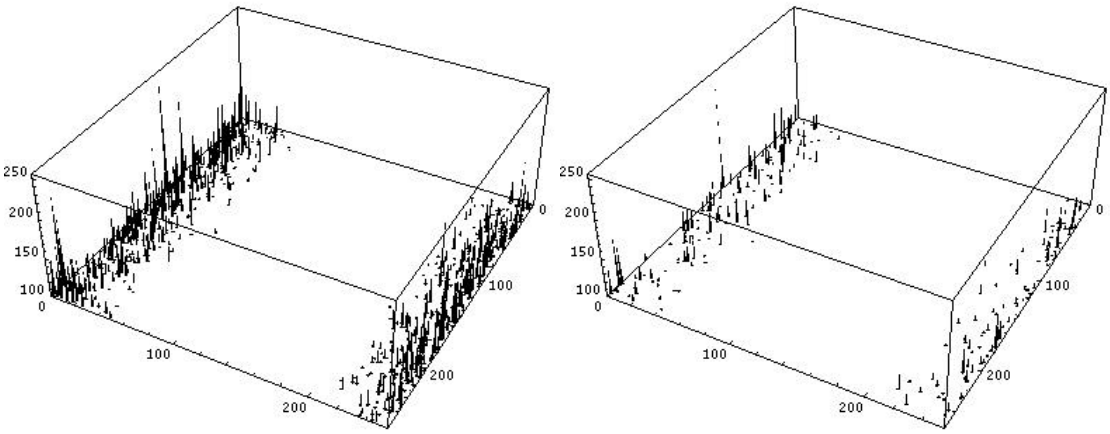
Processing of test images A and B

Figure B.6a: Unfiltered B.6b: Filtered



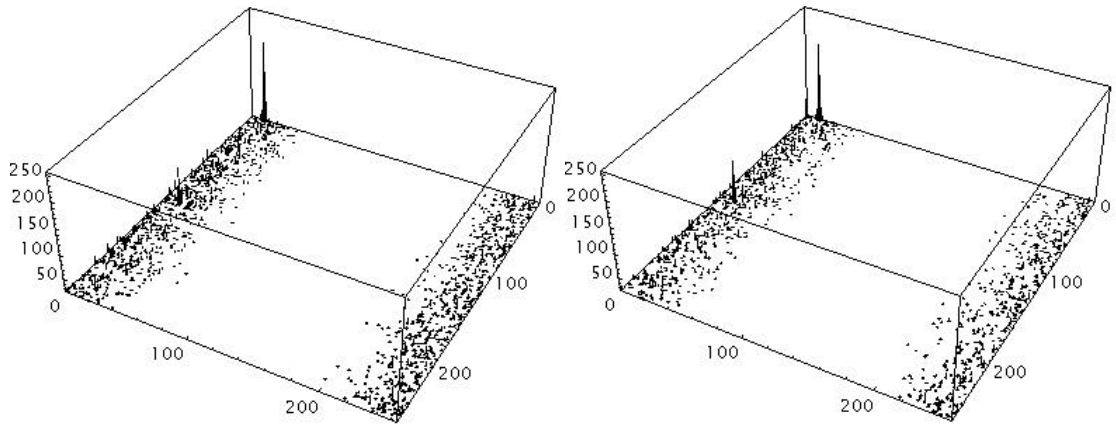
Processing of test images A and C

Figure B.6c: Unfiltered B.6d: Filtered



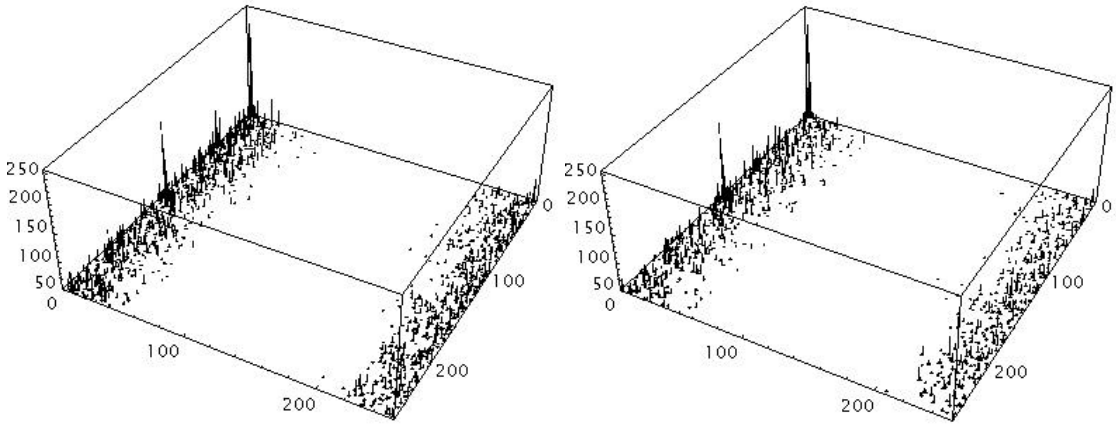
Processing of test images A and D

Figure B.6e: Unfiltered B.6f: Filtered



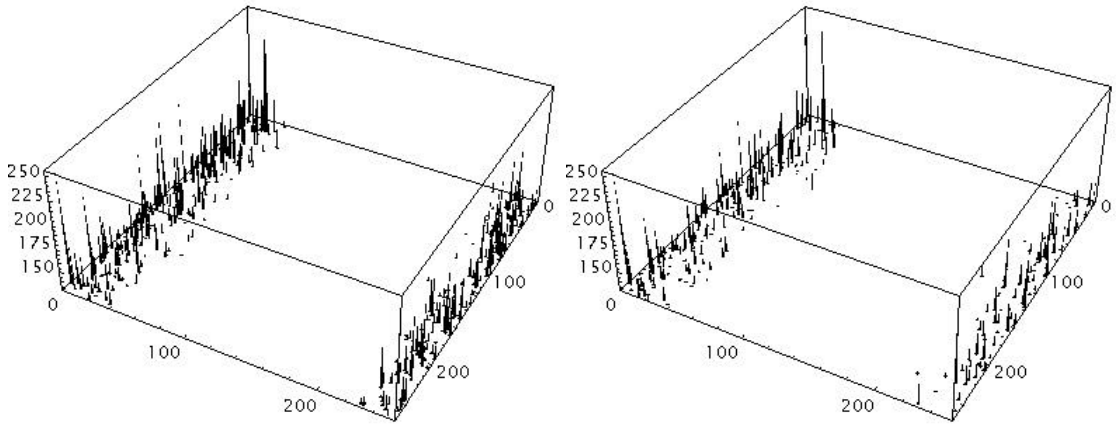
Processing of test images A and E

Figure B.6g: Unfiltered B.6h: Filtered



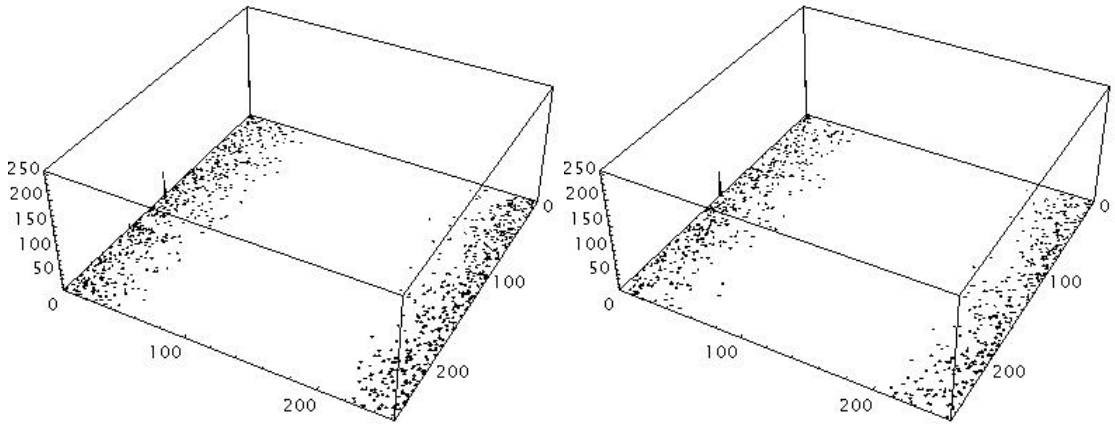
Processing of test images A and F

Figure B.6i: Unfiltered B.6j: Filtered



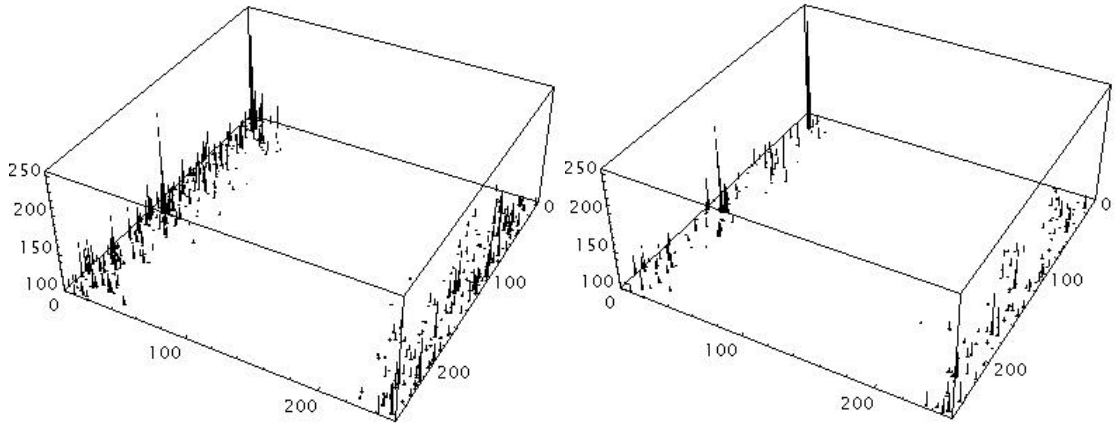
Processing of test images A and G

Figure B.6k: Unfiltered B.6l: Filtered



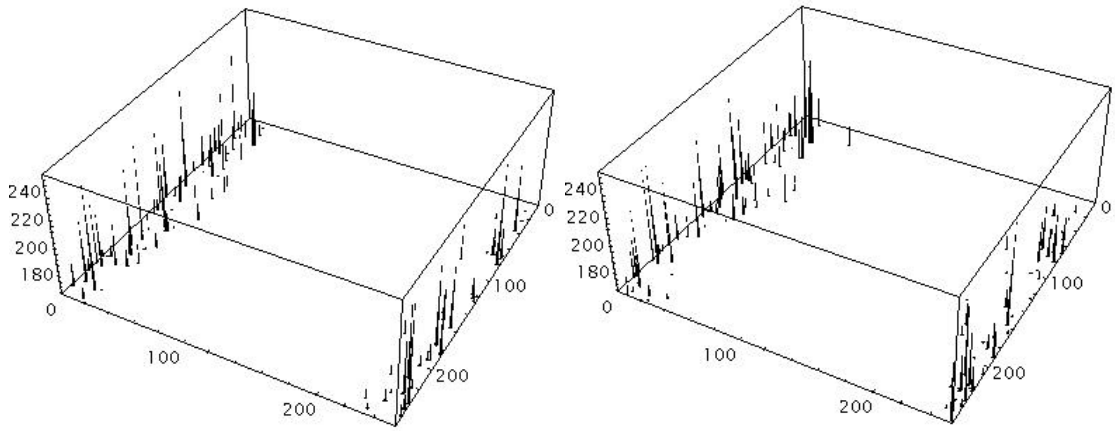
Processing of test images B and D

Figure B.6m: Unfiltered B.6n: Filtered



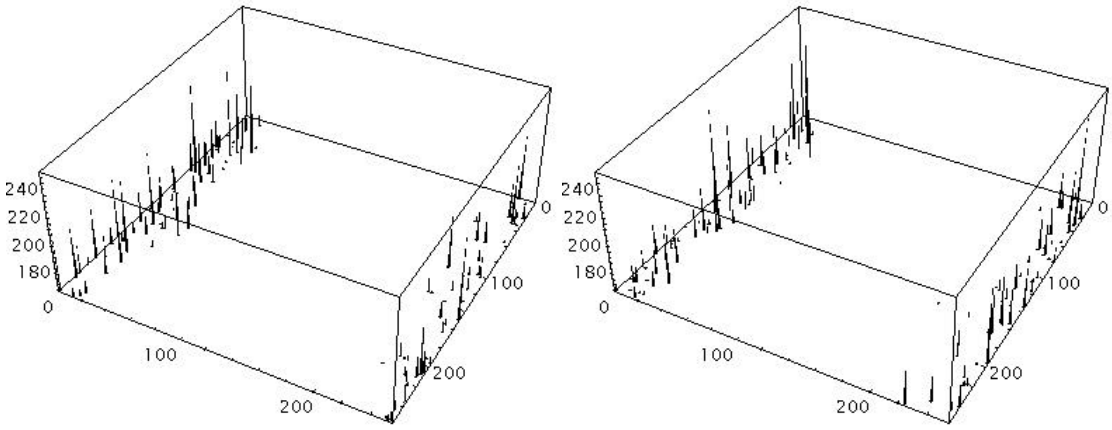
Processing of test images B and E

Figure B.6o: Unfiltered B.6p: Filtered



Processing of test images B and G

Figure B.6q: Unfiltered B.6r: Filtered



Processing of test images C and G

Figure B.6s: Unfiltered B.6t: Filtered

APPENDIX C - SOFTWARE

The development of the research was implemented using ADA and compiled using Alsys Ada for Sun workstations. Many individual programs were written which allowed various programs to be implemented consecutively using a Unix script file which allowed a high level of flexibility. Only the main programs have been included here.

