Line Detection and Tracking in Video Recordings of Rugby Games

A thesis presented in partial fulfilment of the requirements for the degree of

Master of Science

in

Computer Science

at Massey University, Palmerston North, New Zealand

Xiao HUANG (黄晓)

2004

Errata Sheet

P.7, third paragraph, change "no-glass-coloured" to "no-grass-coloured"

P.10, move last line (sub-heading) to P.11

P.14, change equation at the top of the page from

$$G_{Ori} = \arctan(G_y / G_x) - 3\pi/4$$

to

$$G_{Ori} = \arctan(G_y / G_x) + \pi/4$$

P.19, line11, change "b is the intercept" to "b is the y intercept"

P.53, line 3, change "In frame 109" to "In frame 139"

Abstract

Video analysis has long been used in sports analysis. More and more coaches and instructors choose to use computer-based video analysis systems in decision-making and player training, which makes computer-aided sports analysis a fast growing industry. AnalySports Ltd is a New Zealand based sports analysis company providing performance analysis for sports, especially for team games such as rugby. Currently AnalySports uses human coders to manually track the activities and position of the ball carrier.

This thesis is part of a player-tracking project. The overall aim of the wider project is to build a cost-effective system to semi-automatically track individual players' positions in video recordings of team sport games. To obtain the information on movement and tactics of the team as a whole, it is necessary to identify the position of each player on the field at every point of time during the game. To perform the tracking, wide-angle video recordings from rugby games are used as input data. As the camera is moving, it is necessary to find the mapping between the pixels in the image and positions in the rugby field for every video frame. To make the size of the task realistic for a one-year masters project, the work presented in this thesis focuses on finding formulae and parameters for this conversion. Analysis of the position data for player performance and game tactics is outside the scope of this thesis.

The conversion between image coordinates and field positions can be established by identifying the field characteristics. The positions of the players in the field then can be calculated using this conversion once they are identified on the video frame. Based on single video frames, algorithms have been developed to detect and identify field characteristics (lines) in the frames of the video recordings. Using the identified field characteristics as reference, a transform matrix was calculated to convert pixels in the image to positions in the rugby field. For a sequence of video frames, algorithms have been developed to track the identified reference lines in order to save time and human power. These tasks were complicated by the zooming, tilting, and panning movements of the camera and therefore a potential of loss of reference lines, the noise in the data caused by field properties such as advertisement, the varying light conditions, the movements of the sun, or the shadows of the stadium roof. An application was developed to perform the developed algorithms. The testing shows that for about seventy percent of the video clips investigated, lines can be recognised and tracked. That means the application can be used to find the conversion for the majority of the video clips. Based on the testing performed,

further development based on this project could be a refinement of the image recognition parameters, efficiency improvements and the development of the actual player recognition.

This project was supported in part by a grant from the (New Zealand) Foundation for Research, Science, and Technology (FRST) and the Technology for Industry Fellowships (TIF) programme (contract number: ANLY0201).

Acknowledgements

First of all, I would like to thank Dr Eva Heinrich, my project supervisor, for her support throughout the project. Without the advice, comments and helpful guidance from her, this thesis would not have been possible. I thank her especially for her great patience and her professional, unconditional, dedicated encouragement and support.

I would like to express my gratitude to Dr George Serrallach of AnalySports Ltd, for his valuable advice and guidance in this project. Thanks must go to AnalySports Ltd, for providing video files and a laptop for this research.

Many thanks to Foundation for Research, Science, and Technology (FRST) for granting the financial support for this project.

Acknowledgement is also due Dr Donald Bailey for his advice on image and video processing techniques.

Finally, I would like to thank my family for their infinite love and support throughout my life.

Table of Contents

List of Figures

List of Tables

1 Introduction

Computer systems are being used in modern coaching and training, especially in sport analysis, to improve the sport performance of the athletes. Video recordings of sports are often transferred to the computer systems to perform accurate measurements on the performance of the athletes, which are essential to sport analysis. In order to improve the accuracy of the measurement, image-processing techniques can be used in computer aided sport analysis systems.

Section 1.1 of this chapter will give a brief introduction to computer aided sport analysis systems. In Section 1.2, some basic image processing techniques will be briefly reviewed. Section 1.3 will discuss the goal of this project. In Section 1.4, the structure of this thesis will be introduced.

1.1 Computer Based Sport Analysis

Computer systems have become more and more popular in sport coaching and training. Video recordings of sports are important input for those computer systems. Two kinds of computer systems will be introduced below: computer systems developed for training and computer systems developed for performance analysis.

In computer systems developed for athletes training, athletes' motions are captured and transferred to the computer. The video recordings in such a computer system are usually taken in a controlled environment, such as a controlled background or a controlled camera movement. Compared to video recordings played in VCRs, video recordings in computer systems are much easier to control; for example, the user can examine any frame of the video easily. The computer systems also provide accessorial tools, such as drawing tools and video editing tools to facilitate the analysis. Developed by Seaside Software, the SportsCAD GOLD system (SportsCAD GOLD, 2002) is an example of this kind of computer application. SportsCAD GOLD is an advanced video motion analysis program, which combines the power of video instruction and the graphic capabilities of the computer. It is used to support the training of athletes in sports like baseball, golf, and bowling. The drawing tools in SportsCAD GOLD allow users to draw shapes directly on the videos. Drawings can be saved as drawing macros and be called up and displayed over any video. The drawing tools also provide calibration for lines and can calculate angles. The path tracking function allows the user to mark interesting objects on the video

and track their trajectory using the mouse. The video editing tools in SportsCAD GOLD allow the user to display or align the videos in different modes. The user can play different videos on one screen, or show multiple frames of one video on the screen. Further more, two videos can be placed transparently for comparison. In such computer aided training systems, athletes' performances can be measured using accessorial tools provided by the computer application; for example, distance can be measured by measuring distance for lines in the image.

In computer systems developed for performance analysis, the video recordings used are usually taken in uncontrolled environments. Thus measurements (quantitative data) or observations (qualitative data) of the athletes' performances are usually entered to the system by the human investigator through observing the video recordings of sports. The computer applications then analyse the data prepared by human investigator and generates statistics and reports automatically. StadeXpert (StadeXpert, 2003) is an example of this kind of tool. Developed by REM Informatique, StadeXpert is a powerful game analysis system. For example, the StadeXpert application developed for rugby analysis allows the users to group the players' activities into different classes, such as 'tackle' and 'lineout'. Then it provides interfaces (Figure 1-1) to analyse the sequences of the rugby game to find out plays in the video clips, and qualify how successful the plays are. The computer system can generate reports and statistics of the rugby game.

There are difficulties in providing accurate measurements in video recordings taken in an uncontrolled environment. The biggest barrier comes from the fact that the camera's parameters (such as focal length, position and movement) are unknown, which makes traditional movement detection and analysis techniques not feasible. One possible solution is to use human investigators to take the measurement manually, which is time consuming and requires lots of human resources. The research project presented in this thesis focuses on providing accurate measurement data semi-automatically using image-processing techniques for video recordings taken in uncontrolled environments. Supported by Foundation for Research, Science and Technology (FRST) New Zealand, this research project was undertaken in collaboration with AnalySports Ltd. AnalySports Ltd is a New Zealand based sport analysis company providing performance analysis for sports, especially for team games such as rugby and soccer. Like other sports analysis companies, AnalySports currently uses human coders to manually track the activities and position of the ball carrier. The overall aim of this project is to develop a system that could track the players' positions in the field semi-automatically and thus provide accurate position information of the players in video recordings taken in an uncontrolled

environment. The final system would provide the coach with positions of players in the field at anytime, so that can be used in development and analysis of game tactics.

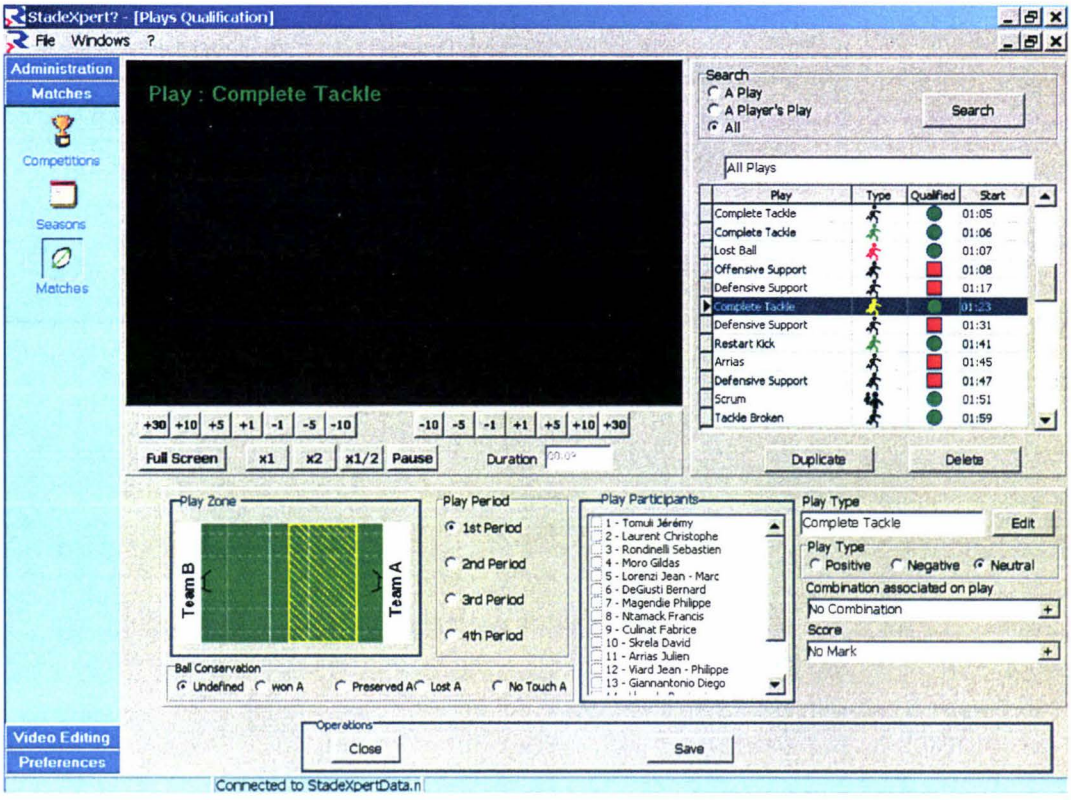


Figure 1-1 Video analysing interface of StadeXpert

1.2 Digital Image Processing Techniques

Digital image processing techniques have been used in a wide range of applications such as biological research, medical diagnostic imaging, video/film special effects, and remote sensing. Baxes (1994, p.1) gives a definition of image processing:

> Image Processing, in general terms, refers to the manipulation and analysis of pictorial information. In this case, pictorial information means a two-dimensional visual image. Any operation that acts to improve, correct, analyze, or in some way change an image is called image processing.

