Direction of Camera Based on Shadows

Darren Caulfield
B.A. (Mod.) Computer Science

Final Year Project, May 2003
Supervisor: Dr Kenneth Dawson-Howe

Abstract

The compass direction in which a camera is facing (e.g. North, South, East, West) is an important piece of information for surveillance applications that must relate input obtained from multiple cameras. When the directions of cameras are known an object, e.g. a person, appearing in one camera is constrained in where it can appear in another. This project determines a camera's compass direction by exploiting the information conveyed by the shadows in a video sequence.

The position of the Sun in the sky is calculated for the given date, time and geographic location using astronomical formulae. Shadows in the scene are identified using a combination of background subtraction and a colour model that is similar to the human perception of colour. By associating a direction with each of the shadows it is possible to determine the orientation of the Sun with respect to the camera. A pinhole camera model is used to simplify the mathematics. Combining this orientation information with the astronomical data yields the compass direction of the camera.

The project successfully determines the camera's direction given a video sequence, in spite of difficulties in the identification of shadows. The present solution to the problem requires that the user provide certain parameters. A technique for tracking lines across video frames, developed during the course of the project, has the potential to make the algorithm fully automatic, in addition to increasing the accuracy of the results.

Acknowledgements

I would like to thank my supervisor Kenneth Dawson-Howe for his advice and direction throughout this project.

I must also thank my family and friends for their support and encouragement when things were not working out.

Table Of Contents

ABSTRACT ..... **II**

ACKNOWLEDGEMENTS ..... **III**

1 INTRODUCTION **1**

1.1 AIMS 1
1.2 MOTIVATION 2

2 OVERVIEW OF PROBLEM **4**

2.1 POSITION OF THE SUN 4
2.2 IDENTIFICATION OF SHADOWS 5
2.3 DIRECTION OF SHADOWS 6
2.4 INFERRING THE CAMERA DIRECTION 7

3 ASTRONOMY **8**

3.1 ACCURACY 8
3.2 EQUATORIAL CO-ORDINATES 9
3.2.1 Julian Dates *10*
3.2.2 Ecliptic Longitude *11*
3.2.3 Right Ascension and Declination *11*
3.3 HORIZONTAL CO-ORDINATES 12
3.3.1 Sidereal Time *12*
3.3.2 Altitude and Azimuth *13*

4 SHADOW IDENTIFICATION **14**

4.1 EXISTING RESEARCH 14
4.2 ALGORITHM SELECTION 15
4.3 BACKGROUND IMAGE 15
4.4 COLOUR MODELS 16
4.5 ALGORITHM 18
4.6 PROBLEMS 20
4.6.1 Isolated Object and Shadow Pixels *20*
4.6.2 False Shadow *20*

5 SHADOW DIRECTION **21**

5.1 IDENTIFICATION OF REGIONS 22
5.2 DIRECTION OF A REGION 23
5.3 REMOVAL OF FALSE SHADOW 26
5.4 EXCLUDING UNRELIABLE LINE SEGMENTS 27

6 CAMERA DIRECTION INFERENCE 30
6.1 CO-ORDINATE SPACES 30
6.2 CAMERA ORIENTATION AND PARAMETERS 31
6.2.1 Extrinsic Parameters 32
6.2.2 Intrinsic Parameters 33
6.3 BACK-PROJECTION 33
6.4 VANISHING POINT ESTIMATION 35
6.4.1 Unsuccessful Approaches 36
6.4.2 TanBin Partitioning 36
6.5 COMPASS DIRECTION OF CAMERA 39
6.5.1 Virtual Compass 39
7 EVALUATION 40
7.1 RESULTS 40
7.2 SUCCESSFUL ASPECTS OF THE PROJECT 42
7.2.1 Inferring the Camera Direction 42
7.2.2 Application Design 43
7.3 DIFFICULTIES ENCOUNTERED 43
7.4 FUTURE WORK 44
8 CONCLUSION 46
9 REFERENCES 48
APPENDIX A: JULIAN DATES 50
APPENDIX B: LEAST SQUARES METHOD 51

1 Introduction

Computer Vision seeks to allow computers to act intelligently by emulating the human vision system. This is an extremely difficult problem. To be solved completely we would need to understand at a very deep level how the human brain operates. Most experts would agree that this milestone is some time away.

1.1 Aims

In spite of the fact that Computer Vision is very difficult it is still possible to solve specific problems in the area. This project is one such problem. Its aim is to develop a computer system that can infer from a video sequence the direction in which the video camera is pointing. "Direction" in this case refers to the compass angle of the camera, e.g. "23 degrees West of South". The system is to use the shadows that are observed in the video clip to determine the direction.

In order to clarify the aim of the project it is useful to take an example of how it will operate, in terms of inputs and outputs. The system will be presented with a video clip shot from a fixed camera. Typical frames from such a sequence are shown in Figure 1.1.

Figure 1.1: Some frames from an input video sequence

The user must provide information about the video sequence, including the date, time and geographic location (longitude and latitude) at which it was shot. These data are needed to calculate the compass direction of the shadows (which is determined by the

Sun's position in the sky). Doing so will allow the compass direction of the camera to be obtained. This is the principal result of the system and will be displayed to the user in both a textual and a visual format. The latter will be a depiction of a compass similar to Figure 1.2.

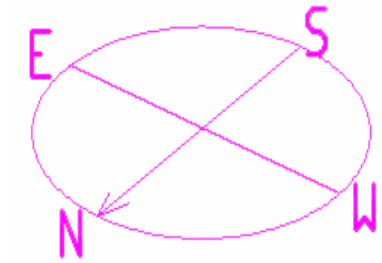

Figure 1.2: Virtual compass illustrating camera's direction

1.2 Motivation

Considering the project in isolation, it is not obvious why such a problem should be tackled. However, when the project is considered in the broader framework of a surveillance application its usefulness becomes apparent. One possible scenario is the collection of data from multiple closed-circuit television cameras (CCTV). The approximate location of the cameras might be known, but it is possible that the direction in which each of them was pointing would not be clear. This ambiguity would greatly hinder the ability of a computer-based system to correlate information from two or more cameras.

If such a system were used to track a person as they moved out of shot of one camera and into another, it would be extremely helpful if there were a constraint on where the person might appear. Figure 1.3 illustrates the difficulties caused by not knowing the relative orientation of two cameras. A person appearing in camera 1 will appear in camera 2 a short time later. However, if the cameras are arranged as in (a) the person will move into shot of camera 2 from the left, whereas in (b) they will appear from the right.

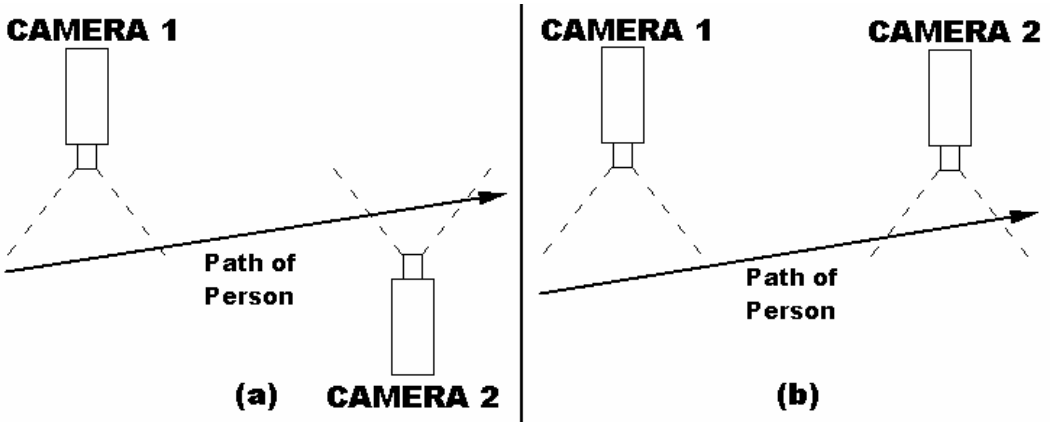

Figure 1.3: Difficulty of tracking when camera directions are unknown

By knowing the compass direction of every camera the system could work out how they were arranged relative to one another. This would simplify the tracking of people by discounting a high proportion of possible correlations. In Figure 1.3 (b), for example, only people who moved into shot of camera 2 from the *right* could have appeared in camera 1 (ignoring unusual travel routes).

The example given above best illustrates the usefulness of this project. It should be regarded as something of a "calibration stage" in the setting up of a much larger surveillance system.

2 Overview of Problem

This chapter is intended to give an overview of the project, identifying the various problems that need to be addressed to arrive at a working solution. Presenting a "road-map" of the project at this stage should help to clarify how the major sections of the project are related. The present chapter will also serve as a "reader's guide" to the rest of the document by outlining the purpose of each chapter.

The principal steps required to establish the camera's compass direction are as follows:

- calculate the position of the Sun in the sky
- identify shadows in the video sequence
- associate a direction with each shadow
- infer the camera's direction from the direction of the shadows and the position of the Sun

The motivation behind each of these steps is discussed in the following sections.

2.1 Position of the Sun

In order to calculate the compass direction of the shadows seen in the video sequence, e.g. "6 degrees East of South", it is necessary to determine the Sun's position in the sky. Figure 2.1 illustrates that all shadows cast on the ground point in the direction of the Sun. All such shadows are in fact parallel, which is a consequence of the Sun being at a very great distance from the Earth.

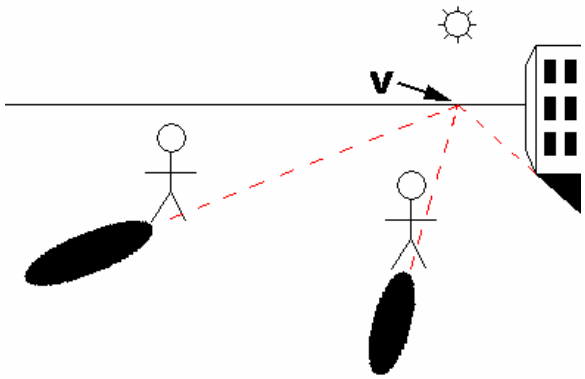

Figure 2.1: All ground shadows point towards the Sun

The Sun's position depends on several variables, which the user must provide. These are: the date and time at which the video sequence was shot, and the geographic location (latitude and longitude) of the camera. Because the various shadows cast on the ground are directed towards the point on the horizon directly below the Sun (see Figure 2.1, point 'V'), all that is required in terms of the Sun's position is its *azimuth*. The azimuth of any object in the sky is the angle between the South point on the horizon and the vertical projection of the object onto the horizon, travelling Westwards (see Figure 2.2).

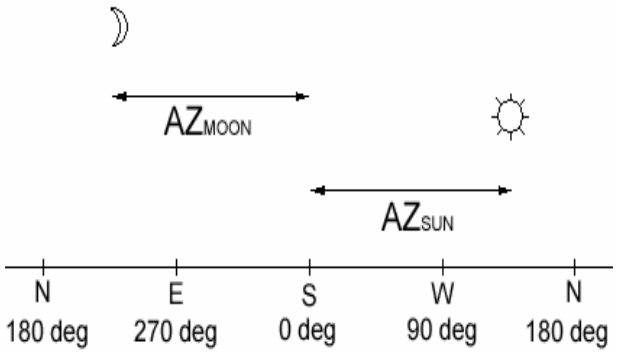

Figure 2.2: Panorama of horizon with the azimuths of celestial objects marked

The astronomical models and calculations required to obtain the Sun's azimuth for any given date, time and position on the planet are treated in detail in Chapter 3.

2.2 Identification of Shadows

Another key element of the project is the automatic identification of shadows. In order to make the overall problem tractable it was decided that only shadows *cast by people on the ground* would be considered. It is necessary to locate as many such shadows in the video sequence as possible so that their orientation relative to the camera can be established. Figure 2.3 shows a frame from a video sequence (a) and the shadows (shown in red) that have been identified in it (b).

Figure 2.3: A frame from a video sequence (a), and the shadows identified (b)

Because of the constraints imposed on the nature of the shadows only very specific video clips can be used in the project. These clips are typically wide shots of a flat, open area across which many people walk. The details of the technique used in this project to find the shadows are presented in Chapter 4.

2.3 Direction of Shadows

It may not be immediately obvious but it is far from trivial for a computer program to determine the direction in which a shadow is pointing, even when the shadow itself has been identified. For example, we as humans have little difficulty in associating a direction with each of the shadows identified in Figure 2.4 (a). The results are likely similar to Figure 2.4 (b). However, developing an algorithm to perform this task is not straightforward. The solution to the problem is detailed in Chapter 5.