BINTORAST.ADA	Converts an image containing greyscale pixels (0..255) to raster format.
CCEP.ADA	Calculates the complex cepstrum.
CEPSTRUM.ADA	Calculates the power cepstrum.
COLBINTORAST.ADA	Converts a colour binary image to raster format.
COLMANYSQUARE.ADA	Generates many coloured squares on a black background.
COLTOAB.ADA	Converts from colour binary image to Z format.
COLTOHUE.ADA	Converts from colour binary image to hue values.
COLTOINT.ADA	Converts from colour binary image to intensity values.
COMPLOGBMAPFILTERLUTLPF3.ADA	Reads image, circumference oversampling LUT and vertically filters the sampled image.
COMPLOGBMAPFILTERLUTTO256.ADA	Downsamples the vertically filtered image to obtain a square image.
COMPLOGBMAPFILTERMKLUT.ADA	Generates the circumference oversampling LUT for log-polar mapping.
COMPLOGBMAPFILTERRADLUTLPF3.ADA	Reads image, radial oversampling LUT and horizontally filters the sampled image.

COMPLOGBMAPFILTERRADLUTTO256.ADA	Downsamples the horizontally filtered image to obtain a square image.
COMPLOGBMAPFILTERRADMKLUT.ADA	Generates the radial oversampling LUT for log-polar mapping.
COMPLOGBVHLUTLPFBILININT.ADA	Reads image, circumferential and radial oversampling LUT and horizontally filters the sampled image using bilinear interpolation.
COMPLOGBVHLUTTO256.ADA	Downsamples the horizontally and vertically filtered image to obtain a square image.
COMPLOGBVHMKLUT.ADA	Generates the circumferential and radial oversampling LUT for log-polar mapping.
COMPTOPHASE.ADA	Calculates the argument for each pixel in a complex image.
FFTC.ADA	Calculates the FFT of a complex image.
HUEIMAGE_SICONSTANT.ADA	Reads a hue image, requests constant saturation and intensity values and outputs a colour binary image.
IABTOCOL.ADA	Generates a colour binary image from IZ values.
IFFTC.ADA	Calculates the inverse FFT of a complex input.
MAG.ADA	Reads in a complex image and alters the magnitude of each pixel to a set value.
PCTIMESALPHA.ADA	Calculates the correlation of two images. Allows variation of the normalisation value.
PHASESET.ADA	Reads in a complex image and alters the phase of each pixel to a set value.
RASTTOBIN.ADA	Generates the binary file from the raster image.

RASTTOCOLBIN.ADA	Generates the colour binary file from the raster image.
SNRC.ADA	Allows the signal-to-noise ratio to be calculated.

BINTORAST.ADA

```
with TEXT_IO, BINARY_IMAGE_IO, SYSTEM_ENVIRONMENT, ELEMENTARY_FUNCTIONS,
     SIZE_OF_IMAGE, SIZE_OF_FILE, PIXEL_TYPES, COMMON_IMAGE_TYPES,
     RASTER_IO;
use  TEXT_IO, PIXEL_TYPES;

-- Generates a raster image from a binary image file.
-- Author: Amy Thornton

procedure bintorast is

   BINIMAGE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
   RASTIMAGE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);

  SIZE : constant NATURAL := 2**SIZE_OF_IMAGE(SIZE_OF_FILE(BINIMAGE));

  package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
  use IMAGES_N;

  type IMAGE is array(SQUARE_COORDINATE, SQUARE_COORDINATE) of PIXEL;
  pragma PACK(IMAGE);
  I : IMAGE;

  package READ_BINARY_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                                   SQUARE_COORDINATE, PIXEL, IMAGE);
  package WRITE_BINARY_IMAGE is new RASTER_IO(SQUARE_COORDINATE,
                                              SQUARE_COORDINATE, PIXEL, IMAGE);

begin
  READ_BINARY_IMAGE.READ_BINARY_IMAGE_FILE(BINIMAGE, I);
  WRITE_BINARY_IMAGE.WRITE_RASTER_FILE(RASTIMAGE, I);
end bintorast;
```

CCEP.ADA

```
with TEXT_IO, SYSTEM_ENVIRONMENT, BINARY_IMAGE_IO, ELEMENTARY_FUNCTIONS,
     SIZE_OF_IMAGE, SIZE_OF_FILE, COMPLEX_NUMBERS_FLOAT, PIXEL_TYPES, LOG10,
     COMMON_IMAGE_TYPES, TWO_DIMENSIONAL_FOURIER_TRANSFORMS, FLOAT_MAX,
     FLOAT_MIN, IMAGE_COORDINATES;
use  TEXT_IO, COMPLEX_NUMBERS_FLOAT, ELEMENTARY_FUNCTIONS, PIXEL_TYPES,
     IMAGE_COORDINATES;

-- Calculates the complex cepstrum of a complex image.
-- Author: Amy Thornton

procedure ccep is

   INPUT_FILENAME : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
   BINCOMPFILE        : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);

  SIZE     : constant NATURAL :=
2**SIZE_OF_IMAGE(SIZE_OF_FILE(INPUT_FILENAME)/(2 * FLOAT'SIZE/8));

  package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
  use IMAGES_N;

  package FFT_N is new TWO_DIMENSIONAL_FOURIER_TRANSFORMS(FLOAT,
                                                          COMPLEX_NUMBER,
                                                          SQUARE_COORDINATE,
                                                          FLOAT_IMAGES.IMAGE,
                                                       COMPLEX_IMAGES.IMAGE);
  type COMPLEX_IMAGE_PTR is access COMPLEX_IMAGES.IMAGE;
  SP : constant COMPLEX_IMAGE_PTR := new COMPLEX_IMAGES.IMAGE;
  S  : COMPLEX_IMAGES.IMAGE renames SP.all;

  type    REAL_IMAGE_PTR is access FLOAT_IMAGES.IMAGE;
  RP : constant    REAL_IMAGE_PTR := new FLOAT_IMAGES.IMAGE;
  R  : FLOAT_IMAGES.IMAGE renames RP.all;
  ANGP : constant    REAL_IMAGE_PTR := new FLOAT_IMAGES.IMAGE;
  ANG  : FLOAT_IMAGES.IMAGE renames ANGP.all;

  type IMAGE_PTR is access PIXEL_IMAGES.IMAGE;
  IP : constant IMAGE_PTR := new PIXEL_IMAGES.IMAGE;
  I  : PIXEL_IMAGES.IMAGE renames IP.all;

  package READ_COMPLEX_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                      SQUARE_COORDINATE, COMPLEX_NUMBER, COMPLEX_IMAGES.IMAGE);
  use READ_COMPLEX_IMAGE;

  package WRITE_COMPLEX_IMAGE is new BINARY_IMAGE_IO
                                      (SQUARE_COORDINATE, SQUARE_COORDINATE,
                  COMPLEX_NUMBERS_FLOAT.COMPLEX_NUMBER, COMPLEX_IMAGES.IMAGE);
  use WRITE_COMPLEX_IMAGE;

  package WRITE_FLOAT_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                SQUARE_COORDINATE, FLOAT, FLOAT_IMAGES.IMAGE);
  use WRITE_FLOAT_IMAGE;

  package WRITE_BINARY_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                SQUARE_COORDINATE, PIXEL, PIXEL_IMAGES.IMAGE);
  use WRITE_BINARY_IMAGE;

  package Real_IO is new FLOAT_IO(FLOAT); use Real_IO;
  package Coord_IO is new INTEGER_IO(COORDINATE);use Coord_IO;

  MN: constant FLOAT := FLOAT(SIZE) * FLOAT(SIZE);
  scale: character;

begin
  READ_COMPLEX_IMAGE.READ_BINARY_IMAGE_FILE(INPUT_FILENAME, S);
  FFT_N.FFT_BY_ROW_COLUMN_SEPARATION(S);                        --FFT of image
  declare
```

```
    function MAXIMUM_MODULUS(I : COMPLEX_IMAGES.IMAGE) return FLOAT is
      M : FLOAT := 0.0;
    begin
      for y in I'RANGE(1) loop
        for x in I'RANGE(2) loop
          M := FLOAT_MAX(M, MODULUS(I(y,x)));
        end loop;
      end loop;
      return M;
    end MAXIMUM_MODULUS;
  begin
    declare
      function MAXIMUM_BUTONE_MODULUS(I : COMPLEX_IMAGES.IMAGE) return FLOAT
is
        M : FLOAT := 0.0;
        Mtmp : FLOAT;
        max : FLOAT := MAXIMUM_MODULUS(I);
      begin
        for y in I'RANGE(1) loop
          for x in I'RANGE(2) loop
            Mtmp := M;
            M := FLOAT_MAX(Mtmp, MODULUS(I(y,x)));
            if M = max then
              M := FLOAT_MIN(Mtmp, MODULUS(I(y,x)));
            end if;
          end loop;
        end loop;
        return M;
      end MAXIMUM_BUTONE_MODULUS;
    begin
      declare
        MAX_SPECTRAL_VALUE : constant FLOAT := MAXIMUM_MODULUS(S);
        MAX_BUTONE_SPECTRAL_VALUE : constant FLOAT :=
MAXIMUM_BUTONE_MODULUS(S);
        subtype INTENSITY is FLOAT range 0.0 .. 1.0;
        function PIXEL_VALUE(I : INTENSITY) return PIXEL is
          FACTOR: constant FLOAT := 255.0 /LOG10(1.0E6);
        begin
          if I < 1.0E-6 then
            return 0;
          else
            return PIXEL(LOG10(I * 1.0E6) * FACTOR);
          end if;
        end PIXEL_VALUE;
      begin
        for Y in S'RANGE(1) loop
          for X in S'RANGE(2) loop
            begin
              R(y,x) := LOG(MODULUS(S(y,x)));       --Log of spectrum of image
            exception
              when ARGUMENT_ERROR =>
                put("*");
                R(y,x) := MAX_BUTONE_SPECTRAL_VALUE;
            end;
            begin
              ANG(y,x) := ELEMENTARY_FUNCTIONS.ARCTAN(
                                        IMAGINARY(S(y,x)), REAL(S(y,x)));
            exception
              when ARGUMENT_ERROR =>                    --Phase of FFT'd image
                ANG(y,x) := 0.0;
                put("!");
            end;
            S(y,x) := COMPLEX(R(y,x), ANG(y,x));
            S(y,x) := CONJUGATE(S(y,x))/MN;
          end loop;
        end loop;
        NEW_LINE;
        FFT_N.FFT_BY_ROW_COLUMN_SEPARATION(S);   --IFFT of log of S/P of image
```

```
        for Y in S'RANGE(1) loop
          for X in S'RANGE(2) loop
            S(y,x) := CONJUGATE(S(y,x));
          end loop;
        end loop;
        put("Output the unscaled cepstrum? (yes: y = complex, no: b = binary,
f = float) ");
        get(scale);
        if scale = 'y' then
          WRITE_COMPLEX_IMAGE.WRITE_BINARY_IMAGE_FILE(BINCOMPFILE, S);
        end if;
        declare
          MAX_SPECTRAL_VALUE : constant FLOAT := MAXIMUM_MODULUS(S);
        begin
          put("Max value is: "); put(MAX_SPECTRAL_VALUE); NEW_LINE;
          for Y in S'RANGE(1) loop
            for X in S'RANGE(2) loop
              R(y,x) := MODULUS(S(y,x))/MAX_SPECTRAL_VALUE;
              I(y,x) := PIXEL_VALUE(R(y,x));
            end loop;
          end loop;              --Spectrum of FFT of log of spectrum of image
        end;
      end;
    end;
    if scale = 'b' then
      WRITE_BINARY_IMAGE.WRITE_BINARY_IMAGE_FILE(BINCOMPFILE, I);
    end if;
    if scale = 'f' then
      WRITE_FLOAT_IMAGE.WRITE_BINARY_IMAGE_FILE(BINCOMPFILE, R);
    end if;
  end;
end ccep;
```