According to Baxes (1994), there are five classes of digital image processing operations – enhancement, restoration, analysis, compression, and synthesis. Of these techniques, digital image-analysis techniques are most commonly used in computer aided sport analysis systems. Image segmentation techniques are used to extract players from

backgrounds (fields) in video frames. Motion detection and tracking techniques are used to detect and track players' activities in video recordings. In Chapter 2 of this thesis, important fundamental image analysis techniques are reviewed.

1.3 Project Goal

The overall goal of the wider project was to develop a cost-effective system for the semi-automated tracking of individual players in video recordings of team sports like rugby and soccer. Such a system can be used to help the coach analyse the performance of players and develop game tactics.

The original plan for this one-year research was to use controlled test conditions, that is, one fixed camera to capture the whole rugby field and to initially place only one player on the field. With a static camera position, motion detection techniques could have been used to detect moving objects in the field, that is, the players and the referees. The conversion from pixels in the image to the position in the field would have required only one formula for the entire video recording. In sport fields, field characteristics like lines usually have known positions. For example, the rugby field is normally 100 meters by 70 meters, which means the distance between two goal lines is 100 meters and the distance between two touchlines is 70 meters. These characteristics make it possible to calculate the conversion formula by detecting and identifying lines. In order to find the formula for a video recording taken by a fixed camera, field characteristics (lines) would only have to be recognised once for the whole video recording, which could be done at the beginning of the video recording by a human operator. Thus a player's position as identified in the video recording could have been transformed to coordinates in the rugby field. A human operator was to manually mark the position of the player at the beginning of the video recording and the application would automatically track the player with image recognition techniques.

From the analysis of the geometric model, it was found that in order to capture the whole rugby field on one MPEG1-format video frame, the camera's position must be very high and far from the rugby field. A fixed camera position and angle is not feasible because the camera cannot be put that far back, further more, in such a situation, the number of pixels per player will become insufficient for recognition. One possible solution to this is to use multiple fixed cameras, which each capturing part of the rugby field. The retrieved video recordings can then be processed individually and the information from different video recordings can be combined to generate the final result. The problem with this approach

is that it requires extra costs for hardware and might consume more time in processing because of the increase in the number of video recordings. Another possible solution is to use only one camera to capture part of the rugby field. In this case, the camera has to swing or zoom to follow the players on the field, so that the camera captures the most important part of the rugby field where most activities of the player take place. As this approach reduces the cost in hardware, and simplifies the processing, it is selected as the direction of the research in this project. Thus, there is no point in making specific video recordings; video recordings from televised rugby games are suitable for the research. This change in the project from fixed to moving camera raised the complexity because with a moving camera, a conversion must be made for every video frame, that is, reference points need to be calculated for every frame.

With this change from a static to a moving camera, the change of a player's position between two sequential video frames can result both from the movement of the camera and the movement of the player. This means that player tracking has three components: detect a player in the video image; refer the location of the player in the video image to reference points on the rugby field to be able to find out the player's position on the rugby field; recognise the identity of the player by tracking. It is therefore essential to have reference points. Field characteristics, lines, have to be recognised for each video frame to achieve correct conversion between image coordinates and rugby field coordinates. Line tracking needs to be performed to avoid asking the human operator to identify lines for every frame.

With the change in requirements from fixed to moving camera, the focus of the project had changed to semi-automated tracking of field characteristics in video recordings of rugby games. A line tracking system needed to be conceptualised, built and tested.

1.4 Structure of the thesis

This thesis consists of six chapters. In Chapter 2, related research is discussed and relevant background knowledge is introduced. Chapter 3 of this thesis presents the conceptual design of the line tracking system. Chapter 4 discusses the implementation issues of the system. In Chapter 5, testing of the developed line tracking system is introduced and analysed. Finally, Chapter 6 concludes the thesis and discusses the potential for future work.

2 Literature Review

In this chapter, similar research in sports tracking systems using image-analysis techniques is reviewed first. Then some fundamental image analysis techniques are introduced.

2.1 Similar systems/Approaches/Tasks

In this section, three similar tracking systems are introduced. The scenarios in which these systems are applied are compared to the scenario of the system presented in this thesis.

- Perš and Kovačič (2001) presented a multi-camera people tracker. The goal of the project is to use a partially controlled environment to examine the accuracy of a proposed player tracking method. Video recordings of hand ball games have been used to test the algorithms.

  Two cameras are placed on the ceiling of the sports hall. The cameras are equipped with wide-angle lenses so each camera can capture half the court. The images from two cameras are combined to form the image of whole court. In this way the cameras can remain static to avoid continuous camera calibration.

  Tracking of players is performed on uncorrected images, and players' positions are transformed to real court positions after tracking. A combination of color-based tracking and template tracking techniques is used to track multiple players. A human operator supervises the tracking process, stops the tracking if necessary, manually marks the player, and then restarts the tracking.

  Transformation between image coordinate and court position is done by geometric calculation. This is possible as the position and parameters of the cameras are known.

  After tracking, the players' positions, velocities, and trajectories are stored to assess the accuracy of the tracking. The "ground truth", the known value, for the assessment is obtained simply by drawing a pattern of lines on the court as reference.

Perš and Kovačič's work shows that although it is possible to measure the accuracy of tracking of players' movement, the method provided is only suitable for limited applications. Further more, only overall accuracy of the tracking can be tested, it is still difficult to analyse the error propagation because there are several sources of errors that can influence the overall accuracy.

- Ekin and Tekalp (2002) presented a complete framework for automatic analysis of soccer video by using domain specific information. A MPEG-2 format soccer game video between two national teams has been used to test the algorithm.

  Shot boundary detection is first performed by finding the grass-colored area. Based on the fact that the sport field occupies the dominant area of the captured video frame, shot boundary detection is done by choosing the dominant color from the histogram in the hue space. The region of interest (ROI) is defined by filtering the original image with the dominant color. Then, based on the ratio of grass-colored pixels versus no-glass-colored pixels in a frame, the shot is classified as belonging to one of three classes: long shots, in-field medium shots, and out-of-field/close-up shots.

  In section four of Ekin and Tekalp (2002), long shots are examined. Line detection and viewpoint registration are performed for long shot frames. Lines are extracted within ROI based on the fact that lines have high RGB values and high vertical and horizontal gradients. An affine transformation model is used in viewpoint registration.

  A player's position is entered in the first frame manually in the form of a bounding box. The tracking of players in long shot frames is done by tracking feature points and the bounding box of the player. The motion of feature points is tracked by the Kanade-Lucas-Tomasi (KLT) feature tracker (developed by Tomasi et. al. between 1981-1996).

  The test result of the algorithm shows that the tracker is robust to instant direction changes of the players. By integrating the motion history, occlusion by the players in the same team can be overcome. The KLT feature tracker is also able to detect and track the player low-resolution video frames.

- Holdaway (2002) described an approach for identifying the behaviour of items in a scene using tracking. The goal of the application is to detect and track vehicles moving in a petrol station forecourt. A sequence of grey-level images captured by a static camera has been used to test the algorithm. The system developed includes detection and tracking units.

  Detection of objects is performed by a hybrid algorithm of background subtraction and double differencing. Position, area, and shape of detected object are stored for tracking.

  The tracking unit associates regions of interest identified by detection with objects of interest identified in previous frames. When a match is established, the object's information is updated.

  The test result shows that the detection and tracking units are able to detect items correctly and track them in many scenarios. The scenarios where it failed often involved obstructions. Solutions to this problem have been discussed in the report, which suggested that additional rules should be used to incorporate the forecourt map into the tracking.

Summary of Related Researches

In both Perš and Kovačič (2001) and Holdaway (2002), fixed cameras are used, which means the background in the video recording is static. Therefore their approaches to detect moving objects cannot be used in this project because in this project, installing dedicated fixed cameras around the rugby field is not appropriate.

The project presented in Ekin and Tekalp (2002) analyses video recordings of soccer games. Camera shots are classified into three classes. The long-shot video frames are quite similar to the inputs used in the project presented in this thesis. However, there are still significant differences between the video frames of soccer field and the video frames of rugby field used in this project. For example, the soccer field are usually of one 'pure' grass-color, while in rugby field, big advertisements are drawn in the grass-color field. For this reason, the techniques used in Ekin and Tekalp (2002) to extract the field areas may not generate a good outcome.

2.2 Basic Techniques

In this section, some image analysis techniques are introduced. In Section 2.2.1, motion detection and tracking techniques are summarised. In Section 2.2.2, image segmentation techniques and edge linking techniques are introduced to present an overview of line detection methods. In Section 2.2.3, perspective and affine transformations will be discussed.

2.2.1 Motion Detection and Tracking

Motion detection and tracking techniques are commonly used in computer systems for surveillance and traffic control. The aim of motion detection and tracking is to extract moving objects from a video recording, thus obtain or predict the trajectory of the object. There are two major types of motion detection: static background motion detection and dynamic background motion detection (Frédéric, 2003).

- Static Background Motion Detection

The simplest motion detection technique with a known static background is to compare the current video frame with the background image pixel by pixel (background subtraction). If a pixel is very different from the background, it is regarded as belonging to the object.

Frame differentiating (double differencing) can be used in cases when the background is unknown. Instead of comparing with the background image, a frame in the video sequence in compared with its preceding and succeeding frames. Proper thresholds are selected to determine the different areas. For example, I_{n-1}, I_n, and I_{n+1} are three frames in the image sequence, two binary images S_1 and S_2 can be generated by frame differentiating and the moving object O can be detected by logically "ANDing" these two binary images.

$$S_1 = I_n - I_{n-1}$$
$$S_2 = I_{n+1} - I_n$$
$$O = S_1 \cap S_2$$

Examples of background subtraction and double differencing can be found in both Frédéric (2003) and Holdaway (2002).

➢ Dynamic Background Motion Detection

Dynamic background motion detection is more complex than static background motion detection. As the background moves at the same time as the object, it's very difficult to separate the object from the background. In this case, one possible solution is to make a hypothesis that the object moves in different speed or direction from the background. Another possible solution is to use domain knowledge in the video analysis, that is, if there are some known features of the object that can be used to separate it from the background, image analysis techniques can be used to find out its position in every video frame.

2.2.2 Line Detection

In this section, two fundamental techniques are reviewed, first the image segmentation technique is summarized, and then technique for edge linking is introduced.