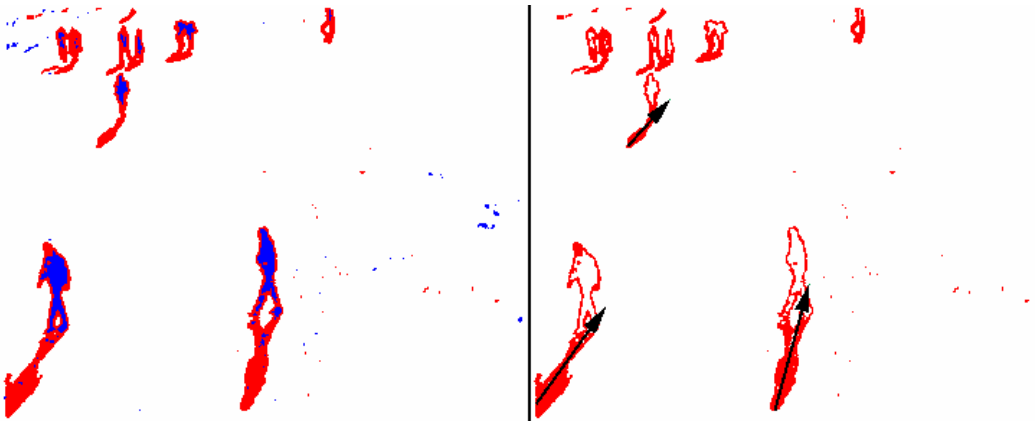

Figure 2.4: Identified shadows (a), and their associated directions (b)

2.4 Inferring the Camera Direction

Associating a direction with each of the shadows in the video clip provides *some* information about how they and the camera are oriented relative to one another (see Figure 2.5). However, the mathematics of perspective projection must be understood before the relative orientation can be calculated. Perspective projection describes how the camera transforms a 3D world into a 2D image.

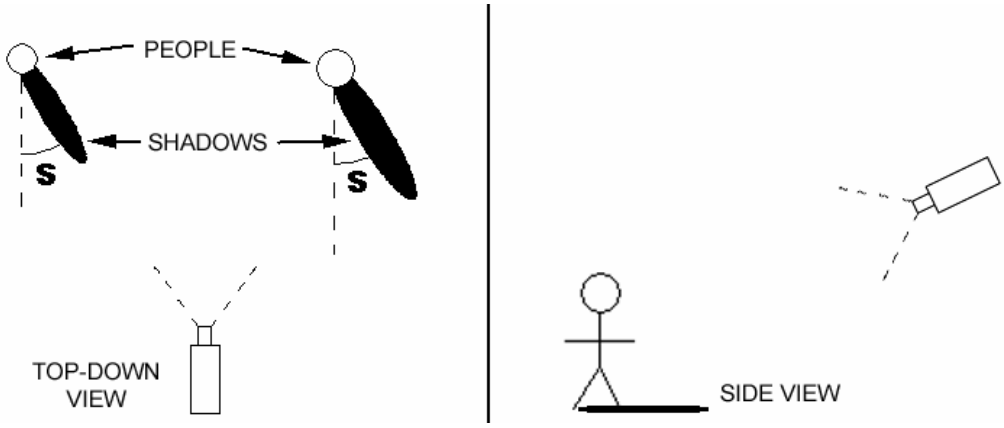

Figure 2.5: Orientation (S) of camera relative to shadows

By combining the relative orientation of the camera and the shadows with the Sun's position in the sky the compass direction of the camera is found. Chapter 6 discusses the camera models and the mathematics required to calculate the end-result, i.e. the compass direction of the camera.

3 Astronomy

In the previous chapter it was explained that the Sun's position for any given date, time and location on the planet is required. This information will allow the direction of shadows to be used to establish the compass direction of the camera.

The relationships between the Sun's position and date, time and location are complex. This makes it infeasible to develop any from of look-up table to find the required information – there are simply too many variables. An analytical approach must be used to solve the problem.

3.1 Accuracy

Calculating the Sun's position is a well-documented problem in astronomy [EQUA 02, GIES 03, NOAA 03]. As with many such problems the solution exists, and can be found using a reasonably simple algorithm – the only question is about the desired accuracy.

In astronomy angular distances are often used to describe how far apart or how big objects in the sky appear to be. For example, one's fist held at arm's length is about 10 degrees of arc wide. There are 60 arc minutes in one degree of arc. An understanding of angular measure will help in the discussion of accuracy that follows.

The first algorithm presented below (for calculating the Sun's equatorial co-ordinates) is the same as that found in [EQUA 02]. Details of its accuracy are also presented (ibid.). They show that the right ascension and declination of the Sun (discussed below) that the algorithm calculates are accurate to approximately one arc minute in the time period 1800A.D. to 2200A.D. To put this in context, the Sun's disk is about 30 arc minutes in diameter. The second algorithm (for calculating horizontal co-ordinates) has similar accuracy. This ensures that the Sun's position will be correct to a fraction of a degree. Note that the Sun moves through approximately one degree of sky every 4 minutes. The algorithm has, at the very least, a 400-year window of applicability!

It should at all times be remembered that the accuracy of the final result of the project, i.e. the camera's compass direction, will depend on factors other than the astronomical algorithms presented below. These include:

- knowing the exact time of day at which the video was shot
- the flatness of the ground onto which the shadows are cast
- the correct identification of shadows

3.2 Equatorial Co-ordinates

For the purposes of finding the Sun's position in the sky it is best initially to ignore the complexities introduced by the location of the observer on the Earth. Depending on the time of year the Sun may, for example, be directly overhead in Rio de Janeiro and at the same moment be setting in Dublin. Such differences will be accounted for in the next section: *Horizontal Co-ordinates*.

The standard approach to finding the position of any celestial body (e.g. Sun, Moon, planet) in the sky involves first finding its *equatorial co-ordinates*. In this co-ordinate system the entire sky is perceived as a large sphere (the *celestial sphere)* surrounding the Earth (see Figure 3.1). It is possible to reference any point on the sphere by quoting its *right ascension* (similar to longitude on Earth) and *declination* (similar to latitude).

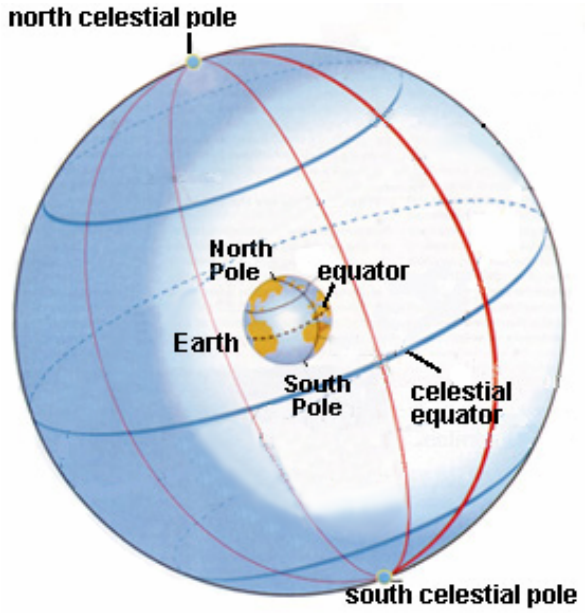

Figure 3.1: The celestial sphere, from [RIT 98]

Key to finding the Sun's position on the celestial sphere for any given date and time is the observation that it always follows the *ecliptic*. This is a great circle on the celestial sphere, inclined at approximately 23.5° to the celestial equator (see Figure 3.2). (Note that the ecliptic passes through all of the signs of the Zodiac.) The consequence of this is that it is only necessary to calculate how far "around" the ecliptic the Sun has travelled for a given date and time in order to find its equatorial co-ordinates.

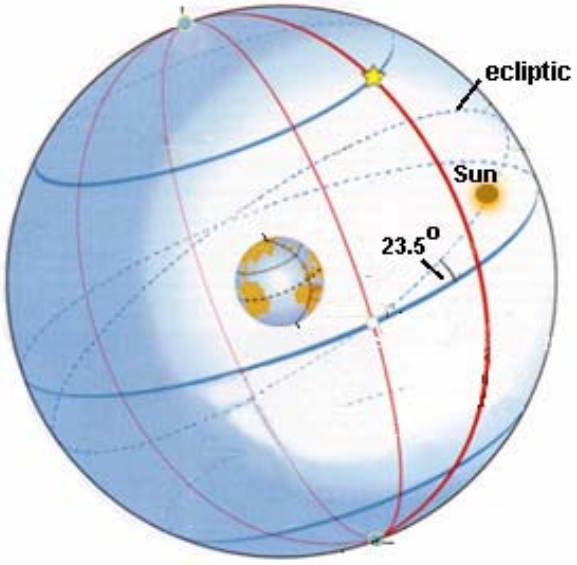

Figure 3.2: The Sun following the ecliptic, from [RIT 98]

3.2.1 Julian Dates

Before the Sun's position can be calculated, the date and time of interest must be converted to a useful format. Many astronomical algorithms work in terms of a *Julian date* [JULD 01, OFEK 02]. This is simply a continuous count of days and fractions of days since noon Universal Time on 1 January 4713 B.C. (on the Julian calendar). Universal Time is essentially the same as Greenwich Mean Time for the purposes of this project. The algorithm used to convert calendar dates and times (e.g. 28 March 2003 14:52:00) to Julian dates is given in Appendix A.

3.2.2 Ecliptic Longitude

The distance the Sun has travelled around the ecliptic for a given date and time is referred to as its *ecliptic longitude*. The algorithm for calculating it is as follows [EQUA 02]:

Compute the Julian date, JD, for the date and time of interest

Compute the number of days, D, since 1 January 2000 at 12:00p.m. GMT. This date and time is called "the epoch J2000.0" and represents the point in time when the algorithm is most accurate. Its Julian date is 2451545.0.

$$D = JD - 2451545.0$$

Compute L, the Sun's ecliptic longitude using the following formulae (g, q an L are all in degrees)

$$g = 357.529 + 0.98560028\ D$$

$$q = 280.459 + 0.98564736\ D$$

$$L = q + 1.915 \sin g + 0.020 \sin 2g$$

The algorithm above yields the very important number L, the ecliptic longitude. It is very difficult to explain what the formulae mean – they are beyond the scope of the project. Put simply, they represent the fact that the Earth travels around the Sun not in a circle but in an ellipse, varying its speed as it does so. Furthermore, the shape and orientation in space of this elliptical orbit are changing with time.

3.2.3 Right Ascension and Declination

With the ecliptic longitude L obtained it is relatively simple to find the Sun's position on the celestial sphere (see Figure 3.2). Spherical trigonometry is used to solve the problem. The well-known tilt of the Earth's axis (E, roughly 23.5°) is required for this calculation. It is decreasing slowly with time, and can be approximated by

$$E = 23.439 - 0.00000036\ D$$

Right ascension (RA) and declination (d) are then found using the following formulae (the C-function "atan2" must be used when coding the algorithm):

$$\tan RA = \cos E \sin L / \cos L$$

$$\sin d = \sin E \sin L$$

3.3 Horizontal Co-ordinates

Once the equatorial co-ordinates of the Sun have been found it is necessary to take account of the Earth's daily rotation, and the fact that each observer sees a different "local sky" depending on their position on the planet. *Horizontal co-ordinates* are required for this purpose. They quote the Sun's position as its angle above the observer's horizon (altitude) and its angle "around" the horizon from some reference point (azimuth). The Sun's azimuth is, by definition, equivalent to its compass direction, which is precisely what is required for this project (see Figure 3.3).