CEPSTRUM.ADA

```
with TEXT_IO, SYSTEM_ENVIRONMENT, BINARY_IMAGE_IO, ELEMENTARY_FUNCTIONS,
     SIZE_OF_IMAGE, SIZE_OF_FILE, COMPLEX_NUMBERS_FLOAT, PIXEL_TYPES, LOG10,
     COMMON_IMAGE_TYPES, TWO_DIMENSIONAL_FOURIER_TRANSFORMS, FLOAT_MAX,
     IMAGE_COORDINATES;
use  TEXT_IO, COMPLEX_NUMBERS_FLOAT, ELEMENTARY_FUNCTIONS, PIXEL_TYPES,
     IMAGE_COORDINATES;

-- Calculates the power cepstrum of a complex image.
-- Author: Amy Thornton

procedure cepstrum is

   INPUT_FILENAME : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
   BINCOMPFILE        : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);

  SIZE     : constant NATURAL :=
2**SIZE_OF_IMAGE(SIZE_OF_FILE(INPUT_FILENAME)/ (2 * FLOAT'SIZE/8));

  package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
  use IMAGES_N;

  package FFT_N is new TWO_DIMENSIONAL_FOURIER_TRANSFORMS(FLOAT,
                                                          COMPLEX_NUMBER,
                                                          SQUARE_COORDINATE,
                                                          FLOAT_IMAGES.IMAGE,
                                                        COMPLEX_IMAGES.IMAGE);

  type COMPLEX_IMAGE_PTR is access COMPLEX_IMAGES.IMAGE;
  SP : constant COMPLEX_IMAGE_PTR := new COMPLEX_IMAGES.IMAGE;
  S  : COMPLEX_IMAGES.IMAGE renames SP.all;

  type    REAL_IMAGE_PTR is access FLOAT_IMAGES.IMAGE;
  RP : constant    REAL_IMAGE_PTR := new FLOAT_IMAGES.IMAGE;
  R  : FLOAT_IMAGES.IMAGE renames RP.all;

  type IMAGE_PTR is access PIXEL_IMAGES.IMAGE;
  IP : constant IMAGE_PTR := new PIXEL_IMAGES.IMAGE;
  I  : PIXEL_IMAGES.IMAGE renames IP.all;

  package READ_COMPLEX_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                       SQUARE_COORDINATE, COMPLEX_NUMBER, COMPLEX_IMAGES.IMAGE);
  use READ_COMPLEX_IMAGE;

  package WRITE_COMPLEX_IMAGE is new BINARY_IMAGE_IO
                                       (SQUARE_COORDINATE, SQUARE_COORDINATE,
                   COMPLEX_NUMBERS_FLOAT.COMPLEX_NUMBER, COMPLEX_IMAGES.IMAGE);
  use WRITE_COMPLEX_IMAGE;

  package WRITE_BINARY_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                SQUARE_COORDINATE, PIXEL, PIXEL_IMAGES.IMAGE);
  use WRITE_BINARY_IMAGE;

  package Real_IO is new FLOAT_IO(FLOAT); use Real_IO;
  package Coord_IO is new INTEGER_IO(COORDINATE);use Coord_IO;

  scale: character;

begin
  READ_COMPLEX_IMAGE.READ_BINARY_IMAGE_FILE(INPUT_FILENAME, S);
  FFT_N.FFT_BY_ROW_COLUMN_SEPARATION(S);                          --FFT of image
  declare
    function MAXIMUM_MODULUS(I : COMPLEX_IMAGES.IMAGE) return FLOAT is
      M : FLOAT := 0.0;
    begin
      for y in I'RANGE(1) loop
        for x in I'RANGE(2) loop
          M := FLOAT_MAX(M, MODULUS(I(y,x)));
```

```
        end loop;
      end loop;
      return M;
    end MAXIMUM_MODULUS;
  begin
    declare
      MAX_SPECTRAL_VALUE : constant FLOAT := MAXIMUM_MODULUS(S);
      subtype INTENSITY is FLOAT range 0.0 .. 1.0;
      function PIXEL_VALUE(I : INTENSITY) return PIXEL is
        FACTOR: constant FLOAT := 255.0 /LOG10(1.0E6);
      begin
        if I < 1.0E-6 then
          return 0;
        else
          return PIXEL(LOG10(I * 1.0E6) * FACTOR);
        end if;
      end PIXEL_VALUE;
    begin
      for X in S'RANGE(1) loop
        for Y in S'RANGE(2) loop
          begin
            R(y,x) := LOG(MODULUS(S(y,x)));         --Log of spectrum of image
          exception
            when ARGUMENT_ERROR =>
              put("x = "); put(x); put("y = "); put(y);
              put(MODULUS(S(y,x))); put(MAX_SPECTRAL_VALUE); NEW_LINE;
              R(y,x) := -20.0;
          end;
        end loop;
      end loop;
      FFT_N.REAL_ARRAY_TO_COMPLEX_ARRAY(R, S);
      FFT_N.FFT_BY_ROW_COLUMN_SEPARATION(S); --FFT of log of spectrum of image
      put("Output the unscaled cepstrum? (y = complex, n = binary) ");
      get(scale);
      if scale = 'y' then
        WRITE_COMPLEX_IMAGE.WRITE_BINARY_IMAGE_FILE(BINCOMPFILE, S);
      end if;
      declare
        MAX_SPECTRAL_VALUE : constant FLOAT := MAXIMUM_MODULUS(S);
      begin
        put("Max value is: "); put(MAX_SPECTRAL_VALUE); NEW_LINE;
        for X in S'RANGE(1) loop
          for Y in S'RANGE(2) loop
            I(y,x) := PIXEL_VALUE(MODULUS(S(y,x))/MAX_SPECTRAL_VALUE);
          end loop;
        end loop;               --Spectrum of FFT of log of spectrum of image
      end;
    end;
  end;
  if scale /= 'y' then
    WRITE_BINARY_IMAGE.WRITE_BINARY_IMAGE_FILE(BINCOMPFILE, I);
  end if;
end cepstrum;
```

COLBINTORAST.ADA

```
with TEXT_IO, BINARY_IMAGE_IO, SYSTEM_ENVIRONMENT, ELEMENTARY_FUNCTIONS,
     SIZE_OF_IMAGE, SIZE_OF_FILE, PIXEL_TYPES, COMMON_IMAGE_TYPES,
     RASTER_IO;
use  TEXT_IO, PIXEL_TYPES;

-- Generates a raster image from a colour binary file.
-- Author: Amy Thornton

procedure colbintorast is

   COLBINIMAGE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
   COLRASTIMAGE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);

  SIZE : constant NATURAL := 2**SIZE_OF_IMAGE(SIZE_OF_FILE(COLBINIMAGE)/3);

  package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
  use IMAGES_N;

  type COLOUR_IMAGE is array(SQUARE_COORDINATE, SQUARE_COORDINATE) of
                                                                COLOUR_PIXEL;
  pragma PACK(COLOUR_IMAGE);
  ICol : COLOUR_IMAGE;

  package READ_COL_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                SQUARE_COORDINATE, COLOUR_PIXEL, COLOUR_IMAGE);
  package WRITE_COL_IMAGE is new RASTER_IO(SQUARE_COORDINATE,
                                SQUARE_COORDINATE, COLOUR_PIXEL, COLOUR_IMAGE);

begin
  READ_COL_IMAGE.READ_BINARY_IMAGE_FILE(COLBINIMAGE, ICol);
  WRITE_COL_IMAGE.WRITE_RASTER_FILE(COLRASTIMAGE, ICol);
end colbintorast;
```

COLMANYSQUARE.ADA

```
with BINARY_IMAGE_IO, ELEMENTARY_FUNCTIONS, TEXT_IO, SYSTEM_ENVIRONMENT,
     PIXEL_TYPES, COMMON_IMAGE_TYPES, IMAGE_COORDINATES, ELEMENTARY_FUNCTIONS,
     COMPLEX_NUMBERS_FLOAT;
use  TEXT_IO, PIXEL_TYPES, IMAGE_COORDINATES, ELEMENTARY_FUNCTIONS,
     COMPLEX_NUMBERS_FLOAT;

-- Generates a complex image containing coloured squares on a black
background.
-- Author: Amy Thornton

procedure colmanysquare is

package INT_IO is new INTEGER_IO(INTEGER);
use INT_IO;
package REAL_IO is new FLOAT_IO(FLOAT);
use REAL_IO;
package COORD_IO is new INTEGER_IO(COORDINATE);
use COORD_IO;

SIZE: integer;
x1, x2, y1, y2: COORDINATE;

begin
  put("Enter the size of the image: ");
  get(SIZE);
  declare
    package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
    use IMAGES_N;

    type COMPLEX_IMAGE_PTR is access COMPLEX_IMAGES.IMAGE;
    SP : constant COMPLEX_IMAGE_PTR := new COMPLEX_IMAGES.IMAGE;
    S  : COMPLEX_IMAGES.IMAGE renames SP.all;

    package WRITE_COMPLEX_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                      SQUARE_COORDINATE, COMPLEX_NUMBER, COMPLEX_IMAGES.IMAGE);
    use WRITE_COMPLEX_IMAGE;

    ang, xc, yc: float;
    Z: COMPLEX_NUMBER;
    test: CHARACTER;

  begin
    for y in S'RANGE loop
      for x in S'RANGE loop
        S(y,x) := COMPLEX(0.0, 0.0);
      end loop;
    end loop;
    test := 'y';
    while test = 'y' loop
      put("SQUARE"); NEW_LINE;
      put("Start x = "); get(x1);
      put("End x =   "); get(x2);
      put("Start y = "); get(y1);
      put("End y =   "); get(y2);
      put("What angle is the colour? (0.0 .. 360.0) ");
      get(ang);
      if (ang <= 90.0) or (ang >= 270.0) then
        if (ang = 90.0) then
          xc := 0.0;
          yc := 220.0;
        elsif (ang = 270.0) then
          xc := 0.0;
          yc := -220.0;
        else
          xc := 1.0;
          yc := TAN(ang,360.0);
        end if;
```

```
      else
        xc := -1.0;
        yc := -TAN(ang,360.0);
      end if;

Z := 255.0*COMPLEX(xc, yc);
      for y in y1 .. y2 loop
        for x in x1 .. x2 loop
          S(y,x) := Z;
        end loop;
      end loop;
      put("Another Square? y or n ");
      get(test);
    end loop;
    WRITE_BINARY_IMAGE_FILE(SYSTEM_ENVIRONMENT.arg_value(1), S);
  end;
end colmanysquare;
```

COLTOAB.ADA

```
with TEXT_IO, BINARY_IMAGE_IO, SYSTEM_ENVIRONMENT, ELEMENTARY_FUNCTIONS,
     SIZE_OF_IMAGE, SIZE_OF_FILE, PIXEL_TYPES, COMMON_IMAGE_TYPES,
     RGB_CONVERSIONS, COMPLEX_NUMBERS_FLOAT;
use  TEXT_IO, PIXEL_TYPES, COMPLEX_NUMBERS_FLOAT;

-- Converts a colour binary file to the Z format.
-- Author: Amy Thornton

procedure coltoab is

  COLFILE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
  COMPFILE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);

  SIZE : constant NATURAL := 2**SIZE_OF_IMAGE(SIZE_OF_FILE(COLFILE)/3);

  package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
  use IMAGES_N;

  type COLOUR_IMAGE is array(SQUARE_COORDINATE, SQUARE_COORDINATE) of
                                                                COLOUR_PIXEL;
  pragma PACK(COLOUR_IMAGE);
  ICol : COLOUR_IMAGE;

  type COMPLEX_IMAGE_PTR is access COMPLEX_IMAGES.IMAGE;
  SP : constant COMPLEX_IMAGE_PTR := new COMPLEX_IMAGES.IMAGE;
  S  : COMPLEX_IMAGES.IMAGE renames SP.all;

  package READ_COL_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                SQUARE_COORDINATE, COLOUR_PIXEL, COLOUR_IMAGE);
  package WRITE_COMPLEX_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                      SQUARE_COORDINATE, COMPLEX_NUMBER, COMPLEX_IMAGES.IMAGE);

begin
  READ_COL_IMAGE.READ_BINARY_IMAGE_FILE(COLFILE, ICol);
  for y in ICol'RANGE(1) loop
    for x in ICol'RANGE(2) loop
      S(y,x) := RGB_CONVERSIONS.Z(ICol(y,x));
    end loop;
  end loop;
  WRITE_COMPLEX_IMAGE.WRITE_BINARY_IMAGE_FILE(COMPFILE, S);
end coltoab;
```

COLTOHUE.ADA

```
with TEXT_IO, BINARY_IMAGE_IO, SYSTEM_ENVIRONMENT, ELEMENTARY_FUNCTIONS,
     SIZE_OF_IMAGE, SIZE_OF_FILE, PIXEL_TYPES, COMMON_IMAGE_TYPES,
     RGB_CONVERSIONS;
use  TEXT_IO, PIXEL_TYPES;

-- Converts a colour binary file to a hue file.
-- Author: Amy Thornton

procedure coltohue is

   COLIMAGE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
   HUEIMAGE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);

  SIZE : constant NATURAL := 2**SIZE_OF_IMAGE(SIZE_OF_FILE(COLIMAGE)/3);

  package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
  use IMAGES_N;

  type COLOUR_IMAGE is array(SQUARE_COORDINATE, SQUARE_COORDINATE) of
                                                                COLOUR_PIXEL;
  pragma PACK(COLOUR_IMAGE);
  ICol : COLOUR_IMAGE;

  type REAL_IMAGE_PTR is access FLOAT_IMAGES.IMAGE;
  RPHue : constant REAL_IMAGE_PTR := new FLOAT_IMAGES.IMAGE;
  RHue : FLOAT_IMAGES.IMAGE renames RPHue.all;

  package READ_COL_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                SQUARE_COORDINATE, COLOUR_PIXEL, COLOUR_IMAGE);
  package WRITE_FLOAT_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                 SQUARE_COORDINATE, FLOAT, FLOAT_IMAGES.IMAGE);

begin
  READ_COL_IMAGE.READ_BINARY_IMAGE_FILE(COLIMAGE, ICol);
  for y in ICol'RANGE(1) loop
    for x in ICol'RANGE(2) loop
      begin
        RHue(y,x) := RGB_CONVERSIONS.HUE(ICol(y,x));
      exception
        when RGB_CONVERSIONS.INDETERMINATE_HUE => RHue(y,x) := 0.0;
      end;
    end loop;
  end loop;
  WRITE_FLOAT_IMAGE.WRITE_BINARY_IMAGE_FILE(HUEIMAGE, RHue);
end coltohue;
```

COLTOINT.ADA

```
with TEXT_IO, BINARY_IMAGE_IO, SYSTEM_ENVIRONMENT, ELEMENTARY_FUNCTIONS,
     SIZE_OF_IMAGE, SIZE_OF_FILE, PIXEL_TYPES, COMMON_IMAGE_TYPES,
     RGB_CONVERSIONS;
use  TEXT_IO, PIXEL_TYPES;

-- Converts a colour binary file to an intensity (I) file.
-- Author: Amy Thornton

procedure coltoint is

   COLIMAGE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
   INTIMAGE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);

  SIZE : constant NATURAL := 2**SIZE_OF_IMAGE(SIZE_OF_FILE(COLIMAGE)/3);

  package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
  use IMAGES_N;

  type COLOUR_IMAGE is array(SQUARE_COORDINATE, SQUARE_COORDINATE) of
                                                                COLOUR_PIXEL;
  pragma PACK(COLOUR_IMAGE);
  ICol : COLOUR_IMAGE;

  type REAL_IMAGE_PTR is access FLOAT_IMAGES.IMAGE;
  RPInt : constant REAL_IMAGE_PTR := new FLOAT_IMAGES.IMAGE;
  RInt : FLOAT_IMAGES.IMAGE renames RPInt.all;

  package READ_COL_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                SQUARE_COORDINATE, COLOUR_PIXEL, COLOUR_IMAGE);
  package WRITE_FLOAT_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                 SQUARE_COORDINATE, FLOAT, FLOAT_IMAGES.IMAGE);

begin
  READ_COL_IMAGE.READ_BINARY_IMAGE_FILE(COLIMAGE, ICol);
  for y in ICol'RANGE(1) loop
    for x in ICol'RANGE(2) loop
      RInt(y,x) := RGB_CONVERSIONS.INTENSITY(ICol(y,x));
    end loop;
  end loop;
  WRITE_FLOAT_IMAGE.WRITE_BINARY_IMAGE_FILE(INTIMAGE, RInt);
end coltoint;
```