2.2.2.1 Image Segmentation

The goal of image segmentation is to divide the original image into regions, which are made up of pixels that have something in common (intensity, color, etc.). Pixels in one region might have similar intensity or color, which can be used to distinguish them from other pixels in the image, which may indicate that they belong to the same object. Image segmentation includes two classes (Luo, 1998): region-based image segmentation and edge-based segmentation.

➢ Region-based Image Segmentation

- Thresholding

  Thresholding is the simplest and most commonly used image segmentation technique. The values of pixels in the image are compared to a threshold value T. Pixels with value larger than T are considered belonging to the object. The output of thresholding is a binary image of the same size as original image, with pixels belonging to the object set to 1 (foreground) and others set to 0 (background). The threshold value T can be chosen by human operator or estimated automatically from the image or the image's histogram. This technique is often used in grey-level images.

- Multi-level Thresholding

Multi-level thresholding segments regions into intensity bands. But in many cases, the thresholds used to segment the regions are very difficult to estimate. In multi-dimensional images such as color images, the thresholding technique used in grey scale images is generalized to multi-dimensional thresholding. For example, histograms are calculated and thresholds are selected for each color space (e.g., R, G, and B), the final result is calculated by combining the results from each color space. However, as suggested by Russ (1995), Haralick and Shapiro (1985), and Uchiyama and Arbib (1994), there are limitations when using multiple histogram-based thresholding schemes because color distributions are multi-dimensional. For example, with 3-D data, the selected area in the color space will always be cuboid as shown in Figure 2-1, two colors ranges with overlap over some color axis, could not be separated using multi-dimensional thresholding. Ohta, Kanade, and Sakai (1980) suggested that instead of computing histogram individually in red, green, and blue (R, G, and B) color variables, a set of color features ((R+G+B)/3, (R-B), (2G-R-B)/2) could be used, which is more effective for color image segmentation.

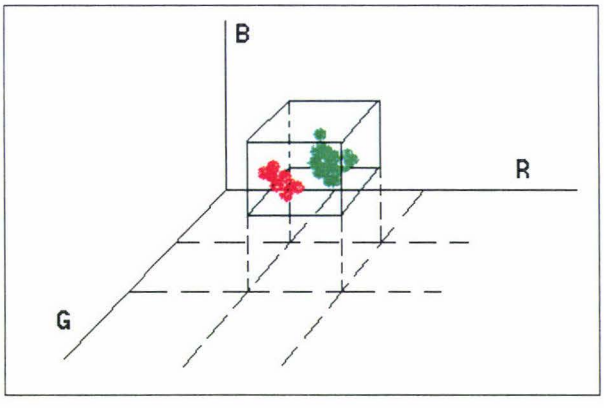


Figure 2-1 Multi-dimensional Thresholding in RGB Color Space

- Recursive Thresholding

Thresholding is implemented recursively in one region to get sub-regions. If the sub-region needs re-segment action, a threshold value will be calculated for the sub-region and the sub-region will be segmented using this threshold. The process ends if no sub-region needs re-segment action. Ohlamder, Price, and Reddy (1978) presented a recursive region splitting algorithm based on thresholding. For each sub-region, histograms of different color components are analysed to find the best peak. The best peak means well-separated peak, which is

a strong peak with very definite minima on both sides. If no best peak can be found, the segmentation is finished.

- Region Growing and Region Splitting

Region growing starts with a small region, called seeds, forming the object by merging neighbouring regions if the neighbouring regions having similar properties. Region splitting is a reverse process of region growing.

➢ Edge-based (boundary) Image Segmentation

Usually the edges of objects in the image have discontinuous intensity values (high gradients), making it possible to apply edge detection techniques to get edges of objects, and then use the edges to reconstruct the regions of objects (Luo, 1998). Some of the most popular edge detection techniques are:

- Gradient Operator

The gradient operator calculates the first order derivate at each position of the image.

$$f(x,y) = \nabla = \partial g(x,y)/\partial x + \partial g(x,y)/\partial y$$

Where the gradient vector *f(x, y)* represents the changing value of the function *g(x, y)* along the x and y directions. As an image is discrete, it cannot be differentiated in the same way as a function, therefore it is necessary to approximate the derivative at a pixel *(x, y)* as the difference in intensity over its adjacent pixels, that is,

$$\text{horizontal } \nabla_x = \partial g(x,y)/\partial x = g(x,y) - g(x-1,y)$$
$$\text{vertical } \nabla_y = \partial g(x,y)/\partial y = g(x,y) - g(x,y-1)$$

The length of the hypotenuse of the right triangle having ∇x and ∇y as sides is the magnitude of the gradient vector; it reflects the strength of the edge at pixel *(x, y)*. The angle the hypotenuse makes with the axis reflects the orientation of the edge. They can be represented as:

magnitude $G_{Mag} = \sqrt{\nabla_x^2 + \nabla_y^2}$

orientation $G_{Ori} = \arctan\left(\frac{\nabla_y}{\nabla_x}\right)$

The magnitude is often approximated by adding the absolute value of ∇x and ∇y for computational simplicity. The gradient operator can be implemented by convolving the input image with below masks:

horizontal $\begin{bmatrix} 1 & -1 \end{bmatrix}$

vertical $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$

- Robert Operator:

The Robert operator, as well as Sobel and Prewitt operators, approximate the first order derivative. The Robert operator uses a 2x2 window:

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Assuming that the intensity value at point *(x, y)* is *g(x, y)*, the Robert operator could be represented as:

$$G_x = g(x, y) - g(x+1, y+1)$$
$$G_y = g(x, y+1) - g(x+1, y)$$

The magnitude of the edge again could be one of the forms:

$$|G_{Mag}| = \sqrt{G_x^2 + G_y^2}$$
$$|G_{Mag}| = |G_x| + |G_y|$$
$$|G_{Mag}| = Max(|G_x|, |G_y|)$$

the orientation of the edge is:

$$G_{Ori} = \arctan(G_y / G_x) - 3\pi/4$$

- Prewitt Operator:

The convolution masks of the Prewitt operator is

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

vertical horizontal

These masks are designed to respond maximally to vertical and horizontal edges. The masks can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (called *Gx* and *Gy*).

$$\begin{aligned} G_x &= [g(x+1, y-1) + g(x+1, y) + g(x+1, y+1)] \\ &\quad -[g(x-1, y-1) + g(x-1, y) + g(x-1, y+1)] \\ G_y &= [g(x-1, y-1) + g(x, y-1) + g(x+1, y-1)] \\ &\quad -[g(x-1, y+1) + g(x, y+1) + g(x+1, y+1)] \end{aligned}$$

These can then be combined to find the absolute magnitude of the gradient at each point and the orientation of that gradient in a similar way as described in the Robert operator. The orientation of the edge can be calculated by:

$$G_{Ori} = \arctan(G_y / G_x)$$

- Sobel Operator:

The Sobel operator is similar to Prewitt operator, except that the masks give higher weight to central pixels:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

vertical horizontal

The gradient components are calculated by:

$$G_x = [g(x+1, y-1) + 2g(x+1, y) + g(x+1, y+1)] - [g(x-1, y-1) + 2g(x-1, y) + g(x-1, y+1)]$$

$$G_y = [g(x-1, y-1) + 2g(x, y-1) + g(x+1, y-1)] - [g(x-1, y+1) + 2g(x, y+1) + g(x+1, y+1)]$$

- Laplacian Operator

The edge-detection techniques introduced so far are first order derivates. One problem with the first order derivates is that they may produce an edge of several pixels wide for slowly ramp steps (smooth linear variations in intensity), it is difficult to locate the centre of the edge and edge thinning is required. A second order derivative will avoid this problem by differentiating the output of first order derivate. The Laplacian operator is one of the most popular second order derivatives.

$$f(x, y) = \nabla^2 = \partial^2 g(x, y)/\partial x^2 + \partial^2 g(x, y)/\partial y^2$$

As analysed above, the first derivative of the image function (such as the gradient operator) should have an extreme at the pixel corresponding to the edge in the image, and so the second derivative should be zero at the same pixel. After applying the Laplacian operator, the next step is to find zero-crossing in the output image. The method of determining zero crossings with some predefined threshold is to pass a 3x3 window across the image, and for each pixel, determine the maximum and minimum values within that window. If the difference between the maximum and minimum value exceeds the predetermined threshold, the pixel is an edge pixel. The Laplacian operator can be approximated by:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 2 & -1 & 2 \\ -1 & -4 & -1 \\ 2 & -1 & 2 \end{bmatrix}$$

The Laplacian operator is isotropic; no direction is given in the output.

- Laplacian of Guassian (LoG) operator

The LoG operator is also called Marr-Hildreth operator. The Laplacian operator is very sensitive to noise, so in LoG, the input image is first smoothed by convolving with a 2-D Guassian operator.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Where x and y are image coordinates, and σ is the standard deviation of the associated probability distribution. The value of σ is proportional to the size of neighbourhood on which the filter operates.

Because of linearity of the operations, the order of differentiation and convolution can be interchanged and combined to form a single operator. That is, the Laplacian of the Gaussian can be precomputed analytically to reduce the complexity of composite operation. The LoG operator can be represented as:

$$LoG(x, y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2 + y^2}{2\sigma^2}\right] e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

It is shown in Figure 2-2 (centred at point (0, 0), for a Gaussian σ of 1.6).

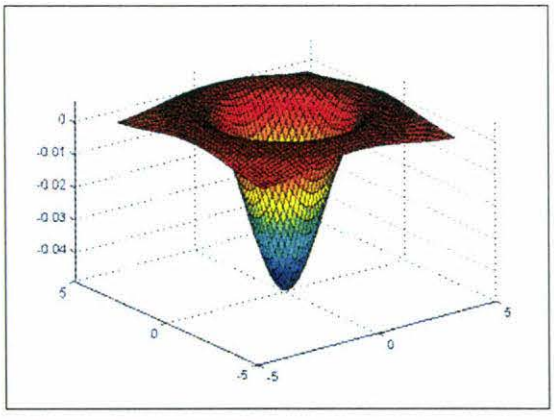


Figure 2-2 LoG Operator

Because of the shape of the LoG operator, it's commonly called Mexican hat.