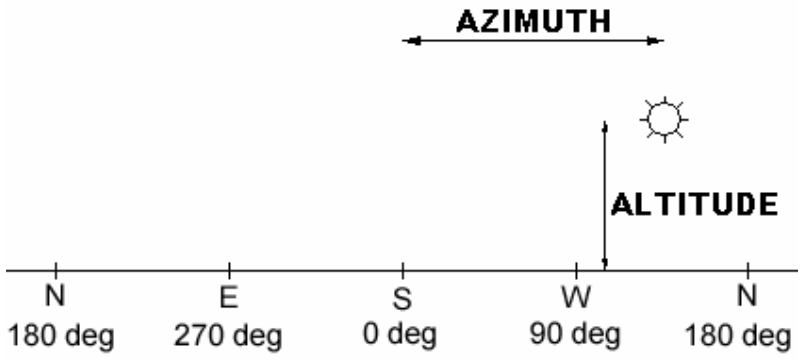

Figure 3.3: Compass points on the horizon, and the Sun's horizontal co-ordinates

3.3.1 Sidereal Time

It may seem strange, but our day of 24 hours is a somewhat arbitrary, albeit useful, measure of time. It is merely the average length of time from "high noon" on one day until high noon on the next. (Winter days in the Northern hemisphere can exceed 24 hours in length by as much as 30 seconds due to the Earth's elliptical orbit [DARK 03].) For astronomical purposes *sidereal time* is more useful than "mean solar time" (as clock time is properly known). The Earth takes one sidereal day to rotate with

respect to the stars on the celestial sphere (see Figure 3.1). A sidereal day is approximately 23 hours 56 minutes and 4 seconds long. An observer's local sidereal time (LST) depends on their longitude and Julian date (D) as follows [SCHM 02]:

$$LST = 280.46061837 + 360.98564736629\ D + longitude$$

Note that longitudes East of Greenwich are taken as positive. From the local sidereal time the Sun's hour angle (HA) can be calculated [BUR 01, ASTR 98]. This is a measure of how long it has been since the Sun was at its highest in the sky, and will subsequently allow the Sun's position in the sky to be calculated. In the following formula RA refers to the Sun's right ascension, calculated as described previously.

$$HA = LST - RA$$

3.3.2 Altitude and Azimuth

Finding the Sun's altitude and azimuth involves transforming points on the celestial sphere into points on the observer's local sky, a hemisphere (see Figure 3.4). Spherical trigonometry is again used to solve the problem [GIES 01]. Altitude (Alt) and azimuth (Az) are derived from HA, the Sun's declination (d) and the observer's latitude (Lat) as follows (the C-function "atan2" must be used when coding the algorithm):

$$\sin Alt = \sin Lat\ \sin d\ +\ \cos Lat\ \cos d\ \cos HA$$

$$\tan (Az + 180) = (-\sin HA) / (\cos Lat\ \tan d\ -\ \sin Lat\ \cos HA)$$

With the Sun's azimuth found for any given date, time and location it will be possible to relate the direction of the shadows identified to the compass direction of the camera.

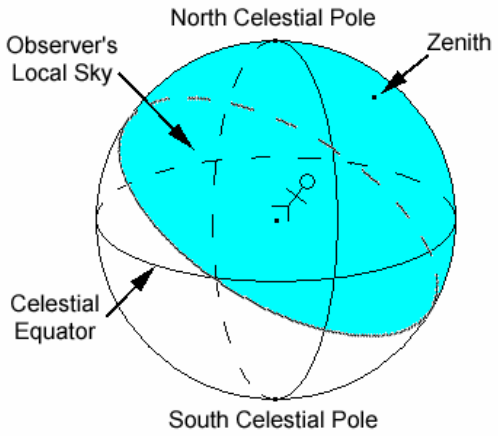

Figure 3.4: The celestial sphere and an observer's local sky

4 Shadow Identification

4.1 Existing Research

A great deal of research has been undertaken in the area of identifying shadows in images and video sequences [HORP 99, FUNG 01, FINL 02]. A very comprehensive evaluation of the different techniques in use can be found in [PRAT 01]. The paper serves as an overview of shadow detection algorithms, describing the principles behind each key technique. It also attempts to quantify their success in identifying shadows.

The primary classification of algorithms examined in [PRAT 01] is based on their "decision rules". An algorithm can be either *deterministic* or *statistical*. In the former case the status of a pixel or a region in an image has a binary value – it is either a shadow or it isn't. In statistical algorithms this "class membership" is described in terms of probabilities – for example, "it is 88% likely that a certain pixel is a shadow".

Deterministic algorithms are further classified as being either *model based* or *non-model based.* Model based algorithms have some understanding of the scene that they are operating on. For example, an algorithm might be developed specifically for the videos shot by a camera located on a bridge across a motorway. It could recognise shadows based on the fact that they all have similar shapes and move at similar speeds. The constraints demanded by such algorithms typically make them very successful, but they are complex, require much effort to implement and are not general-purpose. Non-model based algorithms, on the other hand, are much simpler while still achieving high success rates.

Statistical algorithms can be either *parametric* or *non-parametric*, depending on whether parameters and thresholds required for successful shadow detection must be provided manually or can be learned automatically. The parametric technique examined in [PRAT 01] makes the assumption that objects and shadows form compact regions. It then assigns to each such region a probability that it represents a shadow, and accepts only those that exceed a certain threshold. Crucially, a number of initial frames from the video sequence must be segmented manually into shadow and non-shadow regions, which is a significant investment of effort.

Statistical algorithms are almost always complex. They require a great deal of effort to implement compared to some of the much simpler deterministic algorithms, without achieving significantly better performance.

4.2 Algorithm Selection

Shadow detection is a vast area of research in Computer Vision. A simple algorithm is required to meet the time constraints of this project, even if this is at the expense of accuracy. Fortunately, the success rates of simple algorithms are almost as high as those of more complex ones [PRAT 01].

The solution to this project will take the form of extensions to the *Trinity Image Processing System* (TIPS), developed by Kenneth Dawson-Howe (TCD). A simple yet successful shadow detection utility was added to the system by Scott Tattersall (TCD Computer Science graduate, 2002). It can be classified as a *deterministic non-model based* algorithm. I have decided to use this utility as it identifies shadows sufficiently accurately for the purposes of this project. Its operation is described in the following sections. Time constraints make the implementation of other shadow detection techniques infeasible.

4.3 Background Image

As with many other algorithms for identifying shadows, this one requires a background image. Every frame is compared to this reference image, and each pixel in the frame is classified as moving object, shadow or background. In this project "moving objects" will typically be people walking through the scene. Figure 4.1 shows a background image (a), a frame from a video clip (b) and the moving objects and shadows that have been identified in the frame (c).

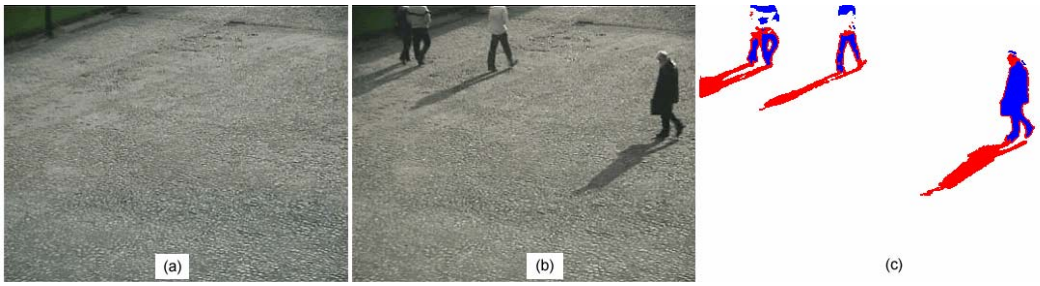

Figure 4.1: Background (a), current frame (b) and objects and shadows found (c)

A suitable background image is one containing no objects that are considered part of the foreground and no associated shadows. Depending on the amount of activity (or absence thereof) in the video sequence, an unaltered frame may be usable as a background image. (This is how the background image was obtained in Figure 4.1.) Alternatively, if there is no frame devoid of foreground objects in the entire video clip, a background image can be constructed by identifying the areas that don't change significantly across a number of frames. These areas are typically portions of the background, and by joining them together a full background image can be created. TIPS provides the feature "Update Stable Changes" for this purpose.

4.4 Colour Models

An explanation of colour models is required in order to understand how the shadow detection algorithm operates. Computer monitors display colour by combining red, green and blue light. They are thus said to operate in RGB. A colour image and its corresponding red, green and blue components are shown in Figure 4.2.

Figure 4.2: Colour image and its red, green and blue channels

There are, however, alternative ways of classifying colour. One such way is by defining every possible colour in terms of its hue, saturation and luminance [CARD 01]. Hue represents the "essence" of a colour, for example "reddish" or "greenish". Colours of the same hue can be described as "washed out" or "vibrant" – these terms are a reflection of a colour's saturation. Finally, some colours are brighter than others, a characteristic which is described by their luminance. The colour model used to encode this information is called HSV, with V representing the luminance channel. Figure 4.3 shows a depiction of HSV space as a cone. Luminance increases along the cone's vertical axis, saturation increases as one moves out from the centre and hue changes as one travels around the cone.

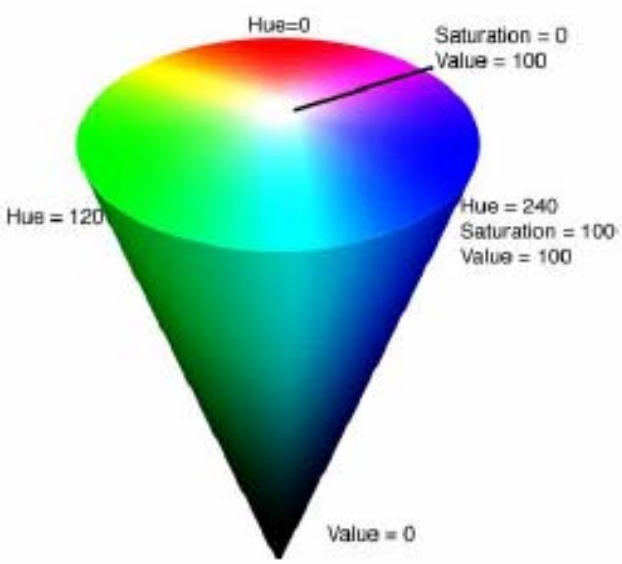

Figure 4.3: A depiction of HSV colour space, from [CARD 01]

The HSV colour model is important because it closely mirrors the human perception of colour. An area of background falling into shadow will have its HSV colour components changed in a predictable way (discussed in the next section). Figure 4.4 shows the saturation and luminance channels of the colour image in Figure 4.2 (high saturation is indicated by dark areas). Specific changes in these channels will be used to identify shadows.

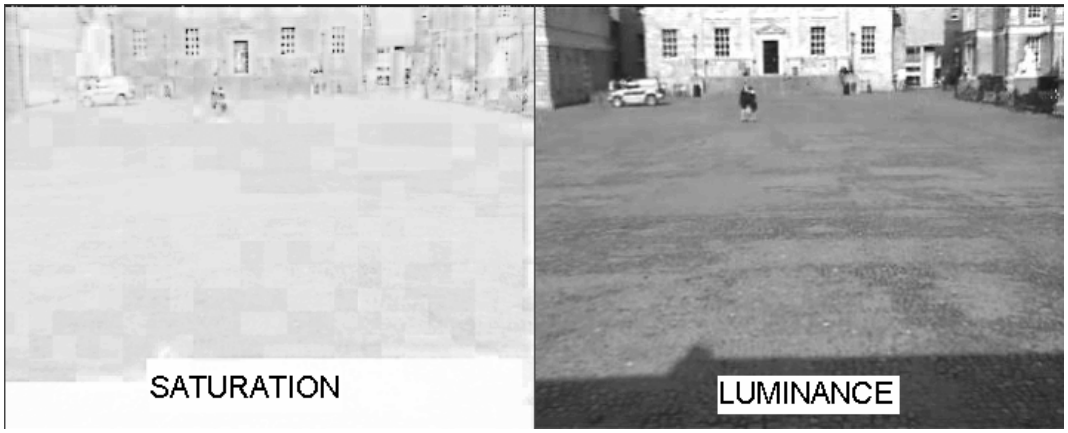

Figure 4.4: Saturation and luminance channels of a colour image

4.5 Algorithm

The algorithm presented here is the one implemented in TIPS by Scott Tattersall. It first identifies all pixels representing either moving objects or shadows in the frame of interest. A pixel is classified "moving or shadow" if it is sufficiently different to the corresponding pixel in the background image (in RGB terms). Some of these pixels are then identified as shadow pixels. The criteria a pixel must meet to be regarded as a shadow pixel are:

- its luminance must drop (by a limited amount) AND
- its saturation may rise only very slightly

The algorithm requires two parameters. The first ("ThresholdA") is the amount by which the pixel under consideration can differ from the corresponding background pixel before it is classified as "moving or shadow". Each of the pixel's three (RGB) channels is examined in turn. They must all fall within the threshold in order for the pixel to be deemed part of the background. The lower the threshold the more "moving or shadow" pixels will be found.

The second parameter ("ThresholdB") is the largest drop in luminance allowed such that a pixel can be regarded as a shadow. This number is given as a percentage. The higher the threshold the more shadow pixels will be identified.