COMPLOGBMAPFILTERLUTLPF3.ADA

```
with TEXT_IO, SYSTEM_ENVIRONMENT, BINARY_IMAGE_IO, COMPLEX_NUMBERS_FLOAT,
     SIZE_OF_IMAGE, PIXEL_TYPES, SIZE_OF_FILE, ELEMENTARY_FUNCTIONS,
     COMMON_IMAGE_TYPES, IMAGE_COORDINATES;
use  TEXT_IO, ELEMENTARY_FUNCTIONS, IMAGE_COORDINATES, PIXEL_TYPES;

-- Reads image, circumference oversampling LUT and vertically filters the
sampled image.
-- Author: Amy Thornton

procedure complogbmapfilterlutlpf3 is

  BINFILE  : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
  LUTFILE  : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);
  BINOUTFILE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(3);

  SIZE : constant NATURAL := 2**SIZE_OF_IMAGE(SIZE_OF_FILE(BINFILE));

  package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
  use IMAGES_N;

  type IMAGE_PTR is access PIXEL_IMAGES.IMAGE;
  I1P : constant IMAGE_PTR := new PIXEL_IMAGES.IMAGE;
  I1  : PIXEL_IMAGES.IMAGE renames I1P.all;

  package READWRITE_BINARY_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                  SQUARE_COORDINATE, PIXEL, PIXEL_IMAGES.IMAGE);

  package Nat_IO is new INTEGER_IO(Natural); use Nat_IO;
  Xout, Yout: natural;

begin
  put("Output Y size? "); get(Yout); put("Output X size? "); get(Xout);
  declare
    subtype xmapsize is natural range 0 .. Xout-1;
    subtype ymapsize is natural range 0 .. Yout-1;
    type IMAGEOUT is array(ymapsize, xmapsize) of PIXEL;
    I2, I3 : IMAGEOUT;
    package READWRITE_MAPBINARY_IMAGE is new BINARY_IMAGE_IO(ymapsize,
                                                    xmapsize, PIXEL, IMAGEOUT);
    type IMAGECOORD is
      record
        YY: natural;
        XX: natural;
      end record;

    type IMAGELUT is array(ymapsize, xmapsize) of IMAGECOORD;
    I : IMAGELUT;

    package READWRITE_LUTBINARY_IMAGE is new BINARY_IMAGE_IO(ymapsize,
                                             xmapsize, IMAGECOORD, IMAGELUT);

  procedure LPF(filtin: in IMAGEOUT; filtout: out IMAGEOUT) is

  begin
    for y in filtout'RANGE(1) loop
      for x in filtout'RANGE(2) loop
        begin
          filtout(y,x) := PIXEL(float(filtin(y-1,x) + filtin(y,x) +
                                                          filtin(y+1,x))/3.0);
        exception
          when CONSTRAINT_ERROR =>
            if y = 0 then
              filtout(y,x) := PIXEL(float(filtin((Yout-1),x) + filtin(y,x) +
                                                          filtin(y+1,x))/3.0);
            end if;
            if y = (Yout-1) then
```

```
              filtout(y,x) := PIXEL(float(filtin(y-1,x) + filtin(y,x) +
                                                          filtin(0,x))/3.0);
            end if;
          end;
        end loop;
      end loop;
    end LPF;

    begin
      READWRITE_BINARY_IMAGE.READ_BINARY_IMAGE_FILE(BINFILE, I1);
      READWRITE_LUTBINARY_IMAGE.READ_BINARY_IMAGE_FILE(LUTFILE, I);
      for y in I2'RANGE(1) loop
        for x in I2'RANGE(2) loop
          I2(y,x) := I1(SQUARE_COORDINATE(I(y,x).YY),
SQUARE_COORDINATE(I(y,x).XX));
        end loop;
      end loop;
      LPF(I2, I3);
      READWRITE_MAPBINARY_IMAGE.WRITE_BINARY_IMAGE_FILE(BINOUTFILE,I3);
    end;
end complogbmapfilterlutlpf3;
```

COMPLOGBMAPFILTERLUTTO256.ADA

```
with TEXT_IO, SYSTEM_ENVIRONMENT, BINARY_IMAGE_IO, COMPLEX_NUMBERS_FLOAT,
     SIZE_OF_IMAGE, PIXEL_TYPES, SIZE_OF_FILE, ELEMENTARY_FUNCTIONS,
     COMMON_IMAGE_TYPES, IMAGE_COORDINATES;
use  TEXT_IO, ELEMENTARY_FUNCTIONS, IMAGE_COORDINATES, PIXEL_TYPES;

-- Downsamples the vertically filtered image to obtain a square image.
-- Author: Amy Thornton

procedure complogbmapfilterlutto256 is

  BINFILE  : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
  BINOUTFILE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);

  package Nat_IO is new INTEGER_IO(Natural); use Nat_IO;
  pi   : constant float := 3.141592654;
  Xin, Yin, ang, SIZE: natural;

begin
  put("Input X size? "); get(Xin); put("Input Y size? "); get(Yin);
  put("Output x,y size? "); get(SIZE);
  declare
    package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
    use IMAGES_N;

    type IMAGE_PTR is access PIXEL_IMAGES.IMAGE;
    I256P : constant IMAGE_PTR := new PIXEL_IMAGES.IMAGE;
    I256  : PIXEL_IMAGES.IMAGE renames I256P.all;

    package READWRITE_BINARY_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                SQUARE_COORDINATE, PIXEL, PIXEL_IMAGES.IMAGE);

    subtype xmapsize is natural range 0 .. Xin-1;
    subtype ymapsize is natural range 0 .. Yin-1;
    type IMAGEOUT is array(ymapsize, xmapsize) of PIXEL;
    Iin : IMAGEOUT;
    package READWRITE_MAPBINARY_IMAGE is new
                           BINARY_IMAGE_IO(ymapsize,xmapsize, PIXEL, IMAGEOUT);
  begin
    READWRITE_MAPBINARY_IMAGE.READ_BINARY_IMAGE_FILE(BINFILE, Iin);
    for y in I256'RANGE(1) loop
      ang := natural(pi*float(y));
      for x in I256'RANGE(2) loop
        I256(y,x) := Iin(ang, natural(x));
      end loop;
    end loop;
    READWRITE_BINARY_IMAGE.WRITE_BINARY_IMAGE_FILE(BINOUTFILE,I256);
  end;
end complogbmapfilterlutto256;
```

COMPLOGBMAPFILTERMKLUT.ADA

```
with TEXT_IO, SYSTEM_ENVIRONMENT, BINARY_IMAGE_IO, COMPLEX_NUMBERS_FLOAT,
     SIZE_OF_IMAGE, PIXEL_TYPES, SIZE_OF_FILE, ELEMENTARY_FUNCTIONS,
     COMMON_IMAGE_TYPES, IMAGE_COORDINATES;
use  TEXT_IO, ELEMENTARY_FUNCTIONS, IMAGE_COORDINATES, PIXEL_TYPES;

-- Generates the circumference oversampling LUT for log-polar mapping.
-- Author: Amy Thornton

procedure complogbmapfiltermklut is

  BINFILE  : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
  package Nat_IO is new INTEGER_IO(Natural); use Nat_IO;
  pi   : constant float := 3.141592654;
  NSIZE: natural;

begin
  put("N size? "); get(NSIZE);
  declare
    xcentre  : constant float := float(NSIZE/2);
    ycentre  : constant float := float(NSIZE/2);
    xnew, ynew, radius, ang : float;
    mappedysize: constant natural := natural(2.0*pi*ycentre);
    mappedxsize: constant natural := NSIZE-1;
    subtype xmapsize is natural range 0 .. mappedxsize;
    subtype ymapsize is natural range 0 .. mappedysize;
    type IMAGEMAP is array(ymapsize, xmapsize) of PIXEL;
    IM: IMAGEMAP;
    package READWRITE_MAPBINARY_IMAGE is new BINARY_IMAGE_IO(ymapsize,
                                                  xmapsize, PIXEL, IMAGEMAP);
    type LUTCOORD is
      record
        YY: ymapsize;
        XX: xmapsize;
      end record;
    type LUT is array(ymapsize, xmapsize) of LUTCOORD;
    I : LUT;

    package READWRITE_BINARY_IMAGE is new BINARY_IMAGE_IO(ymapsize, xmapsize,
                                                        LUTCOORD, LUT);
    angconst: constant float := 360.0/(2.0*pi*xcentre);
    x, y: natural;

   begin
     put("LUT y size: "); put((mappedysize+1), width=>4);
     put("   LUT x size: "); put((mappedxsize+1), width=>4); NEW_LINE;
     for v in I'RANGE(1) loop
      ang := angconst*float(v);
      for u in I'RANGE(2) loop
        radius := float((NSIZE/2))**(float(u)/float(NSIZE-1));
        x := natural(xcentre + radius*cos(ang,360.0));
        y := natural(ycentre - radius*sin(ang,360.0));
        if (x < 0) then
          x := 0;
        elsif (x > (NSIZE-1)) then
          x := NSIZE-1;
        end if;
        if (y < 0) then
          y := 0;
        elsif (y > (NSIZE-1)) then
          y := NSIZE-1;
        end if;
        I(v,u).YY := y;
        I(v,u).XX := x;
      end loop;
    end loop;
    READWRITE_BINARY_IMAGE.WRITE_BINARY_IMAGE_FILE(BINFILE,I);
end;
end complogbmapfiltermklut;
```

COMPLOGBMAPFILTERRADLUTLPF3.ADA

```
with TEXT_IO, SYSTEM_ENVIRONMENT, BINARY_IMAGE_IO, COMPLEX_NUMBERS_FLOAT,
     SIZE_OF_IMAGE, PIXEL_TYPES, SIZE_OF_FILE, ELEMENTARY_FUNCTIONS,
     COMMON_IMAGE_TYPES, IMAGE_COORDINATES;
use  TEXT_IO, ELEMENTARY_FUNCTIONS, IMAGE_COORDINATES, PIXEL_TYPES;

-- Reads image, radial oversampling LUT and horizontally filters the sampled
image.
-- Author: Amy Thornton

procedure complogbmapfilterradlutlpf3 is

  BINFILE  : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
  LUTFILE  : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);
  BINOUTFILE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(3);

  SIZE : constant NATURAL := 2**SIZE_OF_IMAGE(SIZE_OF_FILE(BINFILE));

  package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
  use IMAGES_N;

  type IMAGE_PTR is access PIXEL_IMAGES.IMAGE;
  I1P : constant IMAGE_PTR := new PIXEL_IMAGES.IMAGE;
  I1  : PIXEL_IMAGES.IMAGE renames I1P.all;

  package READWRITE_BINARY_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                  SQUARE_COORDINATE, PIXEL, PIXEL_IMAGES.IMAGE);

  package Nat_IO is new INTEGER_IO(Natural); use Nat_IO;
  Xout, Yout: natural;

begin
  put("Output Y size? "); get(Yout); put("Output X size? "); get(Xout);
  declare
    subtype xmapsize is natural range 0 .. Xout-1;
    subtype ymapsize is natural range 0 .. Yout-1;
    type IMAGEOUT is array(ymapsize, xmapsize) of PIXEL;
    I2, I3 : IMAGEOUT;
    package READWRITE_MAPBINARY_IMAGE is new BINARY_IMAGE_IO(ymapsize,
                                                    xmapsize, PIXEL, IMAGEOUT);
    type IMAGECOORD is
      record
        YY: natural;
        XX: natural;
      end record;

    type IMAGELUT is array(ymapsize, xmapsize) of IMAGECOORD;
    I : IMAGELUT;

    package READWRITE_LUTBINARY_IMAGE is new BINARY_IMAGE_IO(ymapsize,
                                             xmapsize, IMAGECOORD, IMAGELUT);

  procedure LPF(filtin: in IMAGEOUT; filtout: out IMAGEOUT) is

  begin
    for y in filtout'RANGE(1) loop
      for x in filtout'RANGE(2) loop
        begin
          filtout(y,x) := PIXEL(float(filtin(y,x-1) + filtin(y,x) +
                                                          filtin(y,x+1))/3.0);
        exception
          when CONSTRAINT_ERROR =>
            if x = 0 then
              filtout(y,x) := PIXEL(float(filtin(y,(Xout-1)) + filtin(y,x) +
                                                          filtin(y,x+1))/3.0);
            end if;
            if x = (Xout-1) then
```

```
              filtout(y,x) := PIXEL(float(filtin(y,x-1) + filtin(y,x) +
                                                          filtin(y,0))/3.0);
            end if;
          end;
        end loop;
      end loop;
    end LPF;

    begin
      READWRITE_BINARY_IMAGE.READ_BINARY_IMAGE_FILE(BINFILE, I1);
      READWRITE_LUTBINARY_IMAGE.READ_BINARY_IMAGE_FILE(LUTFILE, I);
      for y in I2'RANGE(1) loop
        for x in I2'RANGE(2) loop
          I2(y,x) := I1(SQUARE_COORDINATE(I(y,x).YY),
SQUARE_COORDINATE(I(y,x).XX));
        end loop;
      end loop;
      LPF(I2, I3);
      READWRITE_MAPBINARY_IMAGE.WRITE_BINARY_IMAGE_FILE(BINOUTFILE,I3);
    end;
end complogbmapfilterradlutlpf3;
```