The Difference of Gaussians (DoG) filter is a variant of LoG. It uses the difference of two differently sized Gaussians to approximate the LoG filter. One advantage of DoG is the ability to specify the width of the edges by varying the values of σ_1 and σ_2.

$$DoG(x,y) = \frac{e^{-(\frac{x^2+y^2}{2\pi\sigma_1^2})}}{2\pi\sigma_1^2} - \frac{e^{-(\frac{x^2+y^2}{2\pi\sigma_2^2})}}{2\pi\sigma_2^2}$$

The ratio $\sigma_1/\sigma_2 = 1.6$ results in a good approximation of the LoG (Mai, 2000).

o Canny Edge Detector

The Canny edge detector is another important edge detection method developed by Canny in 1986. Canny's approach is based on optimising a set of goals (Parker, 1997):

- Error rate --- The edge detector should respond only to edges, and should find all of them; no edges should be missed.
- Localization --- The distance between the edge pixels as found by the edge detector and the actual edge should be as small as possible.
- Response --- The edge detector should not identify multiple edge pixels where only a single edge exists.

The Canny edge detector uses the following algorithm (Parker, 1997):

- Read in the image to be processed, *I*.
- Create a 1D Gaussian mask *G* to convolve with *I*. The standard deviation s of this Gaussian is a parameter to the edge detector.
- Create a 1D mask for the first derivative of Gaussian in the *x* and *y* directions; call these *Gx* and *Gy*. The same s value is used as in step 2 above.
- Convolve the image *I* with *G* along the rows to get the *x* component image *Ix*, and down the columns to give the *y* component image *Iy*.
- Convolve *Ix* with *Gx* to give *Ix'*, the *x* component of *I* convolved with the derivative of the Gaussian, and convolve *Iy* with *Gy* to give *Iy'*.

- Combine x and y components to give the result. Calculate the magnitude and orientation of the edge at each edge point.
- Non-maximum suppression, remove pixels that are not local maxima.

Summary of Image Segmentation

Image segmentation can be region-based or edge-based.

For region-based image segmentation, thresholding and multi-dimensional thresholding are easy to implement and faster. The problem is how to find the thresholds automatically. The limitation of multi-dimensional thresholding is that it sometimes will fail to distinguish two colors because of the overlap between them.

For edge-based image segmentation, gradient operators and template operators based on first order derivatives are fast and simple, the output includes magnitude and orientation of the edge pixel. The structures of the convolution masks make those operators favour horizontal and vertical edges, although compass operators (such as the Kirsch operator, which uses eight masks to convolve the image) can be used to improve the situation, the directions they favour are still limited. First order derivatives may have problems with ramp edges. The Laplancian operator uses second order derivative to overcome this problem. But it is even more sensitive to noise. So the LoG and DoG are introduced as they smooth the image before derivations.

2.2.2.2 Edge Linking

Edge linking groups the connected pixels in the output of the edge detector to form boundaries of the object. In general, edge linking methods can be classified into two categories: local edge linking and global edge linking.

➢ Local Edge Linking

Edge points are grouped to form edges by considering each edge point's relationship to any neighbouring edge points. As discussed above, most edge detectors (such as Robert and Sobel operator) yield the information about the magnitude and orientation of the edge at each edge pixel. The basic technique for edge following can be described as:

- Start from one edge point, seek from its neighbours (a small area surrounding it, usually a 3x3 or 5x5 window) similar magnitude and orientation points based on predefined criterions. If found, add the point to current edge set.

- Iterate the process for new-found edge points until no similar edge can be found.

- Start a new edge set, choose another unmarked edge point, repeat until no edge is left.

➢ Global Edge Linking

Hough transform is the most popular global edge linking technique. The earliest Hough transform was proposed by Hough in 1962 for straight-line detection. Subsequently, this method was developed to detect circles and arbitrary curves (Luo, 1998).
In Cartesian coordinates, the equation of a straight line is

$$y = m \cdot x + b$$

Where m is the slope and b is the intercept. In the Hough transform, the polar coordinate representation of a line is used; each (ρ , φ) pair defines a line.

$$\rho = x \cdot Cos(\varphi) + y \cdot Sin(\varphi)$$

The mapping between two coordinate systems is shown in Figure 2-3. A point (x_0, y_0) in Cartesian coordinate system forms an equation in polar coordinate system.

$$\rho = x_0 \cdot Cos(\varphi) + y_0 \cdot Sin(\varphi)$$

This equation represents a sinusoid curve in the polar coordinate system. In Cartesian coordinates, two points define a line; similarly, in polar coordinates, the intersection of two sinusoid curves represents a line.

Hough space is an accumulator space, which means it sums up the votes of many pixels in the image. Every point in real-space image casts its votes into ρ-φ points in Hough space along the sinusoid. Points in Hough space that have large total vote (intersections of sinusoids) are been interpreted as lines in the real-space image (Russ, 1995).

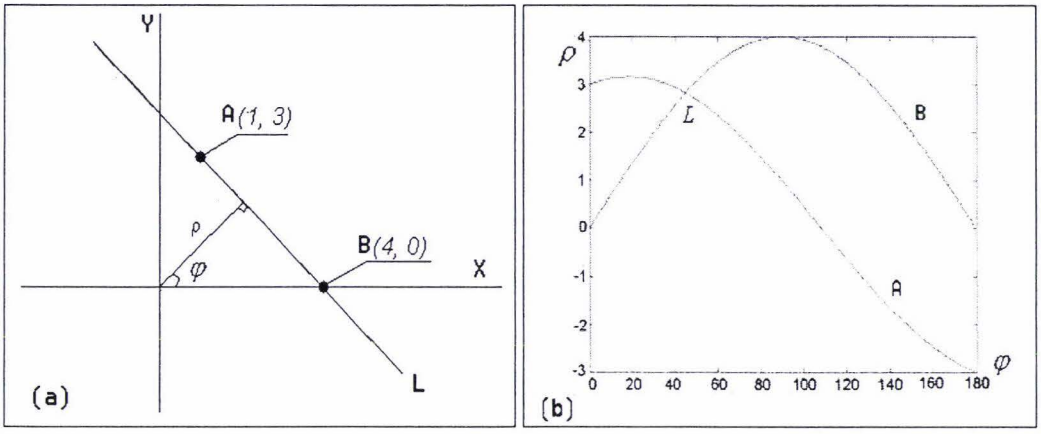


Figure 2-3 Hough Transform.

(a). Line L in Cartesian coordinate, A and B are two points in it. (b). Sinusoid curves of points A and B; the intersection represents line L in polar coordinate.

2.2.3 Geometric Transformation

Two kinds of geometric transformation are introduced in this thesis. The *perspective transformation* establishes a mapping between the coordinate system in an image and the coordinate system of a flat surface in the real world, which is captured by the camera. An *affine transformation* is an important class of linear 2-D geometric transformations, which maps two coordinate systems in two images by applying a linear combination of translation, rotation, scaling, and shearing operations. Many sources describe these transformations, such as Loomis (1997) and Roth (2002).

2.2.3.1 Perspective Transform

A camera uses a lens to focus part of the visual environment onto a sensor, thus forms a mapping between 3D world and 2D image. This model of projection is commonly referred to as the perspective camera model or pinhole camera model, as shown in Figure 2-4. A point *P1* in a flat surface of the real world is projected to point *P1'* in the camera's sensor plane. In projective geometry, parallel lines are not preserved, but straight lines remain straight (collineation), that is, points in a straight line in real world will be projected to form a straight line in the image.

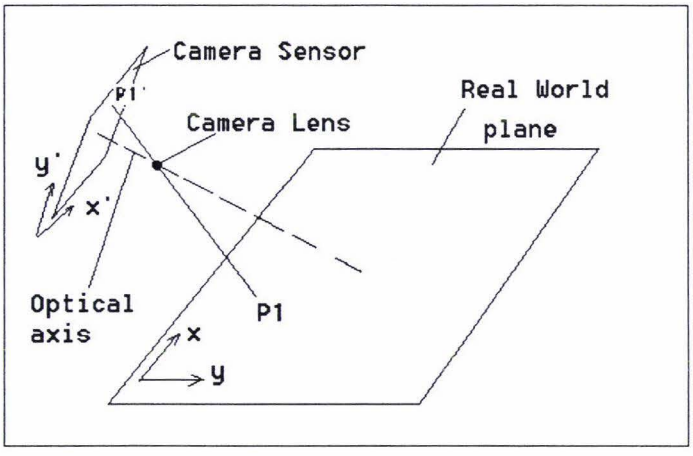


Figure 2-4 Pinhole Camera Perspective Projection Model.

A projective transformation is an invertible linear mapping h from P^2 (projective plane) to itself such that collinearity is preserved. A mapping h: $P^2 \rightarrow P^2$ is a projectivity if and only if there exist a non-singular 3x3 matrix H such that for any point in P^2 represented by a vector x, it is true that $h(x) = Hx$, that is,

$$\begin{pmatrix} x_1' \\ x_2' \\ x_3' \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Here the points' coordinates are represented in homogenous coordinate system; for example a 2-D point (x, y) is represented as a 3-D vector $(x1, x2, x3)$ where $x=x1/x3$ and $y=x2/x3$. As a result, inversions or combinations of linear transformations (such as translation, rotation, and scaling) could be simplified to inversion or multiplication of the corresponding matrices. To convert a point (x, y) in Cartesian coordinates to homogenous coordinates, simply set $x3$ to 1, so that the point in homogenous coordinates is $(x, y, 1)$.

The matrix H has eight degrees of freedom (DOF), which means there are eight variants in the matrix. Bailey (2002, Section 2.5) presented an approach to find the coefficients in the transform matrix H, in case that the mapping is from a planar surface with known grid lines (horizontal and vertical lines) to the image. The general steps are described as below:

- Represent a point (on the image) in homogenous coordinate system as:

$$P = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- A 2D perspective transformation can be represented as

$$kP' = \begin{bmatrix} kx' \\ ky' \\ k \end{bmatrix} = HP$$

- A line L can be represented in homogenous coordinate system as:

$$ax + by + c = 0 \Rightarrow L = [a\ b\ c]$$

so a point will be on a line if:

$$LP = 0$$

- A line on the planar surface can be represented as:

$$L' = LH^{-1}$$

- Assume that for one vertical line [a b c] on the image, the representation of this line on the planar surface is [1 0 Pv], so:

$$[1\ 0\ P_V] = [a\ b\ c]H^{-1}$$

For horizontal line, there is

$$[0\ 1\ P_H] = [a\ b\ c]H^{-1}$$

- Assume that

$$H^{-1} = \begin{bmatrix} h_1 & h_4 & h_7 \\ h_2 & h_5 & h_8 \\ h_3 & h_6 & h_9 \end{bmatrix}$$

- For each vertical line, there are two equations:

$$\begin{bmatrix} 0\,0\,0\,a\,b\,c\,0\,0\,0 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = 0 \text{ and}$$

$$\begin{bmatrix} -aP_V & -bP_V & -cP_V & 0\,0\,0\,a\,b\,c \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = 0$$

Similarly, for each horizontal line, the equations are:

$$\begin{bmatrix} a\,b\,c\,0\,0\,0\,0\,0\,0 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = 0 \text{ and}$$

$$\begin{bmatrix} 0 & 0 & 0 & -aP_H & -bP_H & -cP_H & a & b & c \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = 0$$

- Having four lines (two vertical, two horizontal), there is a set of equations:

$$\begin{bmatrix} 0 & 0 & 0 & a & b & c & 0 & 0 & 0 \\ -aP_V & -bP_V & -cP_V & 0 & 0 & 0 & a & b & c \\ \cdots\cdots & & & & & & & & \\ a & b & c & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -aP_H & -bP_H & -cP_H & a & b & c \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

- By solving this, the transformation matrix can be formed. With this transform matrix, it is possible to calculate the transformed coordinates in the image if the coordinates in the real world plane are known before transformation.