A pseudo-code representation of the algorithm is presented below. This will help in understanding the algorithm while removing the complexities of data structures found in the real C++ implementation.

```
//frame_pixel = current pixel of frame of interest
//bg_pixel = current pixel of background image
FOR each pixel in the current frame and background image
    IF  |frame_pixel's R-channel  –  bg_pixel's R-channel| > ThresholdA  OR
    |frame_pixel's G-channel  –  bg_pixel's G-channel| > ThresholdA  OR
    |frame_pixel's B-channel  –  bg_pixel's B-channel| > ThresholdA
            Mark pixel as "moving or shadow"
    ENDIF

    //get the lowest luminance this pixel can have to be a shadow
    lowest_luminance = bg_pixel's luminance x (100 – ThresholdB) / 100

    SMALL_INCREASE = 30          //saturation can increase slightly

    IF  pixel is "moving or shadow"  AND
    frame_pixel's luminance < bg_pixel's luminance  AND
    frame_pixel's luminance > lowest_luminance  AND
    frame_pixel's saturation < bg_pixel's saturation + SMALL_INCREASE
            Mark pixel as shadow
    ENDIF
NEXT
```

Although the algorithm above had already been implemented in TIPS, it contained a serious bug that impaired its usability. The TIPS environment allows multiple image processing operations, e.g. shadow detection, to be performed on the same image or video clip (and on the resulting images). In order to avoid repetition the user can save a sequence of such operations to a workspace file. Later, the same sequence of operations can be applied to the video by simply opening the workspace file. Unfortunately, workspaces containing the shadow detection operation could not be saved – attempting to do so resulted in TIPS crashing. I successfully found the cause of this bug and rectified the problem. This meant that a long sequence of operations could be saved to a workspace file, which allowed different strategies for the project to be tried quickly.

4.6 Problems

4.6.1 Isolated Object and Shadow Pixels

The objects and shadows found in Figure 4.1 (c) are very "clean", i.e. there are very few isolated points where background pixels have been misclassified as moving objects or shadow. Figure 4.5 shows an example of shadow detection where this is not the case.

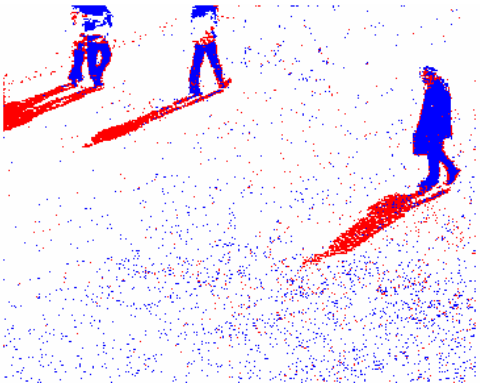

Figure 4.5: Shadow detection resulting in many isolated points

In order to achieve the clean results of Figure 4.1 it was first necessary to apply a certain image processing operation to both the background image and the frame of interest. The operation in question is *averaging*. It works by taking a square neighbourhood of pixels centred on each pixel in the image. The corresponding pixel in the resultant image is the average of all the pixels in the neighbourhood. The operation has the effect of blurring the image to which it is applied. The width of the square neighbourhood must be specified. A value of 3 was found to remove isolated points very successfully.

4.6.2 False Shadow

Another problem with the shadow identification algorithm is its strong tendency to detect (incorrectly) thin areas of shadow around regions representing people. These false shadow pixels can be seen in Figure 4.1 (c) – the people appear to be surrounded by a thin border of shadow. This would later cause difficulties when finding the direction of the shadow regions. A technique for dealing with the problem is presented in Chapter 5.

5 Shadow Direction

Once the shadows cast by people in the video sequence have been identified it is necessary to associate a direction with them. As explained in section 2.1 all shadows cast on the ground have the same compass direction (azimuth) as the Sun. All such shadows are parallel because the Sun is at a very great distance from the Earth. Figure 5.1 illustrates that in a 2D image these shadows appear, if extended along their direction, to meet at a point on the horizon. This point, which may fall outside the image, is referred to as the "vanishing point". The mathematics of the following chapter reveals that finding the vanishing point is central to discovering the camera's compass direction.

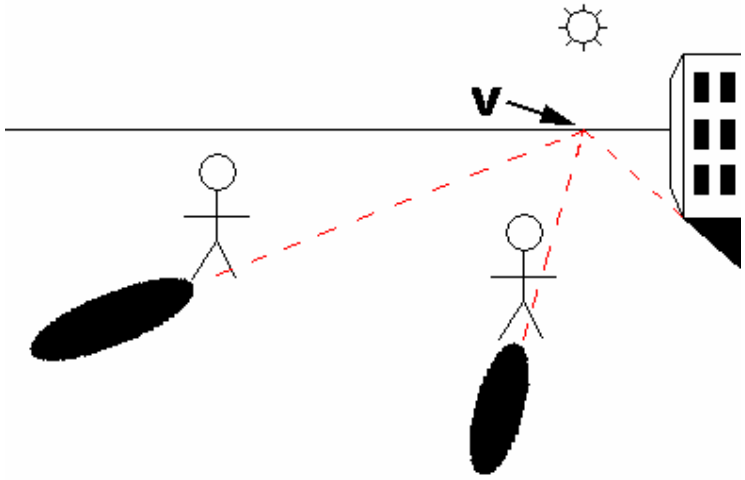

Figure 5.1: All ground shadows meet at a point V on the horizon

The vanishing point for a given video clip is theoretically found by calculating the common intersection of all the shadows (see Figure 5.1). Unsurprisingly, such a simple approach cannot be used because it is impossible to associate an exact direction with each shadow. Furthermore, it is possible for the shadow identification algorithm to detect false shadows or to miss portions of shadows, resulting in very inaccurate directions being determined (see Figure 5.2). To overcome these problems evidence from all of the shadows must be combined so that the vanishing point can be located robustly. The precise manner in which this is done is described in section 6.4.

Figure 5.2: Poor shadow detection leading to inaccurate directions

5.1 Identification of Regions

It is useful to think of each shadow identified in a video frame as a separate region of a black and white (binary) image (see Figure 5.3). The goal is to find the direction of each region. A formula is given in [SONKA 99] that accomplishes this, but it requires a binary image containing only one region to work. It was necessary to develop an algorithm in order to use the formula to find directions for multiple shadow regions. The algorithm is given in the next section.

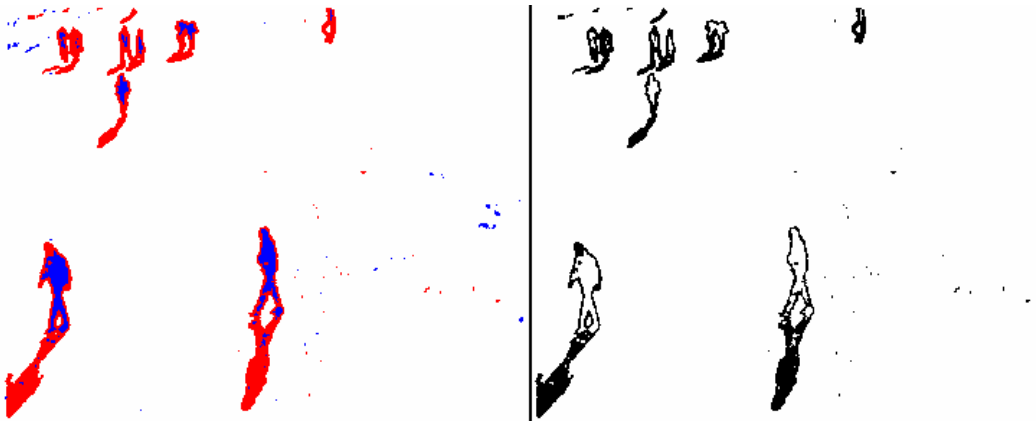

Figure 5.3: Moving objects and shadows and a binary shadow image

Because there are multiple shadow regions in a video frame each shadow pixel must be labelled with the shadow to which it belongs. Connected Components Analysis (CCA), which is provided by TIPS, can be used for this purpose. CCA gives all pixels in a connected region the same label, i.e. all pixels that are "touching" are grouped together

into one region. It can only operate on binary images. Figure 5.4 shows the results of CCA applied to a binary shadow image (each pixel's label is indicated by its colour).

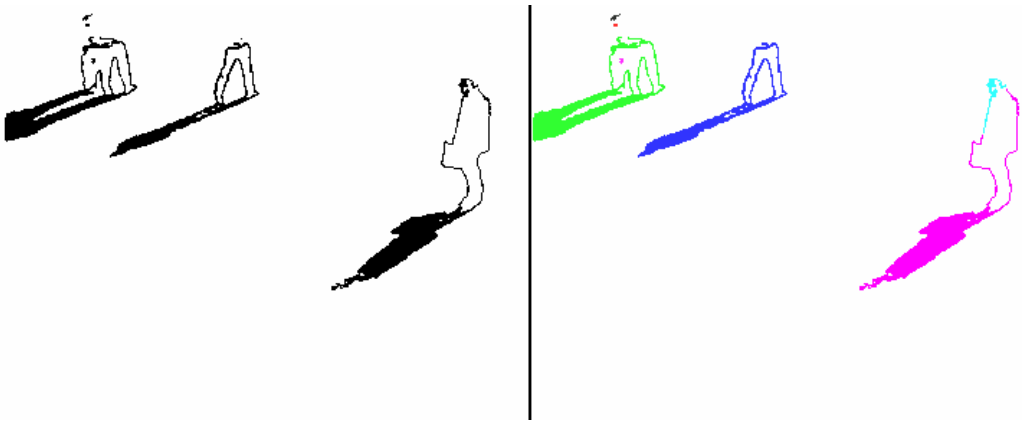

Figure 5.4: Binary shadow image and the results of CCA

5.2 Direction of a Region

The direction of a region can be defined as the direction of the longer side of the region's smallest bounding rectangle (see Figure 5.5). However, this approach cannot be used to find a region's direction – it requires too much computation. It would be necessary to find multiple bounding rectangles, each with a slightly different direction, before picking the one with minimum area. A faster technique is needed.

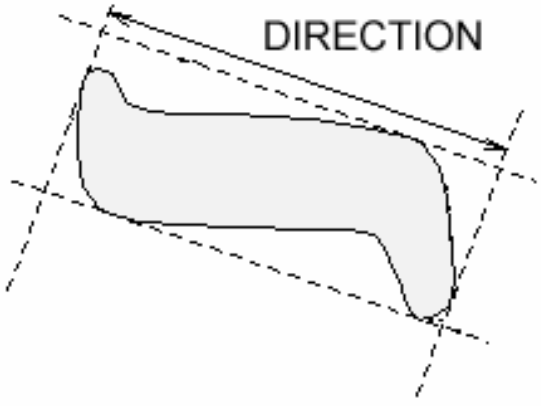

Figure 5.5: Region's bounding rectangle yields direction, from [SONKA 99]

Every region has a centre of gravity. This can be thought of as the point on which the region would balance if it were a piece of card. Its co-ordinates (x_c, y_c) are given by

$$x_C = \frac{m_{10}}{m_{00}} \qquad y_C = \frac{m_{01}}{m_{00}}$$

m_{pq} is a spatial moment of order (p + q), which is given by

$$m_{pq} = \sum_{row} \sum_{col} col^p row^q f(col, row)$$

where

$$f(col, row) = \begin{cases} 1 \text{ if pixel at (row, col) is part of current region} \\ 0 \text{ otherwise} \end{cases}$$

Once the centre of gravity of a region has been obtained it is possible to compute central moments for the region. The central moment μ_{pq} is given by

$$\mu_{pq} = \sum_{row} \sum_{col} (col - x_c)^p (row - y_c)^q f(col, row)$$

where *f(col,row)* is defined as above.

Finally, the direction θ of a region can be computed using central moments [SONKA 99]:

$$\theta = \frac{1}{2} \tan^{-1} \left(\frac{2\mu_{11}}{\mu_{20} - \mu_{02}} \right)$$

It is important to note that, since the algorithm presented below traverses the image on a pixel-by-pixel basis, two passes over the image are required. The first is used to compute spatial moments for each region in the image, which allows each of their centres of gravity to be calculated. Only then is it possible to compute *central* moments for each region, from which the direction can be calculated. The algorithm requires as input a binary image of the shadows, along with the results of Connected Components Analysis performed on this image. The latter allows the algorithm to associate a particular shadow region with each pixel.

```
FOR each row of the image
    FOR each column of the image
            Get shadow region to which the pixel belongs  //from CCA image
            IF pixel belongs to a region
                    //build up the region's spatial moments
                    Accumulate m00 of region
                    Accumulate m01 of region
                    Accumulate m10 of region
            ENDIF
    NEXT column
NEXT row

//each region's centre of gravity can now be calculated
FOR each shadow region of the image
    Xc = m10 / m00
    Yc = m01 / m00
NEXT shadow region

FOR each row of the image
    FOR each column of the image
            Get shadow region to which the pixel belongs  //from CCA image
            IF pixel belongs to a region
                    //build up the region's central moments
                    //using its centre of gravity
                    Accumulate μ02 of region
                    Accumulate μ11 of region
                    Accumulate μ20 of region
            ENDIF
    NEXT column
NEXT row

//each region's direction can now be calculated
FOR each shadow region of the image
    θ = 0.5 x atan2( 2 x μ11, μ20 – μ02 )
NEXT shadow region
```

Figure 5.6 shows several shadow regions and their associated directions, represented as line segments. These line segments, which are meant to point in the direction of the Sun's azimuth, pass through the centre of gravity of their shadow region. (Note: in order to find each region's direction accurately it was necessary to "clean up" the binary shadow image, using the technique described in the next section.)