COMPLOGBMAPFILTERRADLUTTO256.ADA

```
with TEXT_IO, SYSTEM_ENVIRONMENT, BINARY_IMAGE_IO, COMPLEX_NUMBERS_FLOAT,
     SIZE_OF_IMAGE, PIXEL_TYPES, SIZE_OF_FILE, ELEMENTARY_FUNCTIONS,
     COMMON_IMAGE_TYPES, IMAGE_COORDINATES;
use  TEXT_IO, ELEMENTARY_FUNCTIONS, IMAGE_COORDINATES, PIXEL_TYPES;

-- Downsamples the horizontally filtered image to obtain a square image.
-- Author: Amy Thornton

procedure complogbmapfilterradlutto256 is

  BINFILE  : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
  BINOUTFILE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);

  package Nat_IO is new INTEGER_IO(Natural); use Nat_IO;
  Xin, Yin, SIZE: natural;

begin
  put("Input X size? "); get(Xin); put("Input Y size? "); get(Yin);
  put("Output x,y size? "); get(SIZE);
  declare
    package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
    use IMAGES_N;

    type IMAGE_PTR is access PIXEL_IMAGES.IMAGE;
    I256P : constant IMAGE_PTR := new PIXEL_IMAGES.IMAGE;
    I256  : PIXEL_IMAGES.IMAGE renames I256P.all;

    package READWRITE_BINARY_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                SQUARE_COORDINATE, PIXEL, PIXEL_IMAGES.IMAGE);

    subtype xmapsize is natural range 0 .. Xin-1;
    subtype ymapsize is natural range 0 .. Yin-1;
    type IMAGEOUT is array(ymapsize, xmapsize) of PIXEL;
    Iin : IMAGEOUT;
    package READWRITE_MAPBINARY_IMAGE is new BINARY_IMAGE_IO(ymapsize,
                                                    xmapsize, PIXEL, IMAGEOUT);
    factor : constant float := float((Xin-1))/float((256-1));

  begin
    READWRITE_MAPBINARY_IMAGE.READ_BINARY_IMAGE_FILE(BINFILE, Iin);
    for y in I256'RANGE(1) loop
      for x in I256'RANGE(2) loop
        I256(y,x) := Iin(natural(y), natural(float(x)*factor));
      end loop;
    end loop;
    READWRITE_BINARY_IMAGE.WRITE_BINARY_IMAGE_FILE(BINOUTFILE,I256);
  end;
end complogbmapfilterradlutto256;
```

COMPLOGBMAPFILTERRADMKLUT.ADA

```
with TEXT_IO, SYSTEM_ENVIRONMENT, BINARY_IMAGE_IO, COMPLEX_NUMBERS_FLOAT,
     SIZE_OF_IMAGE, PIXEL_TYPES, SIZE_OF_FILE, ELEMENTARY_FUNCTIONS,
     COMMON_IMAGE_TYPES, IMAGE_COORDINATES;
use  TEXT_IO, ELEMENTARY_FUNCTIONS, IMAGE_COORDINATES, PIXEL_TYPES;

-- Generates the radial oversampling LUT for log-polar mapping.
-- Author: Amy Thornton

procedure complogbmapfilterradmklut is

  BINFILE  : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
  package Int_IO is new Integer_IO(INTEGER); use Int_IO;
  package Coord_IO is new Integer_IO(COORDINATE); use Coord_IO;
  NSIZE: natural;

begin
  put("Size of image which is to be sampled later (eg 256)? "); get(NSIZE);
  declare
    angconst : constant float := 360.0/float(NSIZE);
    xcentre  : constant float := float(NSIZE/2);
    ycentre  : constant float := float(NSIZE/2);
    samples  : constant natural := 618;
    xnew, ynew, radius, ang : float;
    mappedysize: constant natural := NSIZE-1;
    mappedxsize: constant natural := samples-1;
    subtype xmapsize is natural range 0 .. mappedxsize;
    subtype ymapsize is natural range 0 .. mappedysize;
    type IMAGEMAP is array(ymapsize, xmapsize) of PIXEL;
    IM: IMAGEMAP;
    package READWRITE_MAPBINARY_IMAGE is new BINARY_IMAGE_IO(ymapsize,
                                                    xmapsize, PIXEL, IMAGEMAP);
    type LUTCOORD is
      record
        YY: natural;
        XX: natural;
      end record;
    type LUT is array(ymapsize, xmapsize) of LUTCOORD;
    I : LUT;

    package READWRITE_BINARY_IMAGE is new BINARY_IMAGE_IO(ymapsize, xmapsize,
                                                          LUTCOORD, LUT);
    x, y: natural;

   begin
     for v in I'RANGE(1) loop
      ang := angconst*float(v);
      for u in I'RANGE(2) loop
        radius := float((NSIZE/2))**(float(u)/float(samples-1));
        x := natural(xcentre + radius*cos(ang,360.0));
        y := natural(ycentre - radius*sin(ang,360.0));
        if (x < 0) then
          x := 0;
        elsif (x > (NSIZE-1)) then
          x := NSIZE-1;
        end if;
        if (y < 0) then
          y := 0;
        elsif (y > (NSIZE-1)) then
          y := NSIZE-1;
        end if;
        I(v,u).YY := y;
        I(v,u).XX := x;
      end loop;
    end loop;
    READWRITE_BINARY_IMAGE.WRITE_BINARY_IMAGE_FILE(BINFILE,I);
  end;
end complogbmapfilterradmklut;
```

COMPLOGBVHLUTLPFBILININT.ADA

```
with TEXT_IO, SYSTEM_ENVIRONMENT, BINARY_IMAGE_IO, COMPLEX_NUMBERS_FLOAT,
     SIZE_OF_IMAGE, PIXEL_TYPES, SIZE_OF_FILE, ELEMENTARY_FUNCTIONS,
     COMMON_IMAGE_TYPES, IMAGE_COORDINATES;
use  TEXT_IO, ELEMENTARY_FUNCTIONS, IMAGE_COORDINATES, PIXEL_TYPES;

-- Reads image, circumferential and radial oversampling LUT and horizontally
filters the sampled image using bilinear interpolation.
-- Author: Amy Thornton

procedure complogbvhlutlpfbilinint is

  BINFILE  : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
  LUTFILE  : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);
  BINOUTFILE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(3);

  SIZE : constant NATURAL := 2**SIZE_OF_IMAGE(SIZE_OF_FILE(BINFILE));
  SIZEF : constant FLOAT := float(SIZE);

  package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
  use IMAGES_N;

  type IMAGE_PTR is access PIXEL_IMAGES.IMAGE;
  I1P : constant IMAGE_PTR := new PIXEL_IMAGES.IMAGE;
  I1  : PIXEL_IMAGES.IMAGE renames I1P.all;

  package READWRITE_BINARY_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                  SQUARE_COORDINATE, PIXEL, PIXEL_IMAGES.IMAGE);

  package Nat_IO is new INTEGER_IO(Natural); use Nat_IO;
  Xout, Yout, filtersize: natural;
  pix: PIXEL;

begin
  put("Output Y size? "); get(Yout); put("Output X size? "); get(Xout);
  put("Filtersize? "); get(filtersize);
  declare
    offset : constant natural := (filtersize-1)/2;
    subtype xmapsize is natural range 0 .. Xout-1;
    subtype ymapsize is natural range 0 .. Yout-1;
    type IMAGEOUT is array(ymapsize, xmapsize) of PIXEL;
    I2, I3 : IMAGEOUT;
    package READWRITE_MAPBINARY_IMAGE is new
                           BINARY_IMAGE_IO(ymapsize,xmapsize, PIXEL, IMAGEOUT);
    type IMAGECOORD is
      record
        YY: float;
        XX: float;
      end record;
    type IMAGELUT is array(ymapsize, xmapsize) of IMAGECOORD;
    I : IMAGELUT;

    package READWRITE_LUTBINARY_IMAGE is new BINARY_IMAGE_IO(ymapsize,
                                                 xmapsize, IMAGECOORD, IMAGELUT);

    procedure BILININT(x, y: in float; I: in PIXEL_IMAGES.IMAGE; pix: out
PIXEL) is
      x1, x2, y1, y2: float;
      x1c, x2c, y1c, y2c: SQUARE_COORDINATE;
    begin
      x1 := float(integer(x-0.5));
      if x1 = -1.0 then x1 := 0.0; end if;
      x2 := x1 + 1.0;
      y1 := float(integer(y-0.5));
      if y1 = -1.0 then y1 := 0.0; end if;
      y2 := y1 + 1.0;
      begin
```

```
        x1c := SQUARE_COORDINATE(x1); x2c := SQUARE_COORDINATE(x2);
      exception
      when CONSTRAINT_ERROR =>
        if x2 = SIZEF then
          x2c := x1c;
        end if;
      end;
      begin
        y1c := SQUARE_COORDINATE(y1); y2c := SQUARE_COORDINATE(y2);
      exception
      when CONSTRAINT_ERROR =>
        if y2 = SIZEF then
          y2c := y1c;
        end if;
      end;
      pix := PIXEL((1.0-x+x1)*(1.0-y+y1)*float(I(y1c,x1c)) +
                       (x-x1)*(1.0-y+y1)*float(I(y1c,x2c)) +
                       (1.0-x+x1)*(y-y1)*float(I(y2c,x1c)) +
                           (x-x1)*(y-y1)*float(I(y2c,x2c)));
    end BILININT;

    procedure LPF(filtin: in IMAGEOUT; filtout: out IMAGEOUT) is

      xint: integer;
      xmod: xmapsize;
      ymod: ymapsize;
      tmp: natural;

    begin
      for y in filtout'RANGE(1) loop
        for x in filtout'RANGE(2) loop
          tmp := 0;
          for yf in 0 .. filtersize-1 loop
            for xf in 0 .. filtersize-1 loop
              begin
                xint := integer(x-offset+xf);
                xmod := xmapsize(xint);
                ymod := (y-offset+yf) mod Yout;
                tmp := natural(filtin(ymod,xmod)) + tmp;
              exception
                when CONSTRAINT_ERROR =>
                  if xint < 0 then
                    tmp := natural(filtin(ymod,xmapsize'FIRST)) + tmp;
                  elsif xint > xmapsize'LAST then
                    tmp := natural(filtin(ymod,xmapsize'LAST)) + tmp;
                  else
                    put("Error.");
                  end if;
              end;
            end loop;
          end loop;
          filtout(y,x) := pixel(tmp/(filtersize*filtersize));
        end loop;
      end loop;
    end LPF;

  begin
    READWRITE_BINARY_IMAGE.READ_BINARY_IMAGE_FILE(BINFILE, I1);
    READWRITE_LUTBINARY_IMAGE.READ_BINARY_IMAGE_FILE(LUTFILE, I);
    for y in I2'RANGE(1) loop
      for x in I2'RANGE(2) loop
        BILININT(I(y,x).XX, I(y,x).YY, I1, pix);
        I2(y,x) := pix;
      end loop;
    end loop;
    LPF(I2, I3);
    READWRITE_MAPBINARY_IMAGE.WRITE_BINARY_IMAGE_FILE(BINOUTFILE,I3);
  end;
end complogbvhlutlpfbilinint;
```

COMPLOGBVHLUTTO256.ADA

```
with TEXT_IO, SYSTEM_ENVIRONMENT, BINARY_IMAGE_IO, COMPLEX_NUMBERS_FLOAT,
     SIZE_OF_IMAGE, PIXEL_TYPES, SIZE_OF_FILE, ELEMENTARY_FUNCTIONS,
     COMMON_IMAGE_TYPES, IMAGE_COORDINATES;
use  TEXT_IO, ELEMENTARY_FUNCTIONS, IMAGE_COORDINATES, PIXEL_TYPES;

-- Downsamples the horizontally and vertically filtered image to obtain a
square image.
-- Author: Amy Thornton

procedure complogbvhlutto256 is

  BINFILE  : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
  BINOUTFILE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);

  package Nat_IO is new INTEGER_IO(Natural); use Nat_IO;
  pi   : constant float := 3.141592654;
  Xin, Yin, ang, SIZE: natural;

begin
  put("Input X size? "); get(Xin); put("Input Y size? "); get(Yin);
  put("Output x,y size? "); get(SIZE);
  declare
    package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
    use IMAGES_N;

    type IMAGE_PTR is access PIXEL_IMAGES.IMAGE;
    I256P : constant IMAGE_PTR := new PIXEL_IMAGES.IMAGE;
    I256  : PIXEL_IMAGES.IMAGE renames I256P.all;

    package READWRITE_BINARY_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                SQUARE_COORDINATE, PIXEL, PIXEL_IMAGES.IMAGE);

    subtype xmapsize is natural range 0 .. Xin-1;
    subtype ymapsize is natural range 0 .. Yin-1;
    type IMAGEOUT is array(ymapsize, xmapsize) of PIXEL;
    Iin : IMAGEOUT;
    package READWRITE_MAPBINARY_IMAGE is new
                           BINARY_IMAGE_IO(ymapsize,xmapsize, PIXEL, IMAGEOUT);
    factor : constant float := float((Xin-1))/float((256-1));

  begin
    READWRITE_MAPBINARY_IMAGE.READ_BINARY_IMAGE_FILE(BINFILE, Iin);
    for y in I256'RANGE(1) loop
      ang := natural(pi*float(y));
      for x in I256'RANGE(2) loop
        I256(y,x) := Iin(ang, natural(float(x)*factor));
      end loop;
    end loop;
    READWRITE_BINARY_IMAGE.WRITE_BINARY_IMAGE_FILE(BINOUTFILE,I256);
  end;
end complogbvhlutto256;
```