2.2.3.2 Affine Transform

In affine transformation, collinearity and parallelism are preserved, i.e., points that lie on the same line will remain on the same line after transformation. And two lines will be parallel after transformation if and only if they are parallel before transformation. Affine transform could be treated as composition of translation, rotation, scaling, shear (Loomis, 1997), as illustrated in Figure 2-5.

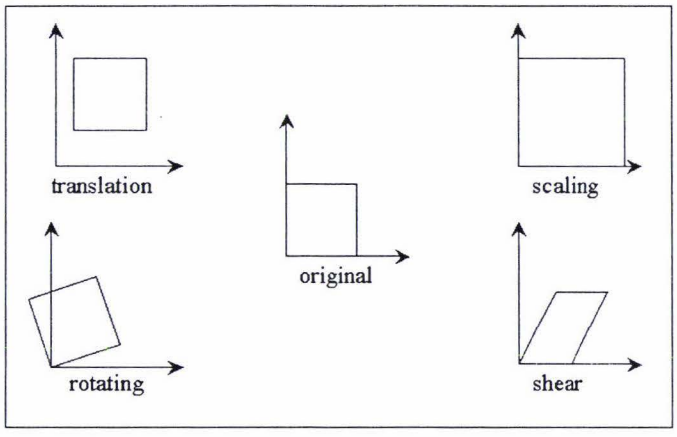


Figure 2-5 Affine Transform Composition

The transform matrices for translation, rotation, scaling, shear are:

translation $\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$

rotation $\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$

scaling $\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$

shear $\begin{pmatrix} 1 & s_y & 0 \\ s_x & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

The transform matrix of affine transform could be represented as:

$$\begin{pmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} A & t \\ 0^T & 1 \end{pmatrix}$$

Where A is the composition of rotations, scale and shear (non-isotropic scaling), and t is translation.

The matrix for affine transformation has six degrees of freedom. The coefficients in the matrix can be calculated in similar way as for perspective transformation, but require less number of lines (only three lines are required as there are only six variants). The motion of a planar surface can often be modelled to good accuracy using an affine transformation.

2.3 Summary

In this chapter, related research on sports image analysis systems has been introduced and relevant image-processing techniques have been reviewed.

In both Perš and Kovačič (2001) and Holdaway (2002), video recordings from controlled environments were used. This makes it possible to distinguish the motion object from the static background. But in this project, televised rugby games are used, making it more complex extracting the moving objects (players) from the background (rugby field). In Ekin and Tekalp (2002), a complete framework for processing video recordings of soccer games was presented. Camera shots were classifies to three classes in their paper, so their work deals with both long shots and in-field medium shots. In this project, only long-shot video frames are examined, as they are more useful in terms of tactic analysis. The video recordings have been pre-processed by AnalySports Ltd, so that most of the video frames are long shots.

With the dynamic background video recordings, motion detection and tracking techniques for static background cannot be used. Furthermore, no assumption can be made on the moving speed or orientation of camera. However, it is assumed that for most of the video clips the lines in the rugby field are clear enough to be detected using image segmentation techniques. The representation of the extracted lines in the image plane then can be calculated using Hough transform. The representation of the lines in real rugby field coordinates are known and can be used to find the matrix of the geometric transform, thus a mapping between the video frame coordinate and the position of the rugby field can be established. Chapter 3 gives a detailed description of the algorithms involved.

3 Conceptual Design and Requirement Analysis

In this chapter, the algorithms developed for line tracking are described, then the functional requirements are summarised. Section 3.1 gives a brief introduction to the task. In Section 3.2, the proposed algorithms are discussed in detail. Section 3.3 gives detailed specification of the requirements.

3.1 Overview of the Task

The overall aim of this project is to develop a cost-effective system for semi-automated tracking of players in video recordings of team sports like rugby and soccer. The final developed system should be able to provide an interface to a human operator to mark a player's position on the video recording, and then automatically track the player's position until the tracking is not possible any more.

Due to the size of the rugby field, it is not possible to capture the whole rugby field in a MPEG-1 format video frame unless the camera can be placed very high and far from the rugby field. Doing so requires very high costs, for example, the cost for hiring a balloon. One alternative way is to use multiple fixed cameras; each captures one part of the rugby field. The recorded videos can be aligned to form a full picture of the rugby field and then detection and tracking can be performed. Compared to one-camera approach, this approach increases the cost of buying extra equipments. It may also raises the processing cost and time consumption due to the alignment and combination of videos.

As discussed above, there are difficulties in obtaining and processing video recordings captured by one or multiple fixed cameras. Therefore the decision has been made to use video recordings captured by one moving camera. More specifically, the video recordings used in this project are taken from televised rugby games. Compared to other video recordings, televised video recordings are fully accessible and cost-effective. There is no extra capturing equipment required; and problems such as obtaining the recording rights are avoided. Furthermore, in televised video recordings, the camera is always moving (zooming, panning, tilting) to follow the hotspot of the play, which means the captured video includes the most important activities happening on the rugby field. One problem with televised video recording is that the quality of the recordings is not guaranteed, it may be affected by reasons such as transmission quality. Another problem with televised video recordings is that they may contain un-continuous play, such as stoppages after

infringements or playback of scenes. The video recordings may contain close-up shots and medium shots, which is not preferred because the focus of the project is the position of the players on the field, not their activities. For these reasons, AnalySports has pre-selected video recordings so that they contain continuous phases of play, and that most of the video frames are long shots (that is, the camera is capturing a large area of the field).

For recording a rugby game, the camera is moving during the recording to follow the play. That means the background of the video recording, the field, is not static. Therefore motion detection and tracking techniques used for static background are not appropriate for this project. Domain knowledge must be adopted to detect and identify the moving objects, the players, in the video recordings.

As introduced in Section 2.2.2.1, many image segmentation techniques exist to extract objects from an image. But even if the players' positions can be identified in the video frame, their positions in the rugby field are still unknown unless there are some techniques to register the video frame to a predefined model of the rugby field. As discussed in Section 2.2.3.1, from the analysis of perspective projection, it is found that lines in the field can be used as reference to calculate a transform matrix. This matrix can be used to convert pixels in the video frame to positions in the rugby field. In Section 2.2.2, line detection techniques have been discussed, which included two major parts: image segmentation and edge (line) linking. Applying the line detection techniques to a video frame can result in a list of detected lines. Then the human operator should be able to name the detected lines. Manually marking and naming lines in every video frame is not realistic and time consuming. So this project focuses on detecting and tracking of field characteristics (lines) in order to make it possible to convert a pixel in the video frame to a position (coordinate) of the field semi-automatically. After the transformation matrix been calculated for every video frame, detection and tracking of the players can be performed, which will be part of the future development of this project.

The proposed system contains three components, as can be seen in Figure 3-1. The preprocessing module prepares the input data for the application. The line detection and tracking module extracts the lines in every frame, identifies them and then calculate the transform matrix based on identified lines. The testing module provides an interface for the human operator to manually check the accuracy of the calculated transform matrix. The next section gives detailed descriptions of these components.

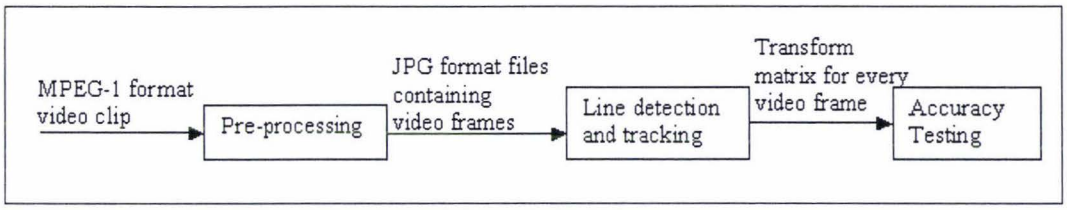


Figure 3-1 Block Diagram of the System

3.2 Proposed Solution

The design of the modules of the system is discussed in this section. In Section 3.2.1, the procedure of pre-processing is covered. Section 3.2.2 explains the algorithm for line detection and tracking, as well as how the transform matrix is calculated. In Section 3.2.3, the algorithm for testing the accuracy of the results from the previous module is discussed.

3.2.1 Pre-processing

As this project is undertaken in cooperation with the AnalySports Ltd, the company provided the video clips used in developing and testing the algorithm. The video clips are already selected so that the camera is making long shots in most of them. Furthermore, they have been cut to cover only one active play of the game (about 50 seconds in average) as data processing during stoppage times would not be required. Figure 3-2 shows a typical frame of a video clip.

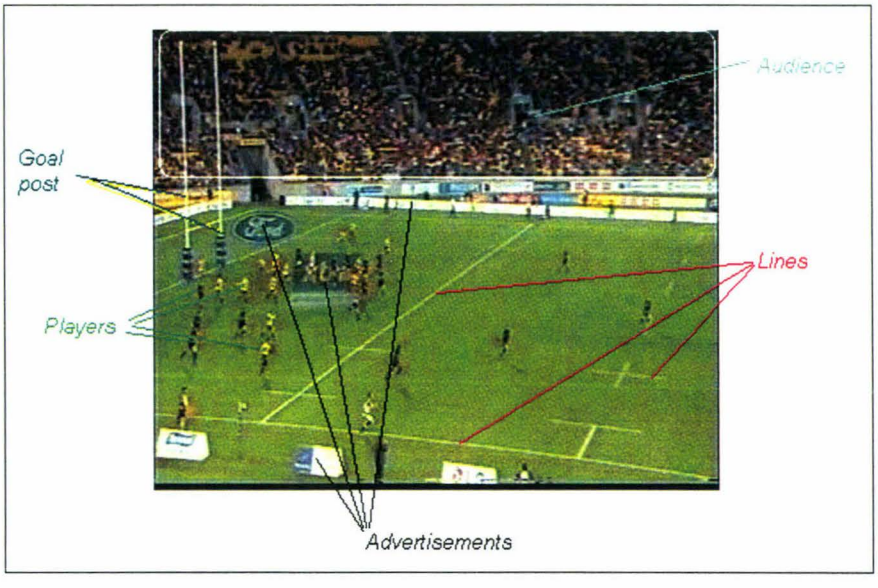


Figure 3-2 A Typical Video Frame

The primary purpose of the pre-processing module is to prepare the input for the line detection and tracking module. As most image processing techniques take an image as an input, video frames need to be extracted from video clip for this purpose. The pre-processing module simplifies this with extracting every frame from the video clip in advance and saving the frames as bitmaps with consecutive names. There are several benefits in doing so:

- Accessing one video frame out of a image sequence is much easier. Using a video clip to directly locate one certain frame accurately, special video editing tools are required. On the other hand, one video frame in the bitmap sequence can fast be located by it name and viewed by ordinary bitmap-viewing tools.