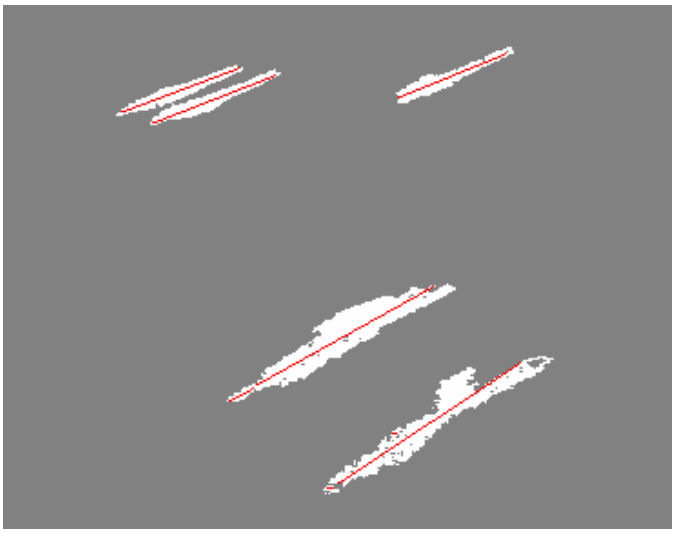

Figure 5.6: Shadow regions overlaid with their associated directions

5.3 Removal of False Shadow

As explained in section 4.6.2 the shadow detection algorithm often incorrectly detects shadow pixels surrounding the bodies of people in the video sequence. Since these false shadow pixels distort the shape of the shadow region significantly, the associated direction will be wildly inaccurate (see Figure 5.7).

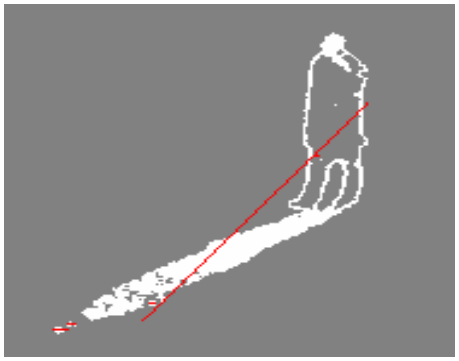

Figure 5.7: Distorted shadow region yielding inaccurate direction

In order to improve the accuracy of the shadows' directions it was necessary to remove the thin areas of false shadow that were distorting the shape of the regions. This was achieved through the use of an "opening" operation (provided by TIPS) on each frame's binary shadow image. Opening operates by first "eroding" pixels from around the edges of regions. It then "dilates" the remaining regions by (effectively) adding pixels around their perimeters. Since the erosion stage will completely destroy thin

parts of a region, the net effect of opening on binary shadow images is to remove a high proportion of false shadow while leaving the areas of true shadow virtually unchanged (see Figure 5.8).

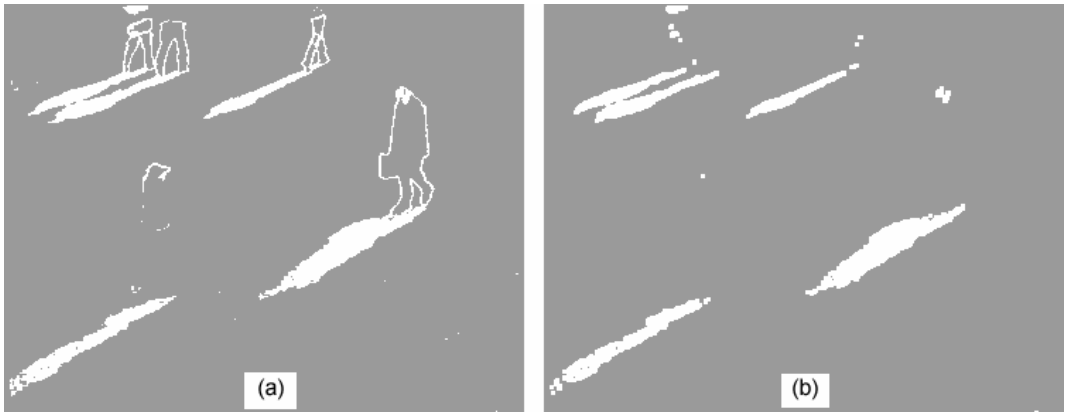

Figure 5.8: Original binary shadow image (a) and results of opening (b)

5.4 Excluding Unreliable Line Segments

As shown in Figure 5.6 line segments can be used to represent each shadow region's direction. These line segments pass through their region's centre of gravity. An imaginary rectangle bounding each shadow region limits their length. In spite of the technique employed in the previous section to extract shadow directions accurately, many line segments simply do not point in the direction of the Sun's azimuth (see Figure 5.9). Many of these "unreliable" line segments are produced by the shadow detection algorithm failing to detect shadows properly.

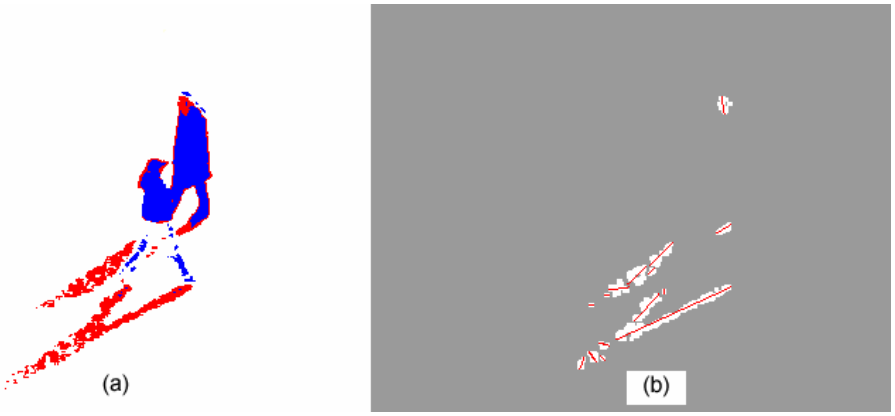

Figure 5.9: Poor shadow detection (a) producing unreliable line segments (b)

The approach used for finding the vanishing point (see section 6.4) is sensitive to the accuracy of the shadows' directions. In order for the vanishing point to be located robustly it is necessary that as high a proportion as possible of the line segments used be accurate. To this end an algorithm for identifying "reliable" line segments, i.e. those representing the shadows of people, was devised. It is presented below.

The idea behind the algorithm is that, if a line segment is observed in a similar position in the image over the course of several frames, having also similar length and direction during this time, it is very likely to represent the shadow of a person. Since this is the kind of line segment needed to find the vanishing point, it is marked as "reliable". As people move through the scene, the position, length and direction of their shadow line segments will change slightly from one frame to the next. It is therefore necessary to provide tolerances when deciding whether or not line segments from subsequent frames represent the same shadow. It should be noted that "tracking" of this sort across multiple frames requires very complex data structures and searching methods. Significant effort was expended in their implementation. A simplified pseudo-code version of the algorithm is given below.

```
//NUM_FRAMES = number of frames for which a line seg must meet similarity
//constraints before it is considered "reliable"

FOR each line seg in the current frame
    Search for similar line seg in previous frame
    //"similar" means length, direction and position are all within tolerances
    IF similar line seg found
            Associate this line seg with similar one
            //this creates a link in a chain of similar line segs (across frames)
    ENDIF

    Search for unbroken chain of associations across last NUM_FRAMES frames
    //a chain with NUM_FRAMES links implies oldest line seg is reliable
    IF unbroken chain found
            Mark oldest line seg in chain as reliable
            Save its position and direction  //needed to find vanishing point
    ENDIF
NEXT

Discard line seg data for oldest frame in buffer
```

Figure 5.10 shows the results of the algorithm. In (a) both reliable and unreliable line segments are shown, while (b) shows only those which pass the tests for reliability. It

can be seen that the algorithm has succeeded in significantly increasing the proportion of line segments that point in the direction of the Sun's azimuth.

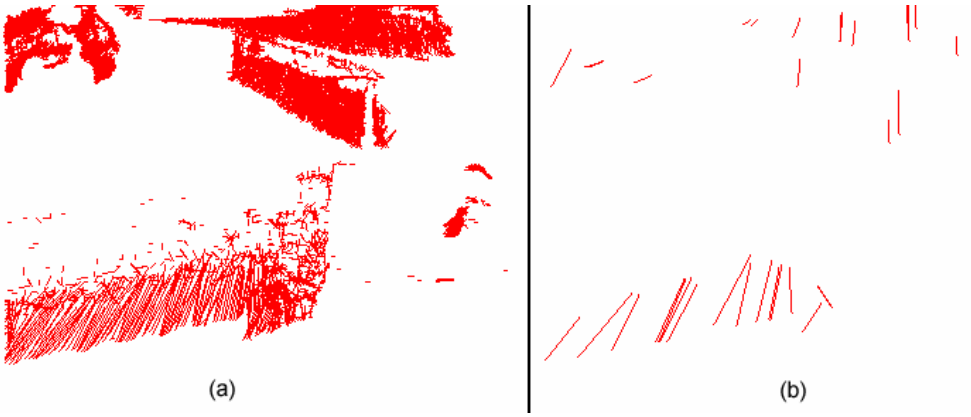

Figure 5.10: All line segments from a video (a) and reliable line segments only (b)

6 Camera Direction Inference

In Chapter 3 the compass direction of the shadows was established. Before the ultimate goal of the project can be achieved, i.e. finding the *camera's* compass direction, the relative orientation of the shadows and the camera must be determined (see Figure 6.1). The present chapter describes the mathematics and the camera models used to solve the problem.

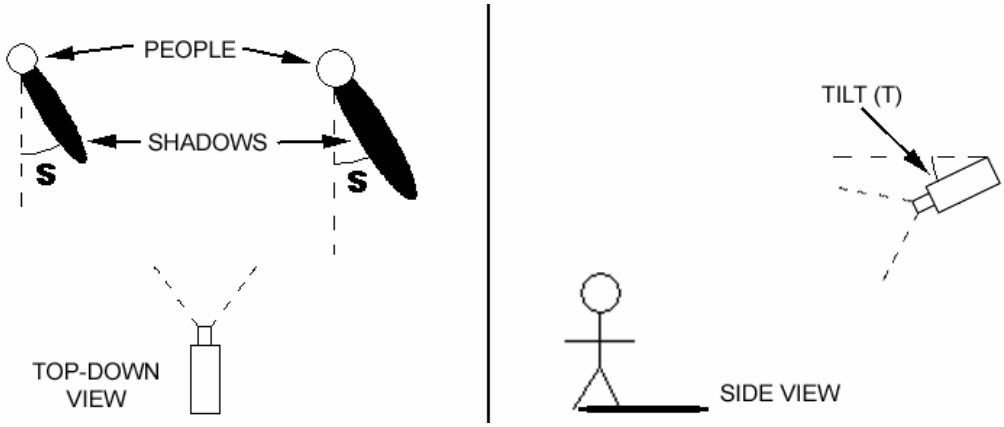

Figure 6.1: Arrangement of camera illustrating relative orientation to shadows (S)

6.1 Co-ordinate Spaces

Most cameras can accurately be described by the pinhole model [SONKA 99]. Rays from objects in the scene pass through the focal point ("pinhole") and are projected onto the image plane (see Figure 6.2). Different co-ordinate spaces are used in order to transform objects in the world into 2D perspective projections. (In Figure 6.2 each of these spaces is represented by a set of mutually perpendicular axes.) Co-ordinates in world space (centred on O_W) are first transformed into camera space (centred on the pinhole O_C). The new co-ordinates are then converted to image space, whose origin O_i is located at the top-left corner of the image.