COMPLOGBVHMKLUT.ADA

```
with TEXT_IO, SYSTEM_ENVIRONMENT, BINARY_IMAGE_IO, COMPLEX_NUMBERS_FLOAT,
     SIZE_OF_IMAGE, PIXEL_TYPES, SIZE_OF_FILE, ELEMENTARY_FUNCTIONS,
     COMMON_IMAGE_TYPES, IMAGE_COORDINATES;
use  TEXT_IO, ELEMENTARY_FUNCTIONS, IMAGE_COORDINATES, PIXEL_TYPES;

-- Generates the circumferential and radial oversampling LUT for log-polar
mapping.
-- Author: Amy Thornton

procedure complogbvhmklut is

  BINFILE  : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);

  package Nat_IO is new INTEGER_IO(Natural); use Nat_IO;

  pi   : constant float := 3.141592654;
  NSIZE: natural;

begin
  put("Size of image which is to be sampled later (eg 256)? "); get(NSIZE);
  declare
    xcentre  : constant float := float(NSIZE/2);
    ycentre  : constant float := float(NSIZE/2);
    samples  : constant natural := 618;
    xnew, ynew, radius, ang : float;
    mappedysize: constant natural := natural(2.0*pi*ycentre);
    mappedxsize: constant natural := samples-1;
    subtype xmapsize is natural range 0 .. mappedxsize;
    subtype ymapsize is natural range 0 .. mappedysize;
    type IMAGEMAP is array(ymapsize, xmapsize) of PIXEL;
    IM: IMAGEMAP;
    package READWRITE_MAPBINARY_IMAGE is new BINARY_IMAGE_IO(ymapsize,
                                                    xmapsize, PIXEL, IMAGEMAP);
    type LUTCOORD is
      record
        YY: natural;
        XX: natural;
      end record;

    type LUT is array(ymapsize, xmapsize) of LUTCOORD;
    I : LUT;

    package READWRITE_BINARY_IMAGE is new BINARY_IMAGE_IO(ymapsize, xmapsize,
                                                          LUTCOORD, LUT);
    angconst: constant float := 360.0/(2.0*pi*xcentre);
    x, y: natural;

   begin
     put("LUT y size: "); put((mappedysize+1), width=>4);
     put("   LUT x size: "); put((mappedxsize+1), width=>4); NEW_LINE;
     for v in I'RANGE(1) loop
      ang := angconst*float(v);
      for u in I'RANGE(2) loop
        radius := float((NSIZE/2))**(float(u)/float(samples-1));
        x := natural(xcentre + radius*cos(ang,360.0));
        y := natural(ycentre - radius*sin(ang,360.0));
        if (x < 0) then
          x := 0;
        elsif (x > (NSIZE-1)) then
          x := NSIZE-1;
        end if;
        if (y < 0) then
          y := 0;
        elsif (y > (NSIZE-1)) then
          y := NSIZE-1;
        end if;
        I(v,u).YY := y;
```

```
        I(v,u).XX := x;
      end loop;
    end loop;
    READWRITE_BINARY_IMAGE.WRITE_BINARY_IMAGE_FILE(BINFILE,I);
  end;
end complogbvhmklut;
```

COMPTOPHASE.ADA

```
with TEXT_IO, BINARY_IMAGE_IO, SYSTEM_ENVIRONMENT, ELEMENTARY_FUNCTIONS,
     SIZE_OF_IMAGE, SIZE_OF_FILE, PIXEL_TYPES, COMPLEX_NUMBERS_FLOAT,
     COMMON_IMAGE_TYPES;
use  TEXT_IO, COMPLEX_NUMBERS_FLOAT;

-- Calculates the argument for each pixel in a complex image.
-- Author: Amy Thornton

procedure comptophase is

   COMPFILE  : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
   FLOATFILE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);

  SIZE : constant NATURAL := 2**SIZE_OF_IMAGE(SIZE_OF_FILE(COMPFILE)/
                                                    (2 * FLOAT'SIZE/8));

  package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
  use IMAGES_N;

  type COMPLEX_IMAGE_PTR is access COMPLEX_IMAGES.IMAGE;
  SP : constant COMPLEX_IMAGE_PTR := new COMPLEX_IMAGES.IMAGE;
  S  : COMPLEX_IMAGES.IMAGE renames SP.all;

  type REAL_IMAGE_PTR is access FLOAT_IMAGES.IMAGE;
  RP : constant    REAL_IMAGE_PTR := new FLOAT_IMAGES.IMAGE;
  R  : FLOAT_IMAGES.IMAGE renames RP.all;

  package READ_COMPLEX_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                       SQUARE_COORDINATE, COMPLEX_NUMBER, COMPLEX_IMAGES.IMAGE);
  package WRITE_FLOAT_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                SQUARE_COORDINATE, FLOAT, FLOAT_IMAGES.IMAGE);

  Re, Im : Float;

begin
  READ_COMPLEX_IMAGE.READ_BINARY_IMAGE_FILE(COMPFILE, S);
  for y in S'RANGE(1) loop
    for x in S'RANGE(2) loop
      Re := REAL(S(y,x));
      Im := IMAGINARY(S(y,x));
      begin
        R(y,x) := ELEMENTARY_FUNCTIONS.ARCTAN(Im, Re, 360.0);
        if R(y,x) < 0.0 then
          R(y,x) := 360.0 + R(y,x);
        end if;
      exception
        when CONSTRAINT_ERROR =>
          R(y,x) := 0.0;
        when ELEMENTARY_FUNCTIONS.ARGUMENT_ERROR =>
          if Im > 0.0 then
            R(y,x) := 90.0;
          else
            R(y,x) := 270.0;
          end if;
      end;
    end loop;
  end loop;
  WRITE_FLOAT_IMAGE.WRITE_BINARY_IMAGE_FILE(FLOATFILE, R);
end comptophase;
```

FFTC.ADA

```
with TEXT_IO, SYSTEM_ENVIRONMENT, BINARY_IMAGE_IO, ELEMENTARY_FUNCTIONS,
     SIZE_OF_IMAGE, SIZE_OF_FILE, COMPLEX_NUMBERS_FLOAT, PIXEL_TYPES,
     COMMON_IMAGE_TYPES, TWO_DIMENSIONAL_FOURIER_TRANSFORMS;
use  TEXT_IO, COMPLEX_NUMBERS_FLOAT, ELEMENTARY_FUNCTIONS;

-- Input complex image, FFT, output complex image
-- Author: Amy Thornton

procedure FFTC is

   INPUT_FILENAME : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
  OUTPUT_FILENAME : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);

  SIZE     : constant NATURAL :=
2**SIZE_OF_IMAGE(SIZE_OF_FILE(INPUT_FILENAME)/(2 * FLOAT'SIZE/8));

  package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
  use IMAGES_N;

  package FFT_N is new TWO_DIMENSIONAL_FOURIER_TRANSFORMS(FLOAT,
                                                          COMPLEX_NUMBER,
                                                          SQUARE_COORDINATE,
                                                          FLOAT_IMAGES.IMAGE,

COMPLEX_IMAGES.IMAGE);

  type COMPLEX_IMAGE_PTR is access COMPLEX_IMAGES.IMAGE;
  SP : constant COMPLEX_IMAGE_PTR := new COMPLEX_IMAGES.IMAGE;
  S  : COMPLEX_IMAGES.IMAGE renames SP.all;

  package READ_COMPLEX_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
             SQUARE_COORDINATE, COMPLEX_NUMBER, COMPLEX_IMAGES.IMAGE);
  use READ_COMPLEX_IMAGE;

  package WRITE_COMPLEX_IMAGE is new BINARY_IMAGE_IO
                                (SQUARE_COORDINATE, SQUARE_COORDINATE,
                                 COMPLEX_NUMBERS_FLOAT.COMPLEX_NUMBER,
                                                 COMPLEX_IMAGES.IMAGE);
  use WRITE_COMPLEX_IMAGE;

begin
  READ_COMPLEX_IMAGE.READ_BINARY_IMAGE_FILE(INPUT_FILENAME, S);
  FFT_N.FFT_BY_ROW_COLUMN_SEPARATION(S);
  WRITE_COMPLEX_IMAGE.WRITE_BINARY_IMAGE_FILE(OUTPUT_FILENAME, S);
end FFTC;
```

HUEIMAGE_SICONSTANT.ADA

```
with TEXT_IO, BINARY_IMAGE_IO, SYSTEM_ENVIRONMENT, ELEMENTARY_FUNCTIONS,
     SIZE_OF_IMAGE, SIZE_OF_FILE, PIXEL_TYPES, COMMON_IMAGE_TYPES,
     RGB_CONVERSIONS;
use  TEXT_IO, PIXEL_TYPES;

-- Reads a hue image, requests constant saturation and intensity values and
outputs a colour binary image.
-- Author: Amy Thornton

procedure hueimage_siconstant is

   HUEIMAGE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
   HSIIMAGE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);

  SIZE : constant NATURAL := 2**SIZE_OF_IMAGE(SIZE_OF_FILE(HUEIMAGE)/4);

  package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
  use IMAGES_N;

  type IMAGE is array(SQUARE_COORDINATE, SQUARE_COORDINATE) of PIXEL;
  pragma PACK(IMAGE);
  IR, IG, IB : IMAGE;

  type COLOUR_IMAGE is array(SQUARE_COORDINATE, SQUARE_COORDINATE) of
                                                                  COLOUR_PIXEL;
  pragma PACK(COLOUR_IMAGE);
  ICol : COLOUR_IMAGE;

  type REAL_IMAGE_PTR is access FLOAT_IMAGES.IMAGE;
  RPHue : constant REAL_IMAGE_PTR := new FLOAT_IMAGES.IMAGE;
  RHue : FLOAT_IMAGES.IMAGE renames RPHue.all;

  package READ_FLOAT_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                  SQUARE_COORDINATE, FLOAT, FLOAT_IMAGES.IMAGE);
  package WRITE_COL_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                 SQUARE_COORDINATE, COLOUR_PIXEL, COLOUR_IMAGE);

  package Real_IO is new Float_IO(FLOAT); use Real_IO;

  Sat, Int: Float;

begin
  READ_FLOAT_IMAGE.READ_BINARY_IMAGE_FILE(HUEIMAGE, RHue);
  put("Saturation = ? "); get(Sat);
  put("Intensity = ? "); get(Int);
  for y in RHue'RANGE(1) loop
    for x in RHue'RANGE(2) loop
      begin
        ICol(y,x) := RGB_CONVERSIONS.RGB(RHue(y,x), Sat, Int);
      exception
        when CONSTRAINT_ERROR => ICol(y,x) := (255, 255, 255);
      end;
    end loop;
  end loop;
  WRITE_COL_IMAGE.WRITE_BINARY_IMAGE_FILE(HSIIMAGE, ICol);
end hueimage_siconstant;
```

IABTOCOL.ADA

```
with TEXT_IO, BINARY_IMAGE_IO, SYSTEM_ENVIRONMENT, ELEMENTARY_FUNCTIONS,
     SIZE_OF_IMAGE, SIZE_OF_FILE, PIXEL_TYPES, COMMON_IMAGE_TYPES,
     RGB_CONVERSIONS, COMPLEX_NUMBERS_FLOAT, IMAGE_COORDINATES,
     INTEGER_MIN, INTEGER_MAX;
use  TEXT_IO, PIXEL_TYPES, COMPLEX_NUMBERS_FLOAT, ELEMENTARY_FUNCTIONS,
     IMAGE_COORDINATES;

-- Generates a colour binary image from IZ values.
-- Author: Amy Thornton

procedure iabtocol is

  COMPFILE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
  INTFILE  : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);
  COLFILE  : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(3);

  SIZE : constant NATURAL := 2**SIZE_OF_IMAGE(SIZE_OF_FILE(COMPFILE)/
                                                    (2 * FLOAT'SIZE/8));

  package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
  use IMAGES_N;

  type COMPLEX_IMAGE_PTR is access COMPLEX_IMAGES.IMAGE;
  SP : constant COMPLEX_IMAGE_PTR := new COMPLEX_IMAGES.IMAGE;
  S  : COMPLEX_IMAGES.IMAGE renames SP.all;

  type REAL_IMAGE_PTR is access FLOAT_IMAGES.IMAGE;
  RP : constant REAL_IMAGE_PTR := new FLOAT_IMAGES.IMAGE;
  R : FLOAT_IMAGES.IMAGE renames RP.all;

  type COLOUR_IMAGE is array(SQUARE_COORDINATE, SQUARE_COORDINATE) of
                                                          COLOUR_PIXEL;
  pragma PACK(COLOUR_IMAGE);
  ICol : COLOUR_IMAGE;

  package READ_COMPLEX_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                     SQUARE_COORDINATE, COMPLEX_NUMBER, COMPLEX_IMAGES.IMAGE);

  package READ_FLOAT_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                                SQUARE_COORDINATE, FLOAT, FLOAT_IMAGES.IMAGE);

  package WRITE_COL_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                               SQUARE_COORDINATE, COLOUR_PIXEL, COLOUR_IMAGE);

  OldRed, OldGreen, OldBlue, Minrgb, Maxrgb, Red, Green, Blue: integer;

  procedure CALC(OldMax, OldMin, OldMid: in integer; Max, Min, Mid: out
integer) is

  Tmpfloat: float;

  begin
    Max := 255;
    Min := 0;
    Tmpfloat := float(255*(OldMid - OldMin))/float(OldMax - OldMin);
    Mid := integer(Tmpfloat);
  end CALC;

begin
  READ_COMPLEX_IMAGE.READ_BINARY_IMAGE_FILE(COMPFILE, S);
  READ_FLOAT_IMAGE.READ_BINARY_IMAGE_FILE(INTFILE, R);
  for y in S'RANGE(1) loop
    for x in S'RANGE(2) loop
      begin
        ICol(y,x) := RGB_CONVERSIONS.RGB(R(y,x), S(y,x), FALSE);
      exception
        when RGB_CONVERSIONS.INVALID_IZ_COORDINATE =>
```

```
          OldRed   := RGB_CONVERSIONS.R;
          OldGreen := RGB_CONVERSIONS.G;
          OldBlue  := RGB_CONVERSIONS.B;
          Minrgb   := INTEGER_MIN(INTEGER_MIN(OldRed, OldGreen), OldBlue);
          Maxrgb  := INTEGER_MAX(INTEGER_MAX(OldRed, OldGreen), OldBlue);
          if ((Maxrgb - Minrgb) <= 255) then
            if Minrgb < 0 then
              Red   := OldRed - Minrgb;
              Green := OldGreen - Minrgb;
              Blue  := OldBlue - Minrgb;
            else
              Red   := OldRed + 255 - Maxrgb;
              Green := OldGreen + 255 - Maxrgb;
              Blue  := OldBlue + 255 - Maxrgb;
            end if;
          else
            if OldRed = Maxrgb then
              if OldBlue = Minrgb then
                CALC(Maxrgb, Minrgb, OldGreen, Red, Blue, Green); --RMax, BMin
              else
                CALC(Maxrgb, Minrgb, OldBlue, Red, Green, Blue);  --RMax, GMin
              end if;
            elsif OldGreen = Maxrgb then
              if OldRed = Minrgb then
                CALC(Maxrgb, Minrgb, OldBlue, Green, Red, Blue);  --GMax, RMin
              else
                CALC(Maxrgb, Minrgb, OldRed, Green, Blue, Red);   --GMax, BMin
              end if;
            else
              if OldRed = Minrgb then
                CALC(Maxrgb, Minrgb, OldGreen, Blue, Red, Green); --BMax, RMin
              else
                CALC(Maxrgb, Minrgb, OldRed, Blue, Green, Red);   --BMax, GMin
              end if;
            end if;
          end if;
          ICol(y,x) := (PIXEL(Red), PIXEL(Green), PIXEL(Blue));
        end;
      end loop;
    end loop;
    WRITE_COL_IMAGE.WRITE_BINARY_IMAGE_FILE(COLFILE, ICol);
end iabtocol;
```