- Control of the starting and stopping point of the processing is much easier for image sequences. The collection of files containing the video frames can be easily split and grouped. This provides a flexible way of controlling which part of the video clip is to be processed.

- For this early stage of the system development, the pre-processing is sufficient, time and money is saved in finding and purchasing a module to read the video clip and extract one frame from it. However, once such a module was available, it would not be hard to integrate it into the final system in later project phases.

The outcome of the pre-processing is that the frames of a video clip are extracted to JPG files, and then saved to a dedicated folder with consecutive names. For example, with an input video file with the name "01.mpg" of 30 seconds length, a dedicated folder "p:\videoclips\01", for example, would be created. The individual video frames would be saved in JPG format as files with consecutive names from "000.jpg" to "749.jpg".

3.2.2 Line Detection and Tracking

In this section, line definition and coordinate systems are introduced. Then the choices of line detection and tracking techniques are explained. Finally the algorithm of calculating the transform matrix is covered.

3.2.2.1 Introduction to Rugby Fields and Coordinate Systems

Figure 3-3 shows a diagram of a rugby field. It explains the names of the lines in a rugby field, which are used in line detection and tracking. Although there are 15 lines shown in the diagram, only 13 of these are in fixed positions and thus have fixed representations in the coordinate system of the rugby field. The positions of the two dead-ball lines are not compulsory binding, so they cannot be used as reference lines.

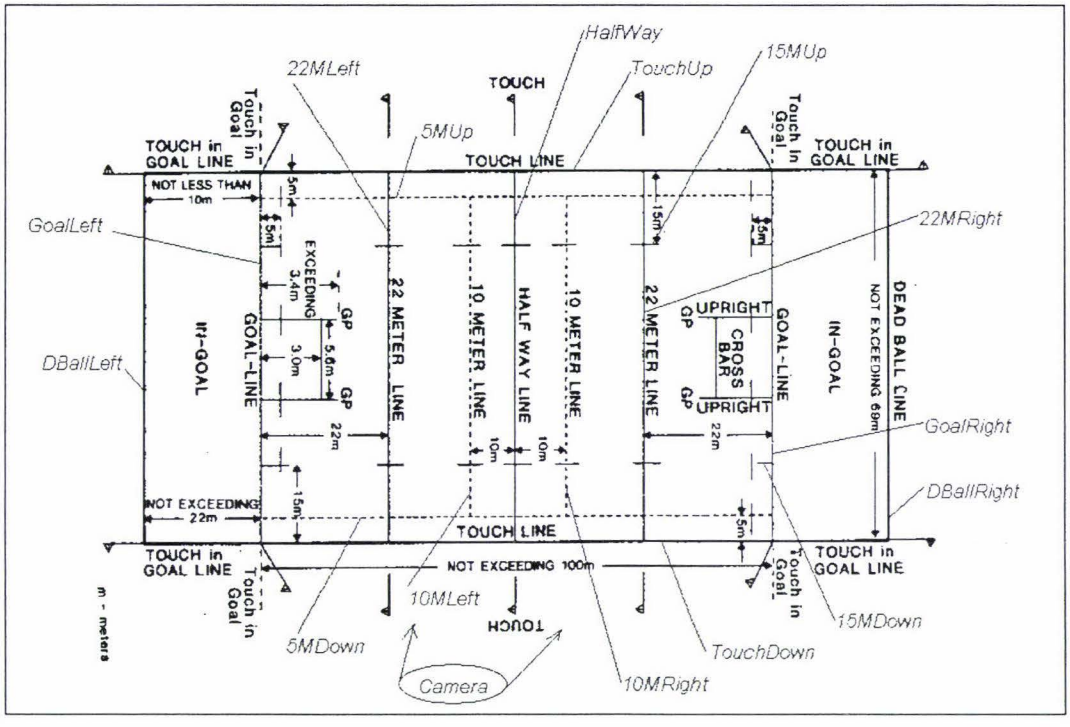


Figure 3-3 Rugby Field Dimensions and Line Definitions*

Before developing the system, it is essential to have precise definitions of the coordinate systems both in the video frame and in the rugby field. Figure 3-4 and Figure 3-5 show the definition of coordinate systems in the rugby field and in the video frame.

The coordinate system in the rugby field takes the centre of the rugby field as origin (coordinate (0, 0)), its x-axis is parallel to the touch line of the rugby field; its y-axis superposes the half way line of the rugby field. According to this definition, the representation of the left goal line can be written as $x+50=0$, which equals to $[1\ 0\ 50]$ in the homogenous coordinate system. The representation of the 13 lines in the rugby field can be given as shown in Table 3-1.

* Original image retrieved from http://www.hendersonrugby.com/rules/field.html

The coordinate system in the video frame takes the centre of the image as origin, with x- and y-axes parallel to the edge of the image respectively. As the video clips are MPEG-1 format, the extracted video frames have the resolution of 352x288. Because a pixel is treated as a discrete unit in the coordinate system (that is, a pixel location such as (-30.5, 0.5) is not meaningful), the origin of the coordinate system is set to the intersection of the 176th vertical line (counted from left to right) and the 144th horizontal line (counted from top to bottom). As a result, the top-left corner of the image has the coordinate (-175, 143) while the bottom-right corner has the coordinate (176, -144).

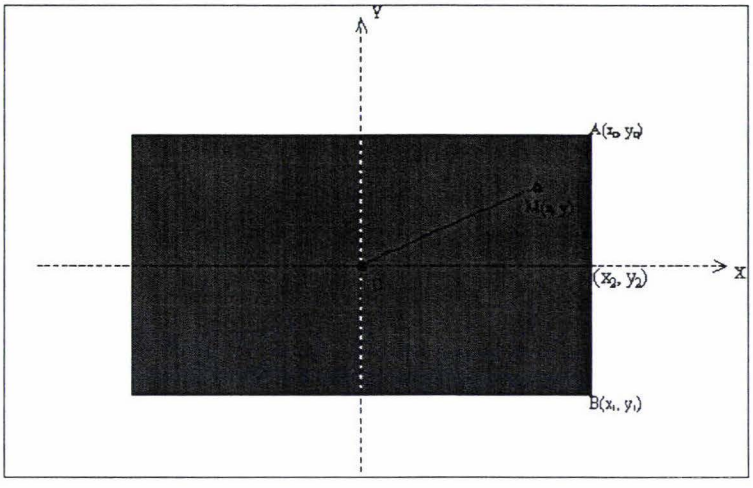


Figure 3-4 Coordinate System in Rugby Field

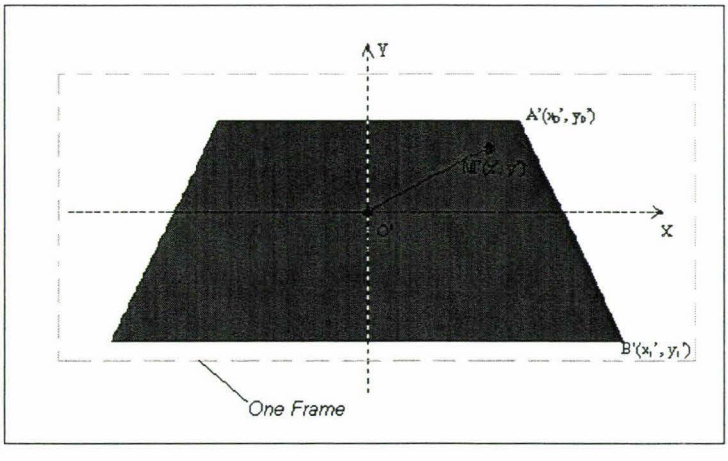


Figure 3-5 Coordinate System in Video Frame

Vertical Lines	No	X	Representation
GoalLeft	1	-50	[1 0 50]
GoalRight	13	50	[1 0 -50]
22MLeft	3	-28	[1 0 28]
22MRight	11	28	[1 0 -28]
10MLeft	5	-10	[1 0 10]
10MRight	9	10	[1 0 -10]
HalfWay	7	0	[1 0 0]
Horizontal Lines	**No**	**Y**	**Representation**
TouchUp	2	35	[0 1 -35]
TouchDown	12	-35	[0 1 35]
5MUp	4	30	[0 1 -30]
5MDown	10	-30	[0 1 30]
15MUp	6	20	[0 1 -20]
15MDown	8	-20	[0 1 20]

Table 3-1 Representation of Lines in Rugby Field

Having the coordinate systems defined, the transform matrix can be calculated once sufficient lines have been identified in both coordinate systems. This transform matrix then can be used to convert from the pixel locations in the coordinate system of video frame to the field position in the coordinate system of the rugby field and vice versa.

3.2.2.2 Line Detection

The purpose of line detection is to find out which pixel in the image belongs to the lines, and then group pixels belonging to the same line to find the representation of the line in the image coordinate system.

3.2.2.2.1 Extracting Lines

Each video frame is actually a 288*352*3 matrix, which means there are 288*352 pixels in one video frame, and each pixel has three color-components (RGB). The process of extracting lines contains two steps. Image segmentation technique is used to generate a list of pixels belonging to the lines. Then the edge linking technique is applied to get the representation of the lines in the video frame.

Line Detection ---- Image Segmentation

Applying one image-segmentation technique

According to Section 2.2.2.1, image segmentation techniques can be grouped into two categories: region based image segmentation techniques, such as multi-dimension thresholding and edge based image segmentation techniques, such as the Canny edge detector. Several image-segmentation techniques have been tested to find the pixels belonging to the lines. Figure 3-6 shows the outcome of applying some image segmentation techniques to one video frame.