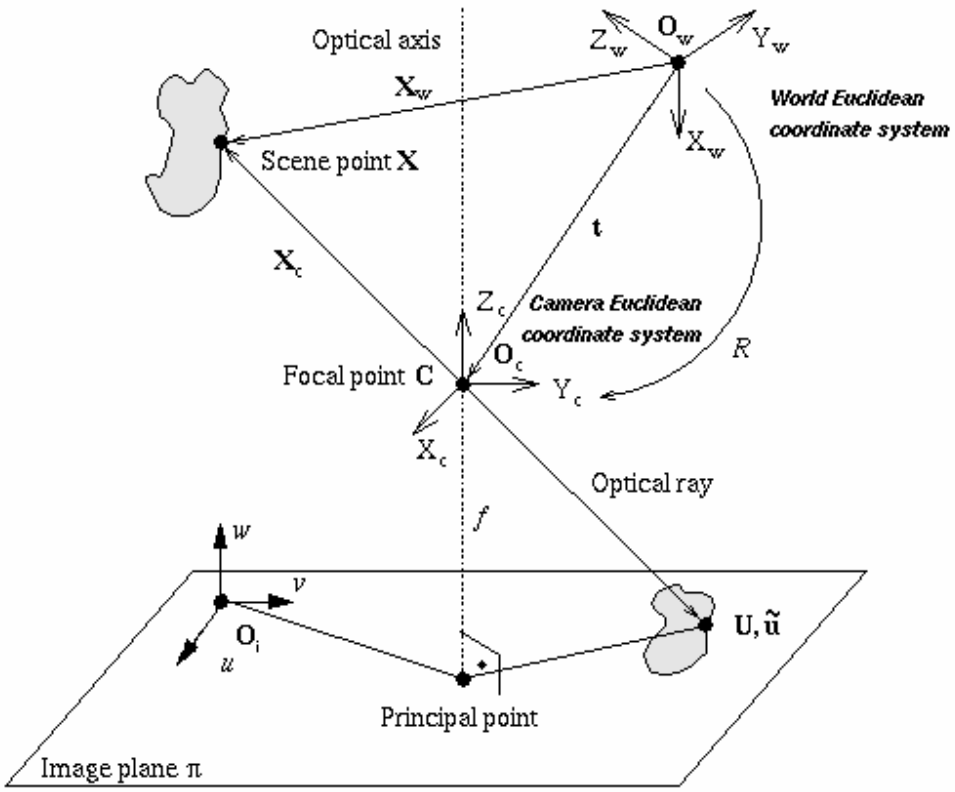

Figure 6.2: Co-ordinate spaces used in a pinhole camera model, from [SONKA 99]

6.2 Camera Orientation and Parameters

In order to establish the relative orientation of the camera and the shadows it is necessary to "back-project" the shadows' line segments from image space into world space. The transformations from one co-ordinate space to another are efficiently described by the multiplication of matrices. If $\mathbf{X_W}$ represents a point in world space, then its projection **u** in the image is given by the formula

$$\mathbf{u} = [\ KR \mid -KR\mathbf{t}\ ]\ \mathbf{X_W} \qquad (6.1)$$

where both **u** and $\mathbf{X_W}$ are in homogeneous co-ordinates (which, crucially, allow directions as well as points to be expressed). Both K and R are 3 × 3 matrices, **u** and **t** are 3-vectors and $\mathbf{X_W}$ is a 4-vector. K, R and **t** are described below.

6.2.1 Extrinsic Parameters

Between them, K, R, and **t** contain 11 unknowns that must either be found or solved so that the camera's relative orientation to the shadows, S, can be established. R and **t** describe the camera's position and orientation in world space, and so their elements define the camera's *extrinsic parameters*. The vector **t** represents the displacement between O_W and O_C, the origins of world space and camera space respectively (see Figure 6.2). In this project distances play no role – only directions are important. It is valid to place O_W anywhere, but by having it coincide with O_C the 3 components of the vector **t** become 0. Three of the unknowns have been eliminated.

In order to make the problem tractable it was necessary to assume that the camera was level, i.e. that the horizon would appear as a horizontal line in the video sequence. It is also valid to regard the camera as being aligned with the Z-axis of world space (see Figure 6.3), since the concept of "forward" is relative to the observer. These considerations make for quite a simple rotation matrix R, which describes how world space and camera space are oriented relative to one another. The matrix contains only one unknown: the downward tilt T of the camera. The structure of R [FOLEY 90] is

$$R = \begin{bmatrix} 0 & \cos T & -\sin T \\ 1 & 0 & 0 \\ 0 & -\sin T & -\cos T \end{bmatrix} \qquad (6.2)$$

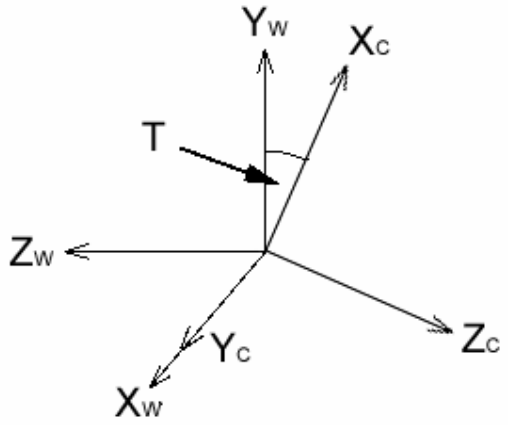

Figure 6.3: The world and camera co-ordinate spaces for this project

6.2.2 Intrinsic Parameters

The matrix K describes the internal ("intrinsic") parameters of the camera, e.g. its focal length and aspect ratio. The structure of K, which contains 5 unknowns, is

$$K = \begin{bmatrix} \alpha_u & \alpha_{shear} & -u0 \\ 0 & \alpha_v & -v0 \\ 0 & 0 & 1 \end{bmatrix} \quad (6.3)$$

The principal point of the camera has co-ordinates (u_0, v_0) in image space (see Figure 6.2). It is reasonable to approximate it by the co-ordinates of the centre of the image. The parameter α_{shear} applies only to cameras that produce skewed images, which are very unusual. It therefore has the value 0 for this project. Finally, α_u and α_v represent the amount by which the image is scaled in the vertical and horizontal directions. (Both α_u and α_v are proportional to the camera's focal length.) The aspect ratio of the camera is given by α_v/α_u. The video sequences used in this project all have an aspect ratio of 1:1. It seems valid to assume this value for the present, which means that $\alpha_u = \alpha_v = \alpha$. The matrix K now contains only one unknown, α.

6.3 Back-projection

By using one correspondence between image space and world space it is possible to relate or eliminate the unknowns α, T in order to find S, the crucial angle that measures the relative orientation of the camera and the shadows (see Figure 6.1). The simplest correspondence to use is the observation that the vanishing point of all the shadows (see Figure 6.4, point 'V') is the projection of the vector in 3-space representing their common direction. (Vectors can be regarded as points at infinity.) It is necessary to "back-project" the vanishing point into this vector in world space.

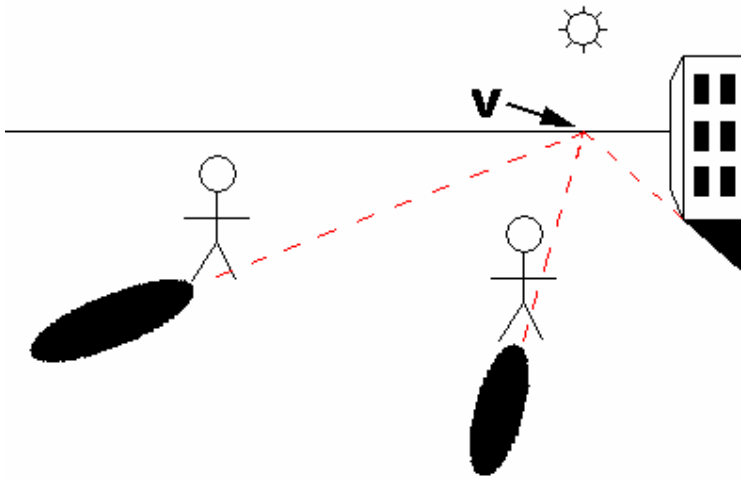

Figure 6.4: Vanishing point is the projection of the shadows' direction

To help in simplifying the mathematics of back-projection the origin of the image should be translated to the principal point, i.e. the centre of the image. The same translation must be applied to the vanishing point. This procedure results in (u_0, v_0) becoming (0, 0).

Since the shadows are cast on the ground (where y = 0), the vector $\mathbf{X_W}$ representing the shadows' common direction can be given in homogeneous 3-space as

$$\mathbf{X_W} = [\ \tan S,\ 0,\ 1,\ 0\ ]^T \tag{6.4}$$

If the co-ordinates of the vanishing point in image space (after the translation described above) are (u_{vp}, v_{vp}) the equation

$$\mathbf{u} = [\ \mathrm{KR}\ |\ -\mathrm{KR}\mathbf{t}\ ]\ \mathbf{X_W} \tag{6.5}$$

expands to

$$\beta \begin{bmatrix} u_{vp} \\ v_{vp} \\ 1 \end{bmatrix} = \left[\begin{bmatrix} -\alpha & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & \cos T & -\sin T \\ 1 & 0 & 0 \\ 0 & -\sin T & -\cos T \end{bmatrix} \middle| \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right] \begin{bmatrix} \tan S \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{6.6}$$

since the vector **t** is zero. (It was necessary to change the sign of one element in the matrix K to account for the inversion of image space. The variable β is needed because of the use of homogeneous co-ordinates.)

It is necessary to solve equation (6.6) for S. Multiplying out the matrices yields the equations

$$\beta u_{vp} = \alpha \sin T \tag{6.7}$$

$$\beta v_{vp} = \alpha \tan S \tag{6.8}$$

$$\beta = -\cos T \tag{6.9}$$

Equation (6.9) can be used to eliminate β from equations (6.7) and (6.8):

$$-\cos T\, u_{vp} = \alpha \sin T \tag{6.10}$$

$$-\cos T\, v_{vp} = \alpha \tan S \tag{6.11}$$

Doing so reveals that, in order to find S, either α or T is required. Since all information easily extractable from the video sequence has been used at this point, the user of the system must provide the value for one of the two remaining unknowns, i.e. α or T. The scaling factor of the camera, α, is an extremely unintuitive quantity, whereas the camera's downward tilt, T, is much more meaningful. Therefore, the value of T must be provided by the user, allowing α to be eliminated from equations (6.10) and (6.11).

$$\begin{aligned} \frac{-\cos T\, u_{vp}}{\sin T} &= \frac{-\cos T\, v_{vp}}{\tan S} \\ \Rightarrow \tan S &= \frac{\sin T\, v_{vp}}{u_{vp}} \end{aligned} \tag{6.12}$$

6.4 Vanishing Point Estimation

The mathematics of the previous section reveals that the "vanishing point" of the shadows must be known in order for the relative orientation of the camera and the shadows to be determined (equation (6.12)). The vanishing point is found, theoretically, by calculating the intersection of any two shadow line segments (see Figure 6.4). The use of only two line segments is not appropriate, however. Line segments can

sometimes be unreliable, in spite of the technique for eliminating them described in section 5.4.

6.4.1 Unsuccessful Approaches

The first approach to locating the vanishing point was to find the "best" intersection point for all of the line segments. The set of equations representing the line segments was solved using the "least squares" method (see Appendix B). Unfortunately, the least squares solution proved to be extremely inaccurate as an approximation to the vanishing point. It would appear that the technique is highly sensitive to the linear equations used. The presence of even a small number of unreliable line segments completely destroyed the accuracy of the result.

The next approach tried was to find the intersection of every pair of line segments, so that the vanishing point could be found by a "majority vote". The image space was partitioned into equal-sized "bins" in both the horizontal and vertical directions (see Figure 6.5). The overlap of the horizontal and vertical bins that contain the most intersection points is deemed to be the vanishing point. The results of this strategy were also poor, as the distribution of votes across image space lacked the "spike" that would characterise the vanishing point's location (as in Figure 6.7).

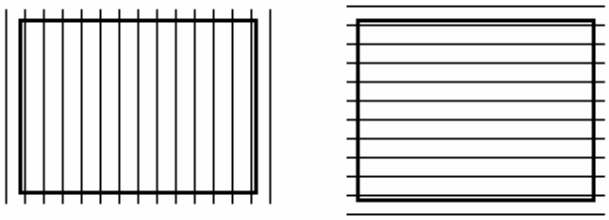

Figure 6.5: Partitioning of image space into equal-sized bins

6.4.2 TanBin Partitioning

The fundamental problem with the previous approach was the partitioning of image space into equal-sized bins. If the vanishing point is located far outside the image boundaries, the votes cast for it will be scattered amongst the surrounding bins. The

problem is analogous to throwing something into one of several containers placed far away – even a slight inaccuracy in the strength or direction of the throw will result in the object entering the wrong container.

The solution required that the size of bins should increase the further they were from the centre of the image. In order to partition the image space in this manner the mathematical tan( ) function was used (see Figure 6.6). In addition to meeting the requirement of increasing bin size, it has the advantage of partitioning the infinite image space into a finite number of bins, which I have called "TanBins".