IFFTC.ADA

```
with TEXT_IO, SYSTEM_ENVIRONMENT, BINARY_IMAGE_IO, COMPLEX_NUMBERS_FLOAT,
     FLOAT_MAX, FLOAT_MIN, IMAGE_COORDINATES, SIZE_OF_IMAGE, PIXEL_TYPES,
     SIZE_OF_FILE, ELEMENTARY_FUNCTIONS, COMMON_IMAGE_TYPES,
     TWO_DIMENSIONAL_FOURIER_TRANSFORMS, IMAGE_COORDINATES;
use  TEXT_IO, COMPLEX_NUMBERS_FLOAT, ELEMENTARY_FUNCTIONS, IMAGE_COORDINATES;

-- Calculates the inverse FFT of a complex input.
-- Author: Amy Thornton

procedure IFFTC is

   INPUT_FILENAME : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
  OUTPUT_FILENAME : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);
  N : constant NATURAL := 2**SIZE_OF_IMAGE(SIZE_OF_FILE(INPUT_FILENAME)/
                                                        (2 * FLOAT'SIZE/8));

  package IMAGES_N is new COMMON_IMAGE_TYPES(N);
  use IMAGES_N;

  package FFT_N is new TWO_DIMENSIONAL_FOURIER_TRANSFORMS(FLOAT,
                                                          COMPLEX_NUMBER,
                                                          SQUARE_COORDINATE,
                                                          FLOAT_IMAGES.IMAGE,
                                                       COMPLEX_IMAGES.IMAGE);

  type COMPLEX_IMAGE_PTR is access COMPLEX_IMAGES.IMAGE;
  SP : constant COMPLEX_IMAGE_PTR := new COMPLEX_IMAGES.IMAGE;
  S  : COMPLEX_IMAGES.IMAGE renames SP.all;

  package COMPLEX_IMAGE is new BINARY_IMAGE_IO (SQUARE_COORDINATE,
                       SQUARE_COORDINATE, COMPLEX_NUMBER, COMPLEX_IMAGES.IMAGE);
  use COMPLEX_IMAGE;

  package Real_IO is new Float_IO(FLOAT); use Real_IO;

  MN : constant FLOAT := FLOAT(N) * FLOAT(N);
  MAXI, MAXR : FLOAT;

begin
  COMPLEX_IMAGE.READ_BINARY_IMAGE_FILE(INPUT_FILENAME, S);
  for y in S'RANGE(1) loop
    for x in S'RANGE(2) loop
      S(y,x) := CONJUGATE(S(y,x))/MN;
    end loop;
  end loop;
  FFT_N.FFT_BY_ROW_COLUMN_SEPARATION(S);
  for y in S'RANGE(1) loop
    for x in S'RANGE(2) loop
      S(y,x) := CONJUGATE(S(y,x));
      if (y = SQUARE_COORDINATE'FIRST) and (x = SQUARE_COORDINATE'FIRST) then
        MAXI := IMAGINARY(S(y,x));
        MAXR := REAL(s(y,x));
      end if;
      if ELEMENTARY_FUNCTIONS.SQRT(IMAGINARY(s(y,x))**2) > MAXI then
        MAXI := IMAGINARY(s(y,x));
      end if;
      if ELEMENTARY_FUNCTIONS.SQRT(REAL(s(y,x))**2) > MAXR then
        MAXR := REAL(s(y,x));
      end if;
    end loop;
  end loop;
  put("Post-ifft, the max. mod. value of the imaginary part is: ");
  put(MAXI, Fore => 8, Aft => 8, Exp => 0); NEW_LINE;
  put("Post-ifft, the max. mod. value of the real part is:      ");
  put(MAXR,Fore => 8, Aft => 8, Exp => 0); NEW_LINE;
  COMPLEX_IMAGE.WRITE_BINARY_IMAGE_FILE(OUTPUT_FILENAME, S);
end IFFTC;
```

MAG.ADA

```
with TEXT_IO, BINARY_IMAGE_IO, SYSTEM_ENVIRONMENT, ELEMENTARY_FUNCTIONS,
     SIZE_OF_IMAGE, SIZE_OF_FILE, PIXEL_TYPES, COMPLEX_NUMBERS_FLOAT,
     COMMON_IMAGE_TYPES;
use TEXT_IO, COMPLEX_NUMBERS_FLOAT;

-- Reads in a complex FFT image and alters the magnitude of the image
-- without altering the phase
-- Author: Amy Thornton

procedure MAG is

   INPUT_FILENAME : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
  OUTPUT_FILENAME : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);

  SIZE : constant NATURAL := 2**SIZE_OF_IMAGE(SIZE_OF_FILE(INPUT_FILENAME)/
                                                    (2 * FLOAT'SIZE/8));

  package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
  use IMAGES_N;

  type COMPLEX_IMAGE_PTR is access COMPLEX_IMAGES.IMAGE;
  SP : constant COMPLEX_IMAGE_PTR := new COMPLEX_IMAGES.IMAGE;
  S  : COMPLEX_IMAGES.IMAGE renames SP.all;

  package READ_COMPLEX_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                    SQUARE_COORDINATE, COMPLEX_NUMBER, COMPLEX_IMAGES.IMAGE);
  package WRITE_COMPLEX_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                    SQUARE_COORDINATE, COMPLEX_NUMBER, COMPLEX_IMAGES.IMAGE);

  package Real_IO is new Float_IO(FLOAT); use Real_IO;
  package Int_IO is new INTEGER_IO(INTEGER); use Int_IO;

  NewMod, OldMod: Float;

begin
  put("Image modulus is:      "); get(NewMod);
  READ_COMPLEX_IMAGE.READ_BINARY_IMAGE_FILE(INPUT_FILENAME, S);
  for y in S'RANGE(1) loop
    for x in S'RANGE(2) loop
      OldMod := ELEMENTARY_FUNCTIONS.SQRT(REAL(S(y,x))**2 +
                                                  IMAGINARY(S(y,x))**2);
      if OldMod /= 0.0 then
        S(y,x) := S(y,x)*NewMod/OldMod;
      else
        S(y,x) := COMPLEX(0.0, 0.0);
      end if;
    end loop;
  end loop;
  WRITE_COMPLEX_IMAGE.WRITE_BINARY_IMAGE_FILE(OUTPUT_FILENAME, S);
end MAG;
```

PCTIMESALPHA.ADA

```
with TEXT_IO, SYSTEM_ENVIRONMENT, BINARY_IMAGE_IO, ELEMENTARY_FUNCTIONS,
     SIZE_OF_IMAGE, SIZE_OF_FILE, COMPLEX_NUMBERS_FLOAT, PIXEL_TYPES,
     COMMON_IMAGE_TYPES, TWO_DIMENSIONAL_FOURIER_TRANSFORMS;
use  TEXT_IO, COMPLEX_NUMBERS_FLOAT, ELEMENTARY_FUNCTIONS;

--  Calculates  the  correlation  of  two  images.  Allows  variation  of  the
normalisation value.
-- Author: Amy Thornton

procedure pctimesalpha is

   INPUT1_FILENAME : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
   INPUT2_FILENAME : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);
   PC_FILENAME     : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(3);
   OUTPUT_FILENAME : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(4);

  SIZE : constant NATURAL := 2**SIZE_OF_IMAGE(SIZE_OF_FILE(INPUT1_FILENAME)/
                                                        (2 * FLOAT'SIZE/8));

  package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
  use IMAGES_N;

  package FFT_N is new TWO_DIMENSIONAL_FOURIER_TRANSFORMS(FLOAT,
                                                          COMPLEX_NUMBER,
                                                          SQUARE_COORDINATE,
                                                          FLOAT_IMAGES.IMAGE,
                                                       COMPLEX_IMAGES.IMAGE);

  type COMPLEX_IMAGE_PTR is access COMPLEX_IMAGES.IMAGE;
  S1P : constant COMPLEX_IMAGE_PTR := new COMPLEX_IMAGES.IMAGE;
  S1  : COMPLEX_IMAGES.IMAGE renames S1P.all;
  S2P : constant COMPLEX_IMAGE_PTR := new COMPLEX_IMAGES.IMAGE;
  S2  : COMPLEX_IMAGES.IMAGE renames S2P.all;
  S3P : constant COMPLEX_IMAGE_PTR := new COMPLEX_IMAGES.IMAGE;
  S3  : COMPLEX_IMAGES.IMAGE renames S3P.all;

  package READ_COMPLEX_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                      SQUARE_COORDINATE, COMPLEX_NUMBER, COMPLEX_IMAGES.IMAGE);
  use READ_COMPLEX_IMAGE;

  package WRITE_COMPLEX_IMAGE is new BINARY_IMAGE_IO
                                      (SQUARE_COORDINATE, SQUARE_COORDINATE,
                   COMPLEX_NUMBERS_FLOAT.COMPLEX_NUMBER, COMPLEX_IMAGES.IMAGE);
  use WRITE_COMPLEX_IMAGE;

  package Real_IO is new Float_IO(FLOAT); use Real_IO;

  mod1mod2, alpha: float;
  choice: character;

begin
  put("Just multiply (m) by mormalisation**alpha or phase correlate (p) as
well? ");
  get(choice);
  put("Alpha = "); get(alpha);
  READ_COMPLEX_IMAGE.READ_BINARY_IMAGE_FILE(INPUT1_FILENAME, S1);
  READ_COMPLEX_IMAGE.READ_BINARY_IMAGE_FILE(INPUT2_FILENAME, S2);
  READ_COMPLEX_IMAGE.READ_BINARY_IMAGE_FILE(PC_FILENAME, S3);
  if choice = 'p' then
    FFT_N.FFT_BY_ROW_COLUMN_SEPARATION(S1);
    FFT_N.FFT_BY_ROW_COLUMN_SEPARATION(S2);
    for y in S2'RANGE(1) loop
      for x in S2'RANGE(2) loop
        S2(y,x) := CONJUGATE(S2(y,x));
        if (S1(y,x) = COMPLEX(0.0, 0.0)) or (S2(y,x) = COMPLEX(0.0, 0.0)) then
          S3(y,x) := COMPLEX(0.0, 0.0);
        else
          mod1mod2 := SQRT(REAL(S1(y,x))**2+IMAGINARY(S1(y,x))**2)*
```

```
                      SQRT(REAL(S2(y,x))**2+IMAGINARY(S2(y,x))**2);
        S3(y,x) := (S1(y,x)*S2(y,x))/(mod1mod2**(1.0-alpha));
      end if;
    end loop;
  end loop;
else
  for y in S3'RANGE(1) loop
    for x in S3'RANGE(2) loop
      S2(y,x) := CONJUGATE(S2(y,x));
        mod1mod2 := SQRT(REAL(S1(y,x))**2+IMAGINARY(S1(y,x))**2)*
                    SQRT(REAL(S2(y,x))**2+IMAGINARY(S2(y,x))**2);
      S3(y,x) := S3(y,x)*(mod1mod2**alpha);
    end loop;
  end loop;
end if;
WRITE_COMPLEX_IMAGE.WRITE_BINARY_IMAGE_FILE(OUTPUT_FILENAME, S3);
end pctimesalpha;
```

PHASESET.ADA

```
with TEXT_IO, BINARY_IMAGE_IO, SYSTEM_ENVIRONMENT, ELEMENTARY_FUNCTIONS,
     SIZE_OF_IMAGE, SIZE_OF_FILE, COMPLEX_NUMBERS_FLOAT, COMMON_IMAGE_TYPES,
     IMAGE_COORDINATES;
use  TEXT_IO, COMPLEX_NUMBERS_FLOAT, ELEMENTARY_FUNCTIONS, IMAGE_COORDINATES;

-- Reads in a complex image and alters the phase of each pixel to a set value.
-- Author: Amy Thornton

procedure PHASESET is

   INPUT_FILENAME : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
  OUTPUT_FILENAME : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);

  SIZE : constant NATURAL := 2**SIZE_OF_IMAGE(SIZE_OF_FILE(INPUT_FILENAME)/
                                                        (2 * FLOAT'SIZE/8));
  package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
  use IMAGES_N;

  type COMPLEX_IMAGE_PTR is access COMPLEX_IMAGES.IMAGE;
  SP : constant COMPLEX_IMAGE_PTR := new COMPLEX_IMAGES.IMAGE;
  S  : COMPLEX_IMAGES.IMAGE renames SP.all;

  package READ_COMPLEX_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                      SQUARE_COORDINATE, COMPLEX_NUMBER, COMPLEX_IMAGES.IMAGE);
  package WRITE_COMPLEX_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                      SQUARE_COORDINATE, COMPLEX_NUMBER, COMPLEX_IMAGES.IMAGE);

  package Real_IO is new Float_IO(FLOAT); use Real_IO;
  package Int_IO is new INTEGER_IO(INTEGER); use Int_IO;
  package Coord_IO is new INTEGER_IO(COORDINATE); use Coord_IO;

  REALANG, OldMod, NewMod, ratio, R, I: Float;

begin
  put("Phase angle (degrees) to be set to:      "); get(REALANG);
  READ_COMPLEX_IMAGE.READ_BINARY_IMAGE_FILE(INPUT_FILENAME, S);
  for y in S'RANGE(1) loop
    for x in S'RANGE(2) loop
      R := REAL(S(y,x));
      I := IMAGINARY(S(y,x));
      OldMod := ELEMENTARY_FUNCTIONS.SQRT(R**2 + I**2);
      if OldMod /= 0.0 then
        begin
          if (REALANG >= 270.0) or (REALANG <= 90.0) then
            R := abs(R);
          else
            R := -abs(R);
          end if;
          I := R*ELEMENTARY_FUNCTIONS.TAN(REALANG, 360.0);
          NewMod := ELEMENTARY_FUNCTIONS.SQRT(R**2 + I**2);
          begin
            ratio := OldMod/NewMod;
            R := R*ratio;
            I := I * ratio;
            S(y,x) := COMPLEX(R, I);
          exception
            when CONSTRAINT_ERROR =>
              put("y, x, newmod, oldmod: constraint_error  ");
              put(y, width=>3); put(" "); put(x, width=>3);put(" ");
              put(NewMod); put(OldMod); NEW_LINE;
              S(y,x) := COMPLEX(0.0, 0.0);
          end;
        exception
          when ELEMENTARY_FUNCTIONS.ARGUMENT_ERROR =>
            put("y, x, value:            argument_error");
            put(y, width=>3); put(" "); put(x, width=>3); put(" ");
            put(REALANG); NEW_LINE;
```

```
            S(y,x) := COMPLEX(0.0, 0.0);
          end;
        end if;
      end loop;
    end loop;
    WRITE_COMPLEX_IMAGE.WRITE_BINARY_IMAGE_FILE(OUTPUT_FILENAME, S);
end PHASESET;
```