There are two major problems with edge-based line detection compared to region-based line detection (multi-dimensional thresholding):

- Edge-based line detection will outline the contour of one line instead of the line itself. This raises some problem in determining the position of the line. In Ekin and Tekalp (2002), this problem is solved by using specially designed masks, which give higher weight to the horizontal and vertical directions. The masks used in Ekin and Tekalp (2002) are:

Vertical Mask

$$\begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$

Horizontal Mask

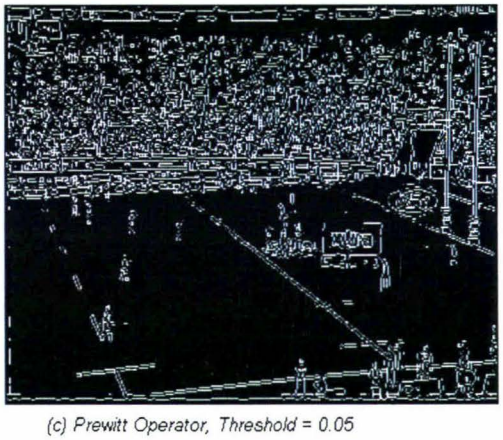
$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix}$$

  In the above masks, vertical lines and horizontal lines are favoured by having a higher factor (2).

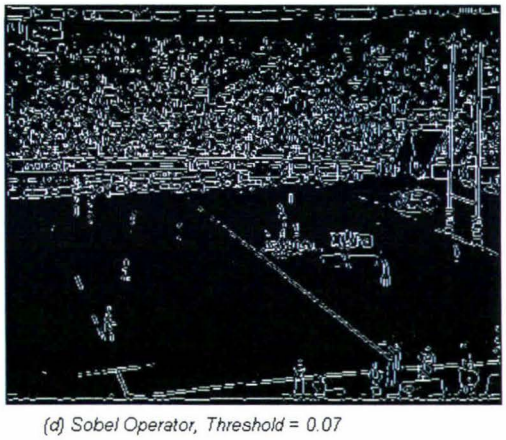
- Due to the complexity of the video frames, a high level of noise exists in terms of the audience and the advertisements. The edge-based image segmentation techniques respond to this noise. To overcome this problem, extra techniques are required, such as the determination of the region of interest.

(a) Original Video Frame

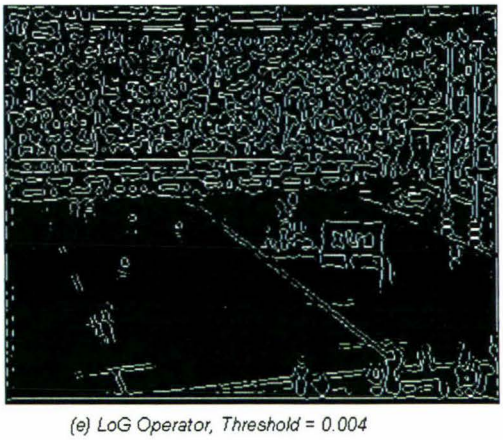
(b) Multi-Thresholding



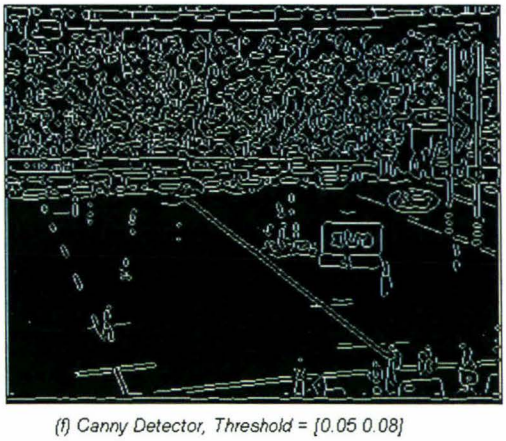
(c) Prewitt Operator, Threshold = 0.05



(d) Sobel Operator, Threshold = 0.07



(e) LoG Operator, Threshold = 0.004



(f) Canny Detector, Threshold = [0.05 0.08]

Figure 3-6 Image Segmentation Results

As can be seen in Figure 3-6, the output of multi-dimensional thresholding is the best among all the test results, for each line in the video frame, only one corresponding line in the output image is recognised. Yet the problem of this approach is that good thresholds are not easy to find. As mentioned in Section 2.2.2.1, in some cases, multi-dimensional thresholding may not be able to isolate two colors. In Figure 3-6 (b), thresholds are found in both RGB and HSV color spaces. That means a set of 12 variants are required (in the example below, multi-dimensional thresholding in R, G, B, and H space is sufficient to get the result, so only 8 variants are given):

$$120 \leq R \leq 170;$$
$$150 \leq G \leq 200;$$
$$80 \leq B \leq 130;$$
$$0.18 \leq H \leq 0.25;$$
$$0 \leq S \leq 1;$$
$$0 \leq V \leq 1;$$

One simple way to help deciding the thresholds is to ask a human operator to manually mark the lines in the video frame. The marked pixels are then grouped together for analysis. Histograms of the pixels on the line are calculated in the RGB and HSV color spaces. According to the histograms, the human operator can determine the upper and lower thresholds in each color space.

Applying a combination of several image-segmentation techniques

Instead of using only one image-segmentation technique, multiple image-segmentation techniques can be used together to get a better result.

In Ekin and Tekalp (2002), a region of interest is determined by finding a dominant color in the Hue space of the image, as this dominant color represents the color of the grass. Then the image is convolved with special horizontal and vertical edge detector masks to find out the pixels having high RGB values. The combination of the color segmentation and the gradient magnitude gives the field lines in the image. Figure 3-7 shows the result of applying the segment technique described in Ekin and Tekalp (2002) to segment one video frame. The masks have been refined to:

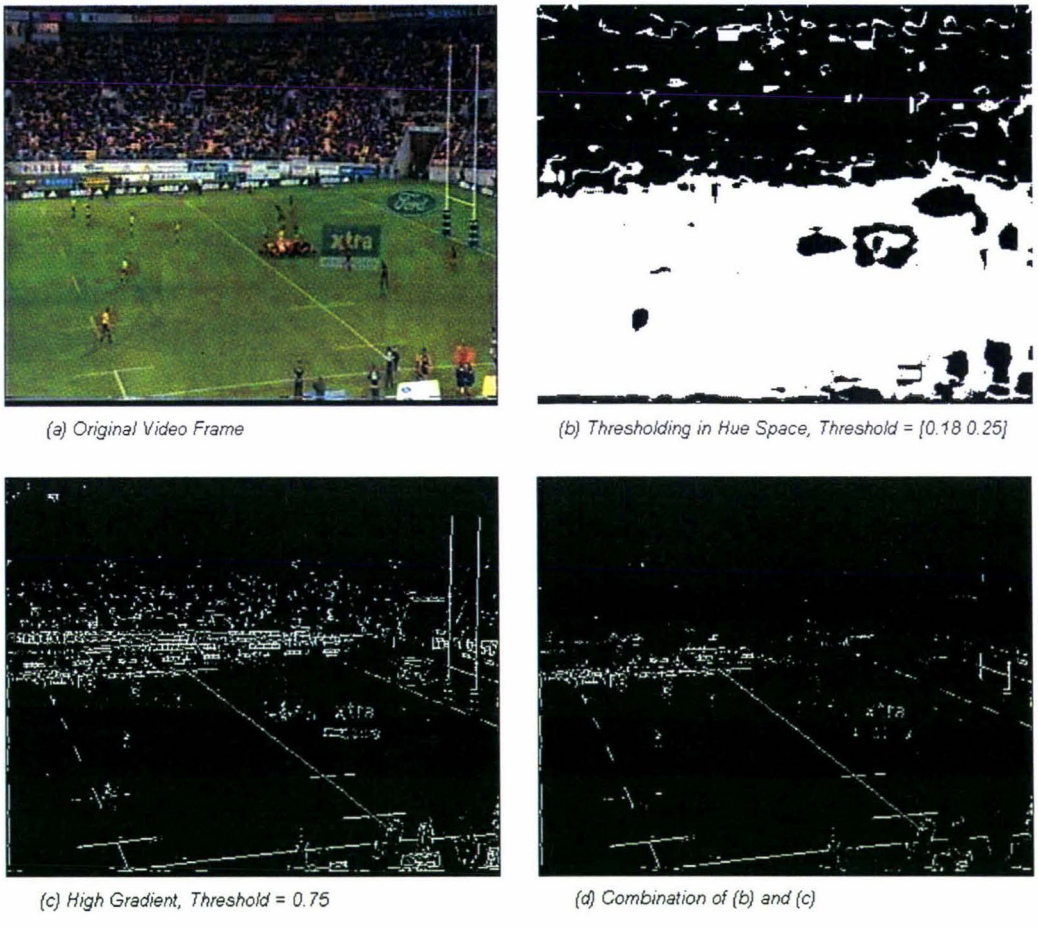
Vertical Mask

$$\begin{bmatrix} -1 & 3 & -1 \\ -1 & 8 & -1 \\ -1 & 3 & -1 \end{bmatrix}$$

Horizontal Mask

$$\begin{bmatrix} -1 & -1 & -1 \\ 3 & 8 & 3 \\ -1 & -1 & -1 \end{bmatrix}$$

In the above masks, the central point is given an extra high factor (8) to make sure that if the pixel has high RGB value, it will have a strong response in the output of the convolution; and vertical lines and horizontal lines are favoured by having a high factor (3).



(a) Original Video Frame

(b) Thresholding in Hue Space, Threshold = [0.18 0.25]

(c) High Gradient, Threshold = 0.75

(d) Combination of (b) and (c)

Figure 3-7 Segmentation using Techniques in Ekin and Tekalp (2002)

As can be seen in Figure 3-2, a typical video frame used in this project normally contains several kinds of objects: the audience, the players, the field characteristics, and the advertisements. Some objects, such as the players on the field, the lines, and the advertisements may superimpose upon the rugby field in the image. This makes it impossible to use one dominant color to correctly

determine the boundary of the field. Due to the fact that there are some white-color advertisements placed in the grass very close to the boundary of the field, there is lots of noise in the output close to the upper touch line. Adding more criteria to decide the dominant color (for example, thresholding in both Hue space and Red space) may reduce the noise, but that requires extra processing time and increases complexity.

Choice of image-segmentation technique

Compared to other image segmentation techniques, multi-dimensional thresholding has only one response to each line in the video frame. It also has less noise and is relatively easy to implement. In this project, multi-dimensional thresholding is therefore chosen as the technique for image segmentation.

Hough transform ---- Edge Linking

To get a representation of a line in the output image of the image segmentation, edge linking needs to be performed. Global linking (Hough transform) is used to find the representation of the lines in polar coordinate system.