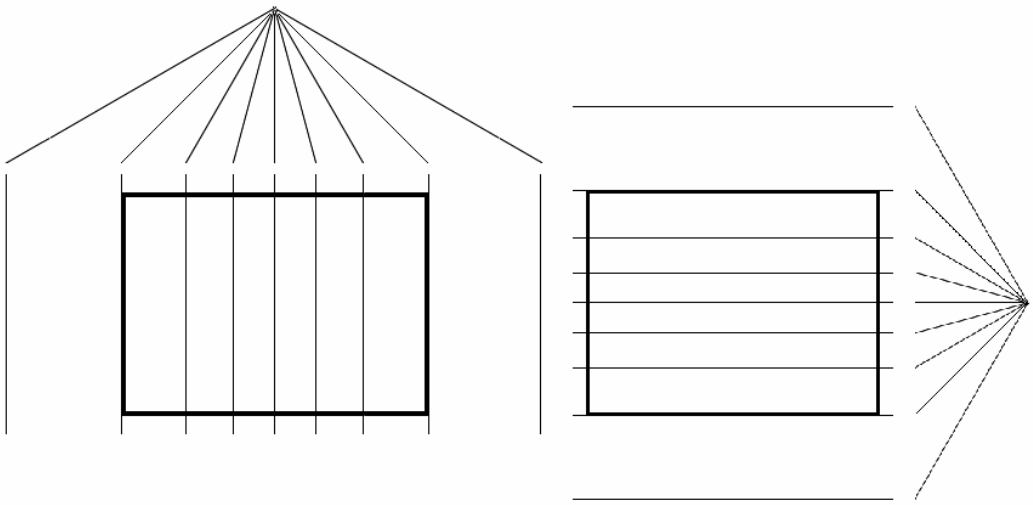

Figure 6.6: Partitioning of image space into TanBins

The approach was very successful in allowing the vanishing point to be located. The graphs of Figure 6.7 both exhibit very sharp spikes, identifying bins that contain a large number of votes. The centres of these bins are used to obtain a very good approximation to the vanishing point.

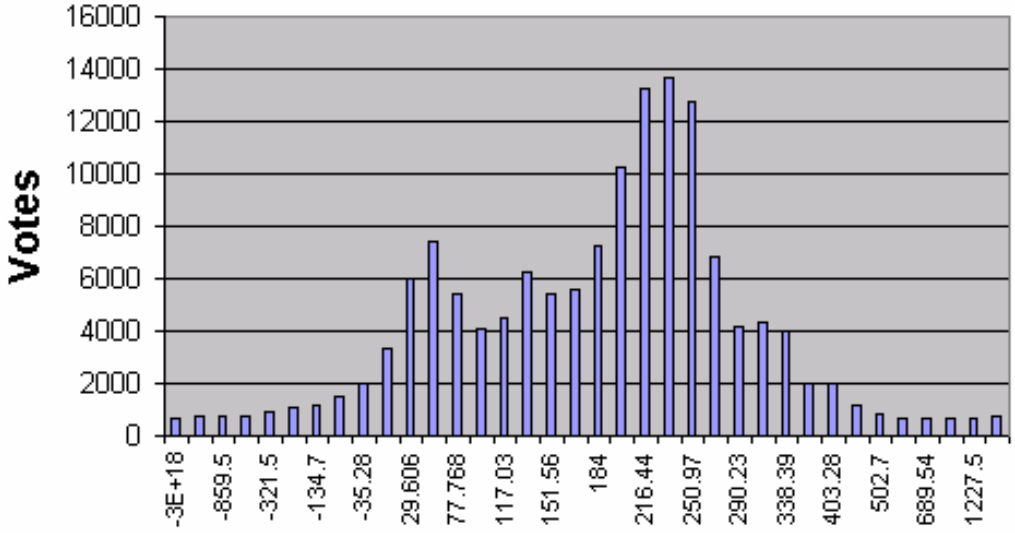

Figure 6.7 (a): Proposed horizontal ordinate of VP

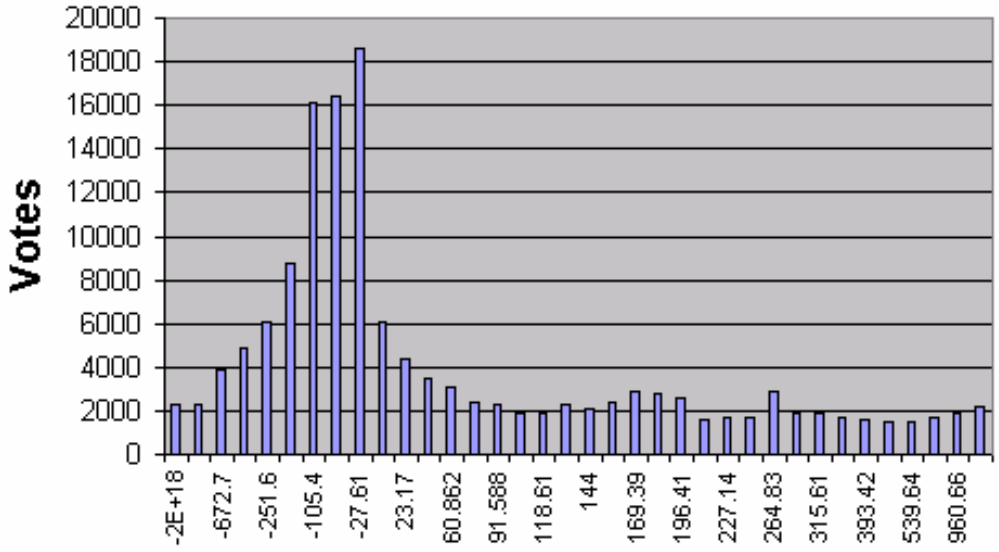

Figure 6.7 (b): Proposed vertical ordinate of VP

6.5 Compass Direction of Camera

The mathematics of section 6.3 yields the relative orientation of the camera and the shadows (S). The astronomical calculations of Chapter 3 provide the compass direction of these shadows. The *camera's* compass direction is then found by the simple formula

$$\text{compass direction of camera} = \text{shadows' compass direction} + S \qquad (6.13)$$

It may be necessary to reduce the answer to a value between 0 and 360 degrees.

6.5.1 Virtual Compass

In order to display the central result of the project, i.e. the camera's compass direction, in a meaningful way, a virtual compass was drawn on the video sequence. Equation (6.1) was used to perform the projection, with the same R and K matrices as those given in equations (6.2) and (6.3). This makes the compass appear to be on a plane parallel to the ground. It was necessary to modify the vector **t** in order to place the compass at some distance from the camera. A series of very short line segments were used to make up the circle of the compass, before the North-South and East-West lines were drawn. Finally, the letters N, S, E and W were added to label the principal directions of the compass. Figure 6.8 shows the results of the projection. I feel that it is an effective means of presenting the project's main result as it also conveys the downward tilt of the camera.

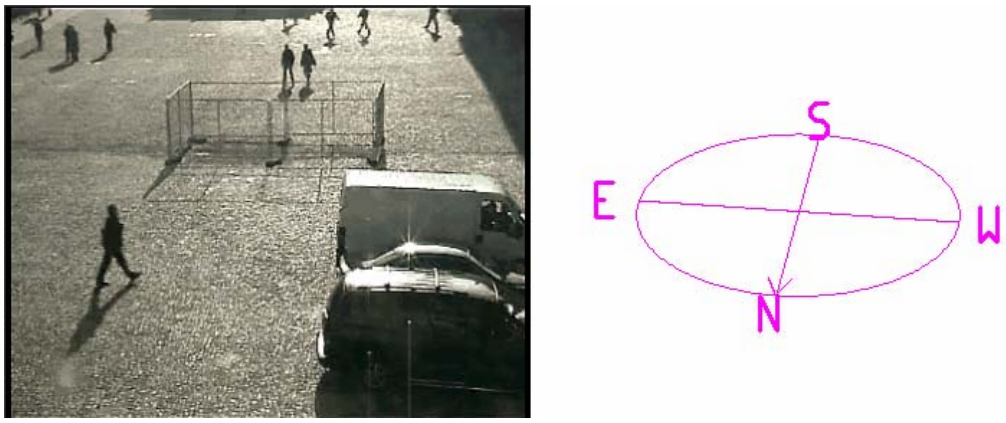

Figure 6.8: A video frame and a virtual compass depicting the camera's direction

7 Evaluation

This chapter serves as a review of the project. The results of the algorithm to determine the camera's compass direction are first presented. Elements of the project that were successful and difficulties that were encountered are subsequently discussed. Finally, ways in which the project could be developed further are outlined.

7.1 Results

Although the central result of the project, i.e. the camera's compass direction, can be expressed as a single number, presenting the results visually is much more informative. The CD accompanying this report contains the video sequences that were used to test the algorithm. A composite video of results accompanies each sequence. This shows (in addition to the original video) the identification of moving objects and shadows, the reliable line segments that have been found and the virtual compass that has been determined for the scene. Figures 7.1 and 7.2 show results from various sections of the project for the first video sequence. Similar results for the second video are presented in Figures 7.3 and 7.4.

Figure 7.1: Frame from 1st video (a) and objects and shadows identified (b)

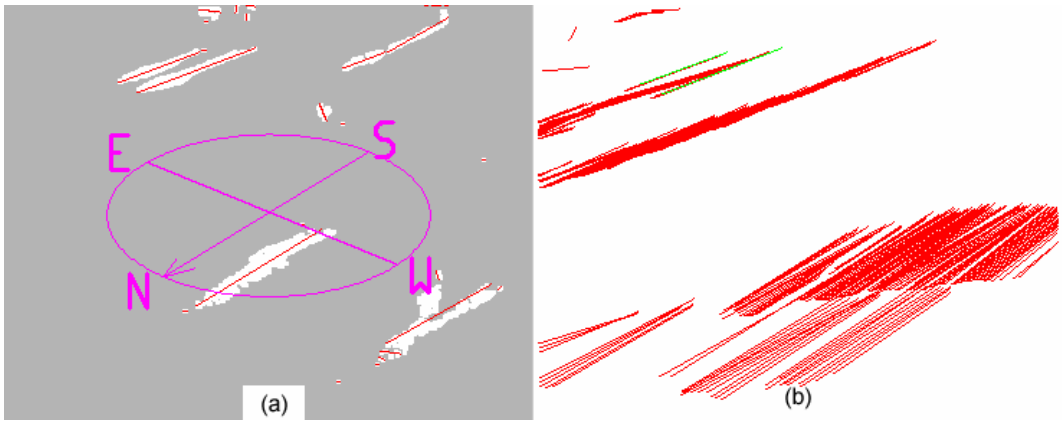

Figure 7.2: Virtual compass and line segments found for frame shown above (a); Reliable line segments from all previous frames (b)

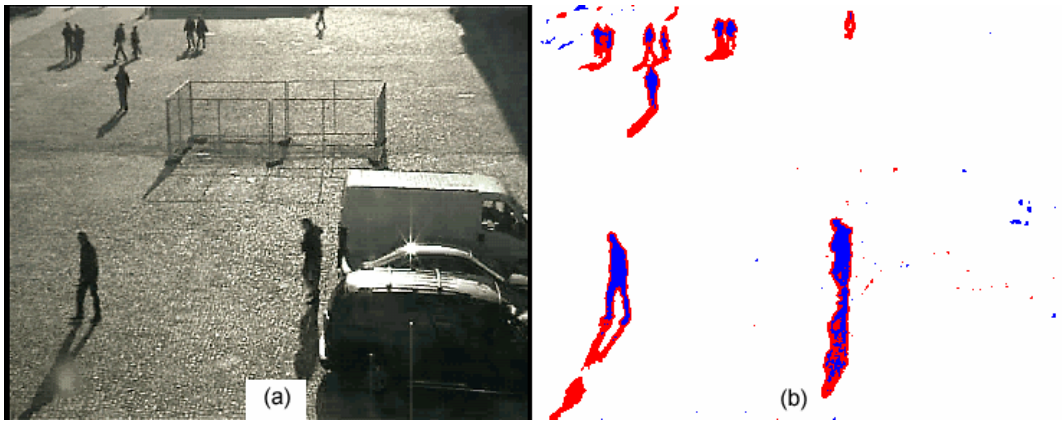

Figure 7.3: Frame from 2nd video (a) and objects and shadows identified (b)

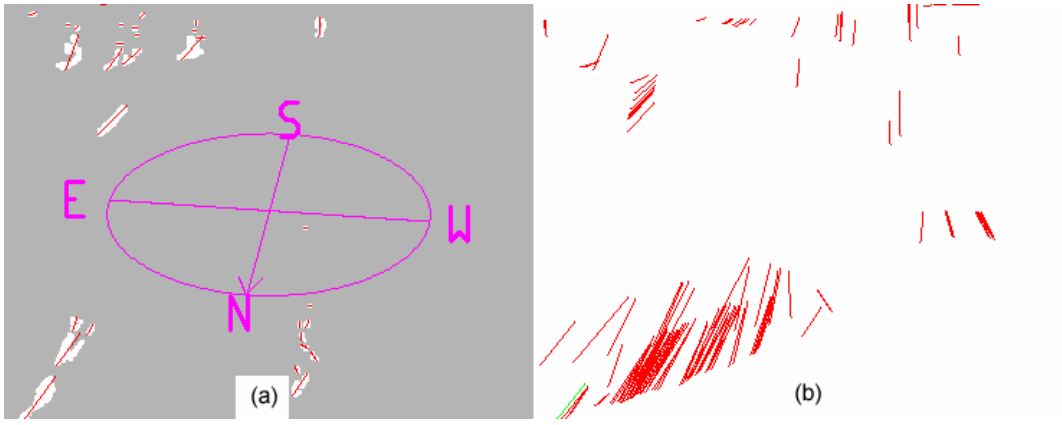

Figure 7.4: Virtual compass and line segments found for frame shown above(a); Reliable line segments from all previous frames (b)

It is not surprising that the virtual compass determined for each of the video sequences changes it direction on occasion, especially early in the sequence. This is a consequence of new reliable line segments being found as people pass through the scene. Each new line segment adds to the number of votes cast for the location of the vanishing point. With each new video frame the location stabilises, since the variability in the voting is decreasing.