RASTTOBIN.ADA

```
with TEXT_IO, BINARY_IMAGE_IO, SYSTEM_ENVIRONMENT, ELEMENTARY_FUNCTIONS,
     SIZE_OF_IMAGE, SIZE_OF_FILE, PIXEL_TYPES, COMMON_IMAGE_TYPES,
     IMAGE_COORDINATES, SEQUENTIAL_IO, SIZE_OF_FILE, DIRECT_IO;
use  TEXT_IO, PIXEL_TYPES, IMAGE_COORDINATES;

-- Generates the binary file from the raster image.
-- Author: Amy Thornton

procedure rasttobin is

  RASTFILE   :  constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
  BINFILE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);

  SIZE : NATURAL;

  RAS_MAGIC     : constant := 16#59A66A95#;

  package INT_IO is new INTEGER_IO(INTEGER); use INT_IO;

  W, L: INTEGER;
  RT, MT, ML, RuNo, S : INTEGER;

begin
  put("X Size? "); get(W);
  put("Y Size? "); get(L);
  put("Ras_type? 1?"); get (RT);
  put("Map_type? 0?"); get (MT);
  put("Map_length? 0?"); get (ML);
  put("Rubbish? "); get(RuNo);
  declare
    subtype YCOORD is NATURAL range 0 .. L-1;
    subtype XCOORD is NATURAL range 0 .. W-1;

    type RUBBISH is array (INTEGER RANGE 0 .. RuNo) of PIXEL;

    type IMAGE is array(YCOORD, XCOORD) of PIXEL;
    I : IMAGE;
    package WRITE_BINARY_IMAGE is new BINARY_IMAGE_IO(YCOORD, XCOORD, PIXEL,
                                                                    IMAGE);
    use WRITE_BINARY_IMAGE;

    type RASTER is
      record
        MAGIC_NO    : INTEGER := RAS_MAGIC;
        WIDTH       : INTEGER := W;
        HEIGHT      : INTEGER := L;
        DEPTH       : INTEGER := 8;
        LENGTH      : INTEGER := W*L;
        RASTER_TYPE : INTEGER := RT;
        MAPTYPE     : INTEGER := MT;
        MAPLENGTH   : INTEGER := ML;
        Ru          : RUBBISH;
        IM          : IMAGE;
    end record;

    R : RASTER;

    package RASTER_IO is new SEQUENTIAL_IO(RASTER);
    use RASTER_IO;
    FILE : RASTER_IO.FILE_TYPE;

  begin
    OPEN(FILE, IN_FILE, RASTFILE);
    READ (FILE, R);
--    for y in YCOORD loop
--      for x in XCOORD loop
--        R.IM(y,x) := 255-R.IM(y,x);
```

```
--      end loop;
--    end loop;
    WRITE_BINARY_IMAGE_FILE(BINFILE, R.IM);
    CLOSE (FILE);
  end;
end rasttobin;
```

RASTTOCOLBIN.ADA

```
with TEXT_IO, BINARY_IMAGE_IO, SYSTEM_ENVIRONMENT, ELEMENTARY_FUNCTIONS,
     SIZE_OF_IMAGE, SIZE_OF_FILE, PIXEL_TYPES, COMMON_IMAGE_TYPES,
     IMAGE_COORDINATES, SEQUENTIAL_IO;
use  TEXT_IO, PIXEL_TYPES, IMAGE_COORDINATES;

-- Generates the colour binary file from the raster image.
-- Author: Amy Thornton

procedure rasttocolbin is

  RASTFILE   :  constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);
  COLBINFILE : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(2);

  SIZE : NATURAL;

  RAS_MAGIC     : constant := 16#59A66A95#;

  package INT_IO is new INTEGER_IO(INTEGER); use INT_IO;

  W, L: INTEGER;

begin
  put("X Size? "); get(W);
  put("Y Size? "); get(L);
  declare
    subtype YCOORD is NATURAL range 0 .. L-1;
    subtype XCOORD is NATURAL range 0 .. W-1;
    type COLOUR_IMAGE is array(YCOORD, XCOORD) of COLOUR_PIXEL;
    package WRITE_COL_IMAGE is new BINARY_IMAGE_IO(YCOORD, XCOORD,
                                                   COLOUR_PIXEL, COLOUR_IMAGE);
    I: COLOUR_IMAGE;

    type RASTER is
      record
        MAGIC_NO    : INTEGER := RAS_MAGIC;
        WIDTH       : INTEGER := W;
        HEIGHT      : INTEGER := L;
        DEPTH       : INTEGER := 24;
        LENGTH      : INTEGER := W*L;
        RASTER_TYPE : INTEGER := 3;
        MAPTYPE     : INTEGER := 0;
        MAPLENGTH   : INTEGER := 0;
        IM          : COLOUR_IMAGE;
    end record;

    R : RASTER;

    package RASTER_IO is new SEQUENTIAL_IO(RASTER);
    use RASTER_IO;

    FILE : RASTER_IO.FILE_TYPE;

  begin
    OPEN(FILE, IN_FILE, RASTFILE);
    READ (FILE, R);
    WRITE_COL_IMAGE.WRITE_BINARY_IMAGE_FILE(COLBINFILE, R.IM);
    CLOSE (FILE);
  end;
end rasttocolbin;
```

SNRC.ADA

```
with TEXT_IO, SYSTEM_ENVIRONMENT, BINARY_IMAGE_IO, COMPLEX_NUMBERS_FLOAT,
     SIZE_OF_IMAGE, PIXEL_TYPES, SIZE_OF_FILE, ELEMENTARY_FUNCTIONS,
     COMMON_IMAGE_TYPES, IMAGE_COORDINATES, LOG10;
use  TEXT_IO, ELEMENTARY_FUNCTIONS, IMAGE_COORDINATES, PIXEL_TYPES,
     COMPLEX_NUMBERS_FLOAT;

-- Allows the signal-to-noise ratio to be calculated.
-- Author: Amy Thornton

procedure snrc is

  COMPFILE  : constant STRING := SYSTEM_ENVIRONMENT.ARG_VALUE(1);

  SIZE : constant NATURAL := 2**SIZE_OF_IMAGE(SIZE_OF_FILE(COMPFILE)/
                                                    (2 * FLOAT'SIZE/8));
  SIZEC : constant coordinate := coordinate(SIZE);

  package IMAGES_N is new COMMON_IMAGE_TYPES(SIZE);
  use IMAGES_N;

  S1  : COMPLEX_IMAGES.IMAGE;

  package READ_WRITE_COMPLEX_IMAGE is new BINARY_IMAGE_IO(SQUARE_COORDINATE,
                     SQUARE_COORDINATE, COMPLEX_NUMBER, COMPLEX_IMAGES.IMAGE);

  package Real_IO is new Float_IO(FLOAT); use Real_IO;
  package Nat_IO is new INTEGER_IO(Natural); use Nat_IO;
  package Coord_IO is new INTEGER_IO(COORDINATE); use Coord_IO;
  NOISESQSUM, NOISERMS, NOISEMRS, SNRRMS, SNRMRS, signalmod, modvalue,
signoise, TotNoise: float;
  ypk, xpk, level, xmod, ymod: coordinate;
  pixignore: natural;

  procedure SIGPLUSNOISE(IMAGEIN:in COMPLEX_IMAGES.IMAGE; Noiseall:in out
float) is
    modvalue, maxpix: float;
  begin
    maxpix := 0.0;
    Noiseall := 0.0;
    for y in IMAGEIN'RANGE(1) loop
      for x in IMAGEIN'RANGE(2) loop
        modvalue := (REAL(IMAGEIN(y,x)))**2 + (IMAGINARY(IMAGEIN(y,x)))**2;
        if modvalue > maxpix then
          maxpix := modvalue;
        end if;
        Noiseall := modvalue + Noiseall;
      end loop;
    end loop;
    maxpix := SQRT(maxpix);
    put("The maximum (mod) value is ");
    put(maxpix, Fore => 11, Aft=>4, Exp=>0); NEW_LINE;
  end SIGPLUSNOISE;

begin
  READ_WRITE_COMPLEX_IMAGE.READ_BINARY_IMAGE_FILE(COMPFILE, S1);
  put("Peak position? x = "); get(xpk); put("Peak position? y = "); get(ypk);
  put("Ignore surrounding pixels?"); NEW_LINE;
  put("0 (only peak), 1 (8 pixels), 2 (24 pixels), 3 (48 pixels) etc: ");
  get(level);
  SIGPLUSNOISE(S1, TotNoise);
  signoise := 0.0;
  pixignore := 0;
  for y in (ypk-level) .. (ypk+level) loop
    for x in (xpk-level) .. (xpk+level) loop
      ymod := y mod SIZEC; xmod := x mod SIZEC;
      modvalue := (REAL(S1(ymod, xmod)))**2 + (IMAGINARY(S1(ymod, xmod)))**2;
      signoise := modvalue + signoise;
```

```
      pixignore := pixignore + 1;
    end loop;
  end loop;
  put("The number of pixels ignored in the noise calculation is ");
  put(PIXIGNORE, 7); NEW_LINE;
  NOISESQSUM := TotNoise - signoise;
  signalmod := SQRT((REAL(S1(ypk,xpk)))**2+(IMAGINARY(S1(ypk,xpk)))**2);
  NOISERMS := SQRT(NOISESQSUM/float((256*256) - pixignore));
  NOISEMRS := SQRT(NOISESQSUM)/float((256*256) - pixignore);
  SNRRMS := signalmod/NOISERMS;
  SNRMRS := signalmod/NOISEMRS;
  put("NOISE S =      "); put(NOISESQSUM, Fore => 14, Aft=>4, Exp=>0);
  NEW_LINE;
  put("NOISE MRS =    "); put(NOISEMRS, Fore => 14, Aft=>4, Exp=>0);
  NEW_LINE;
  put("SNR MRS =      "); put(SNRMRS, Fore => 14, Aft=>4, Exp=>0); NEW_LINE;
  put("SNR MRS (dB) = "); put((20.0*LOG10(SNRMRS)), Fore => 14, Aft=>4,
Exp=>0);
  NEW_LINE;
  put("     Peak value (mod) = "); put(signalmod,  Fore => 11, Aft=>4,
Exp=>0);
  NEW_LINE;
  put("     NOISE RMS =        "); put(NOISERMS, Fore => 11, Aft=>4, Exp=>0);
  NEW_LINE;
  put("     SNR RMS =          "); put(SNRRMS, Fore => 11, Aft=>4, Exp=>0);
  NEW_LINE;
  put("     SNR RMS (dB) =     "); put((20.0*LOG10(SNRRMS)), Fore => 11,
Aft=>4, Exp=>0);
  NEW_LINE;
end snrc;
```

APPENDIX D - PUBLISHED RESEARCH

Three conference papers which have been presented as a result of the research described in this thesis are included in this appendix. In addition, the author has co-authored a chapter in the *Handbook of Colour Image Processing* [Sangwine and Thornton 1998].

APPENDIX D.1: "COLOUR OBJECT LOCATION USING COMPLEX CODING IN THE FREQUENCY DOMAIN"

The first paper reprinted below, [Thornton and Sangwine 1995], was presented at the IEE 5th International Conference on Image Processing and its Applications in Edinburgh, UK in July 1995.

This page of the original thesis contained a reprint of a copyright paper and is not reproduced in this electronic version.

This page of the original thesis contained a reprint of a copyright paper and is not reproduced in this electronic version.

This page of the original thesis contained a reprint of a copyright paper and is not reproduced in this electronic version.

This page of the original thesis contained a reprint of a copyright paper and is not reproduced in this electronic version.

This page of the original thesis contained a reprint of a copyright paper and is not reproduced in this electronic version.

APPENDIX D.2: "COLOUR OBJECT RECOGNITION USING PHASE CORRELATION OF LOG-POLAR TRANSFORMED FOURIER SPECTRA"

The following paper, [Thornton and Sangwine 1996], was presented at the International Workshop on Image and Signal Processing '96 in Manchester, UK in November 1996.

This page of the original thesis contained a reprint of a copyright paper and is not reproduced in this electronic version.

This page of the original thesis contained a reprint of a copyright paper and is not reproduced in this electronic version.

This page of the original thesis contained a reprint of a copyright paper and is not reproduced in this electronic version.

This page of the original thesis contained a reprint of a copyright paper and is not reproduced in this electronic version.

APPENDIX D.3: "LOG-POLAR SAMPLING INCORPORATING A NOVEL SPATIALLY VARIANT FILTER TO IMPROVE OBJECT RECOGNITION"

The following paper, [Thornton and Sangwine 1997], was presented at the IEE 6th International Conference on Image Processing and its Applications in Dublin, Ireland in July 1997.

This page of the original thesis contained a reprint of a copyright paper and is not reproduced in this electronic version.

This page of the original thesis contained a reprint of a copyright paper and is not reproduced in this electronic version.

This page of the original thesis contained a reprint of a copyright paper and is not reproduced in this electronic version.

This page of the original thesis contained a reprint of a copyright paper and is not reproduced in this electronic version.