In spite of the slight instability in the orientation of the compass it is correctly determined for the vast majority of each video sequence. (Since the sequences were shot in a known location – Front Square, Trinity College Dublin – it is possible to estimate the camera's direction empirically.) The directions determined are, however, accurate only to the resolution provided by the size of the TanBins.

7.2 Successful Aspects of the Project

7.2.1 Inferring the Camera Direction

The main aim of this project – to infer the compass direction of the camera from the shadows in the video sequence – has been achieved. However, in order to accomplish this it has been necessary to obtain from the user, or to assume, certain parameters. For example, the user of the system must provide the downward tilt of the camera. The camera's aspect ratio is taken as 1:1 and its principal point is assumed to coincide with the image centre. It was not possible in the time available to develop an algorithm capable of determining all of this information. It should be noted that none of the assumptions made are unreasonable. The application successfully utilises multiple pieces of information to produce a very useful result – the direction of the camera.

This project is, apparently, innovative in its use of astronomical data. Examination of the existing literature suggests that the Sun's position has never been used to provide directional information in a Computer Vision application. If this is the case, the project uses a piece of scene information in a unique manner.

7.2.2 Application Design

The structure of the project allowed it to be broken down into very specific modules, which were developed and tested independently of one another. For example, the results of the astronomical calculations in Chapter 3 were tested against those generated by a Web-based utility [NOAA 03]. This work had no impact on, for example, the techniques described in Chapter 5 for finding reliable shadow line segments. Only at the very end was it necessary to bring the results of the different sections together in order to calculate the camera's compass direction. By taking this approach it was possible to isolate the problems in Computer Vision that needed to be addressed (e.g. identifying shadows accurately) from those in other areas (e.g. finding the direction of a region).

In the initial algorithm the problems caused by unreliable line segments were not envisaged. Their existence made it necessary to develop a complex module for tracking line segments across several frames. Both the data structure and the algorithm proved immensely difficult to implement, as the specifics were unique to this project. Its completion, however, allowed the vanishing point to be located accurately. Without this module the results produced by the application would have been very unstable.

7.3 Difficulties Encountered

There were, unsurprisingly, a number of difficulties which arose during the course of the project.

A bug in the TIPS development environment prevented workspaces featuring shadow detection from being saved. This inconvenience slowed the project development significantly until it was corrected. Further problems with the saving of composite videos made the collection of results an arduous task.

The problems caused by unreliable line segments have been described already. Despite the success of removing a high proportion of these, the remaining few destroyed the result of the "least squares" technique for finding the vanishing point. It was necessary to find the intersection of each pair of line segments, and to distribute these intersection

points amongst a collection of "TanBins" – another data structure and algorithm that had to be developed from scratch.

7.4 Future Work

This project has the potential to be extended in several different directions. Some of these are discussed below.

At present all of the information extracted from the video sequence is utilised only to locate the vanishing point. Using the Sun's altitude in the sky it is possible to calculate the length of the shadows people cast. (An average height for the people observed in the video sequence would be required.) This should provide enough information about scale in the scene to eliminate the need for the user to specify the downward tilt of the camera. Furthermore, by associating an averaging walking speed with the people in the video clip it may be possible to determine the parameters that up to now have been assumed, i.e. the camera's aspect ratio and the location of its principal point. This procedure could be regarded as a pseudo-calibration of the camera.

They fact that there is a finite number of TanBins means that, at present, the vanishing point can only be located up to a certain level of accuracy. In order to improve upon this, a procedure that refines the position of the vanishing point within its TanBin could be developed. One possibility is to associate weightings of importance with each of the line segments, depending on their length and distance from the TanBin. Once an initial approximation to the vanishing point has been obtained, the least accurate line segments (those with the lowest weightings) would be discarded. Repeating the procedure would, in theory, produce a better approximation.

An alternative approach to locating the vanishing point involves first identifying the horizon. This could be done by using the observation that, as people move away from the camera, they converge to the horizon. It would be necessary to track people across frames in order to achieve this – something which could be accomplished by simple changes to the reliable line tracker developed for this project. The vanishing point would then be found very accurately as the intersection of *any* reliable line segment and the horizon.

This project has only used information from shadows cast by people. By using time-lapse photography shadows cast by large immovable objects, e.g. buildings, trees, lampposts, could easily be identified. It may be possible to use the amount by which these shadows move to determine the camera's direction.

For completeness it should be pointed out that the application developed for this project will only operate successfully on very particular video sequences. The scene must be of a large open area across which many people walk. Furthermore, people's shadows must be clearly visible (e.g. not hidden behind the person themselves), and far enough apart from one another that their common vanishing point can be found. I feel that the approach involving the tracking of people as they move towards the horizon (described above) has the potential to remove many of these constraints, providing the opportunity to develop a much more general-purpose application.

8 Conclusion

Automatic surveillance applications often need to relate information obtained from multiple cameras. The task can be made significantly easier if the position of the cameras relative to one another is known. This project successfully determines one piece of information required to solve this problem, namely, the compass direction of the camera.

A large amount of information about a scene can be extracted from the shadows it contains. In particular, since it is possible to know the precise position of the Sun in the sky for any given date time and location on the planet, shadows can be used to infer the compass direction in which the camera is pointing.

Finding the Sun's position is a purely mechanical problem. In this project first-order approximations were used, which provide more than sufficient accuracy. The positional astronomy required to solve the problem has existed for hundreds of years. By implementing the established formulae in a programming language it was possible to compute the answer without a thorough understanding of the Earth's orbit around the Sun. Nevertheless, a basic knowledge of the area was acquired in order to ensure the correctness of the results obtained.

Identifying shadows in a video sequence is generally a trivial task for humans. However, in order for a computer to perform this task the human perception of colour must be modelled. This project used a previously implemented algorithm that detects shadows by their difference from the background of the scene. Areas where both the brightness and saturation of colour are lower than they were in the background are identified as shadows.

Associating a direction with a region identified as a shadow is, once again, not a simple matter for a computer. The approach taken in this project was mathematical – each region's centre of gravity was located, and the line best representing its direction was determined. Unfortunately, the technique also associated directions with areas incorrectly identified as shadow – directions that did not point towards the Sun. To

overcome this problem it was necessary to track regions across frames. Only those that were present for a number of frames were used in subsequent calculations.

Inferring the camera's direction from those of the shadows required a mathematical model for the camera. It transpired that the solution involved finding the "vanishing point" of the shadows, i.e. the point at which all of the shadows appeared to converge. A procedure was developed to allow each pair of shadow regions to "vote" for the location of the vanishing point. The most popular location was used in the calculation to determine the camera's compass direction.

The results show that the algorithm developed for the project is capable of finding the direction of the camera using shadows, although some parameters must be provided by the user. The technique developed for tracking regions of shadow across frames could be altered to track people instead. Doing so would provide the opportunity to extend the application, both in terms of increasing its accuracy and making its operation fully automatic.

9 References

[ANTON 94] H. Anton. “Elementary Linear Algebra”. John Wiley & Sons Inc, Seventh Edition, 1994.

[ASTR 98] J. Mitton. “Dictionary of Astronomy”. Penguin Books, 1998.

[BUR 01] Keith Burnett. “Converting RA and DEC to ALT and AZ”. March 2001. http://www.xylem.f2s.com/kepler/altaz.html

[CARD 01] D. Cardani, Buena Software Inc. “Adventures in HSV Space”. April 2001. http://www.beuna.com/articles/hsvspace.pdf

[DARK 03] U.S. Naval Observatory, Astronomical Applications Department. “The Dark Days of Winter”. January 2003. http://aa.usno.navy.mil/faq/docs/dark_days.html

[EQUA 02] U.S. Naval Observatory, Astronomical Applications Department. “Approximate Solar Coordinates”. March 2002. http://aa.usno.navy.mil/faq/docs/SunApprox.html

[FINL 02] G. Finlayson, S. Hordley, M. Drew. “Removing Shadows from Images”. *Proceedings of the 7th European Conference on Computer Vision*, May/June 2002, Vol. 4, pp.823 ff.

[FOLEY 90] Foley, van Dam, Fiener, Hughes. “Computer Graphics: Principles and Practice”. Addison Wesley, 1990.

[FUNG 01] G. Fung, N. Yung, G. Pang, A. Lai. “Effective Moving Cast Shadow Detection for Monocular Color Image Sequences”. *Proceedings of the 11th International Conference on Image Analysis and Processing*, September 2001, pp.404-409.

[GIES 01] J Giesen. “Astronomical Algorithms”. January 2001. http://www.jgiesen.de/SME/details/basics/meeus.htm

[GIES 03] J Giesen. “Basics of Positional Astronomy and Ephemerides”. March 2003. http://www.jgiesen.de/SME/details/basics/

[HORP 99] T. Horprasert, D. Harwood, L. Davis. “A Statistical Approach for Real-time Robust Background Subtraction and Shadow Detection”. *Proceedings of IEEE ICCV’99 FRAME-RATE Workshop*, September 1999.

[JULD 01] U.S. Naval Observatory, Astronomical Applications Department. “Julian Date Converter”. October 2001. http://aa.usno.navy.mil/data/docs/JulianDate.html

[NOAA 03] United States National Oceanic and Atmospheric Administration, Surface Radiation Research Branch. “Solar Position Calculator”. March 2003. http://www.srrb.noaa.gov/highlights/sunrise/azel.html

[OFEK 02] Eran O. Ofek. “Julian Day”. August 2002. http://wise-obs.tau.ac.il/~eran/Wise/Util/Julian_Day.html

[PRAT 01] A. Prati, R. Cucchiara, I. Mikic, M. Trivedi. “Analysis and Detection of Shadows in Video Streams: A Comparative Evaluation”. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, December 2001, Vol. 2, pp.571-576.

[RIT 98] Judy Egelston-Dodd, Simon Ting. “Celestial Sphere Vocabulary”. April 1998. http://www.rit.edu/~sktnmp/judy/course/wk05/hndoutMedia/terminologies.htm

[SCHM 02] Unknown author. “Sidereal Clock”. November 2002. http://home.att.net/~srschmitt/clock.html

[SONKA 99] M. Sonka, V. Hlavac, R. Boyle. “Image Processing, Analysis, and Machine Vision”. PWS Publishing, Second Edition, 1999.

Appendix A: Julian Dates

This is the algorithm used to convert calendar dates (e.g. 28 March 2003 14:52:00) to Julian dates (2452727.11944 for the example). These are simply a continuous count of days and fractions of days since noon Universal Time on 1 January, 4713 B.C. (on the Julian calendar). They were invented by Josefh Justus Scaliger, who named them for his father Julius.

The algorithm can only operate on Gregorian dates, i.e. dates in the Gregorian calendar, which began on 15 October 1582 in Catholic countries. It is derived from the utilities of the U.S. Naval Observatory [JULD 01] and Eran Ofek [OFEK 02].

```
//Note: universal_time must be in decimal
//E.g. The time 18:36:00 must be passed as 18.6
double CalculateJulianDate( int day, int month, int year, double universal_time ){
    if (month <= 2) {
            year = year –1;
            month = month + 12;
    }

    double A = floor( (double)year / 100);
    double B = 2 – A + floor( A / 4 );

    double day_frac = universal_time / 24.0;

    double julian_date = floor( 365.25 * (year + 4716) )
            + floor( 30.6001 * (month + 1) )
            + day + B – 1524.5 + day_frac;

    return julian_date;
}
```

Appendix B: Least Squares Method

The least squares method can be used to solve an over-determined system of linear equations [ANTON 94]. In the case of equations representing 2D lines, the solution can be interpreted as the "best" approximation to their common intersection point. The technique is described below.

Multiple linear equations of the form

$$a\,x + b\,y = c \tag{B.1}$$

can be represented in matrix form as

$$A\,X = B \tag{B.2}$$

where, for a particular equation, each row of A holds the coefficients of x and y (i.e. a and b), and each row of B holds c. The least squares solution is found by solving for X the matrix equation

$$A^T\,A\,X = A^T\,B \tag{B.3}$$