As discussed in Section 2.2.2.2, the output of a Hough transform is a matrix representing a Hough space. The value in the position (ρ, φ) shows how many votes this (ρ, φ) pair got during the Hough transform. Figure 3-8 shows part of the Hough space. A maximum value exists in this part of the Hough space, which means a (ρ, φ) pair is getting votes from many points, and therefore is a candidate for a line. By determining its coordinates, the (ρ, φ) pair representing a line in the video frame can be found. Figure 3-9 shows a color-map of the Hough space, where the values in the Hough space are represented as RGB colors, the brighter the color, the greater the value. Each local maximum (strong peak) in Figure 3-9 represents a possible line in the video frame. Detection of these local maxima results in (ρ, φ) pairs representing lines in the video frame.

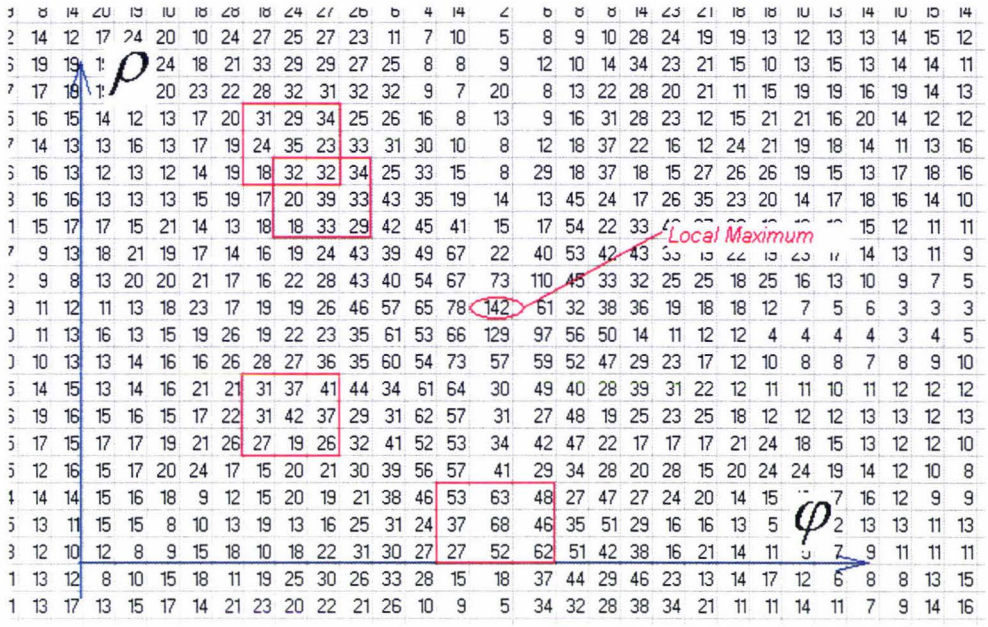


Figure 3-8 Part of Hough Space

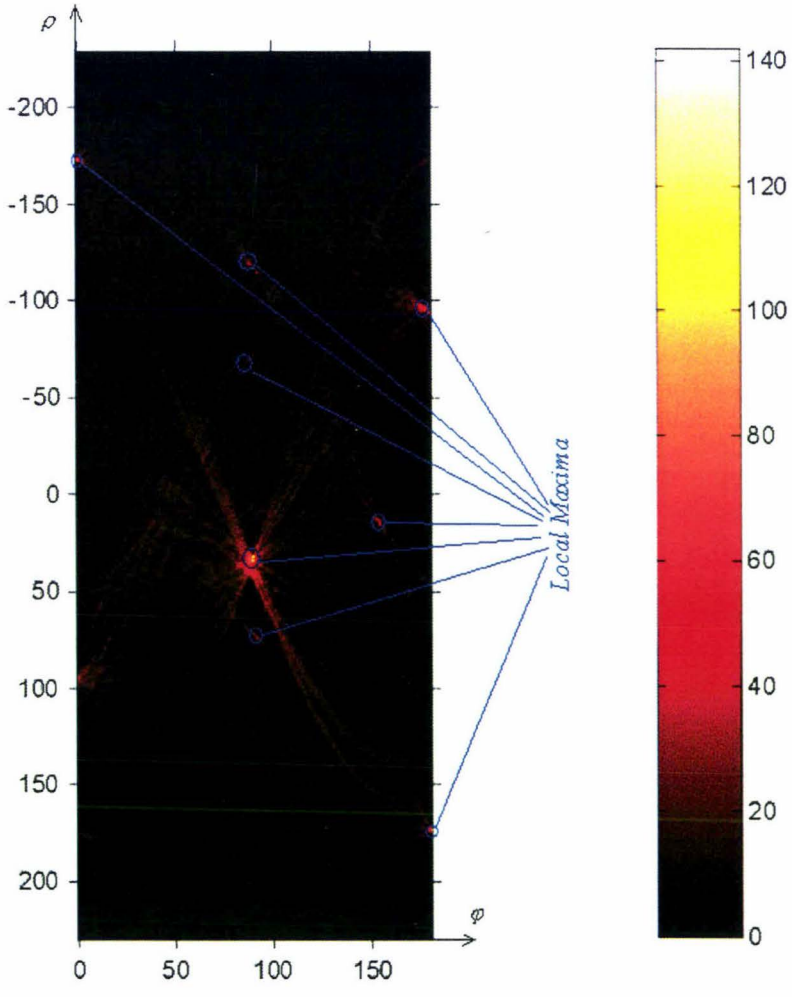


Figure 3-9 The Colormap of Hough Space

3.2.2.2.2 Identifying Lines

For the first video frame, the human operator is asked to name the lines detected by the Hough transform. The manually identified lines then can be tracked in the consecutive frames. Figure 3-10 shows the flow chart of manually identifying lines in one video frame.

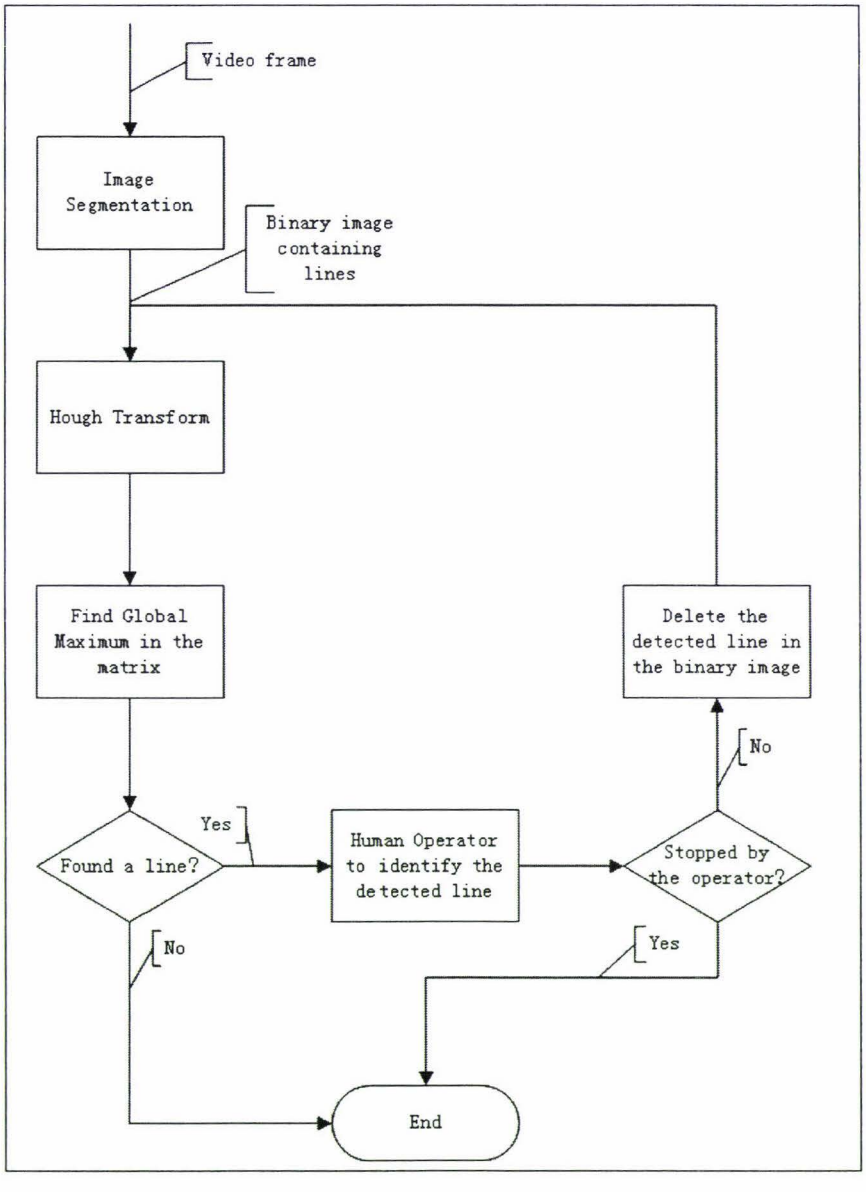


Figure 3-10 Flow Chart of Manually Identifying Lines

Firstly, multi-dimensional thresholding is done to the input video frame. As described above, the output of multi-dimensional thresholding is a binary image, in which the pixels in the lines being '1', and other pixels being '0'. The '1's in the binary image cast

their votes to the Hough space during the Hough transform. The output matrix is relatively large, for a 352x288 input image, the dimension of the output matrix is 459x181 (ρ between –228 and 228, φ between 0 and 180), that is, containing 83079 values. One possible way of finding local maxima is to compare the value in the matrix with its surrounding values. If it is greater than all its neighbours, the value may indicate a local maximum. As the number of the '1' pixels in the image is large, this approach is not appropriate as it will return thousands of (ρ, φ) pairs. For example, there is a peak value at the centre of each red rectangle in Figure 3-8. In this research, this is done by determining the global maximum of the matrix. The indices ((ρ, φ) pair) at this global maximum denote the most significant line in the video frame. This line is then superimposed on the original video frame. The human operator observes the modified video frame and determines which line is presented (for example, in Figure 3-11, the first line is called 'TouchUp'). After that, the pixels in the just detected line are deleted (set to '0') from the binary image. Now the second significant line in the binary image becomes the most significant line (for example, in Figure 3-11, the second line is caused by caused by noise). Applying Hough transform to the modified binary image will result in a new matrix, in which the indices of the global maximum denote another line in the video frame. After the second line being identified, this line is also deleted in order to get the next line (for example, in Figure 3-11, the third line is called 'HalfWay'). This process will continue until there are no more '1' pixels in the binary image or it has been stopped by the human operator. Figure 3-11 shows the result of superimposing all detected lines on the original video frame.

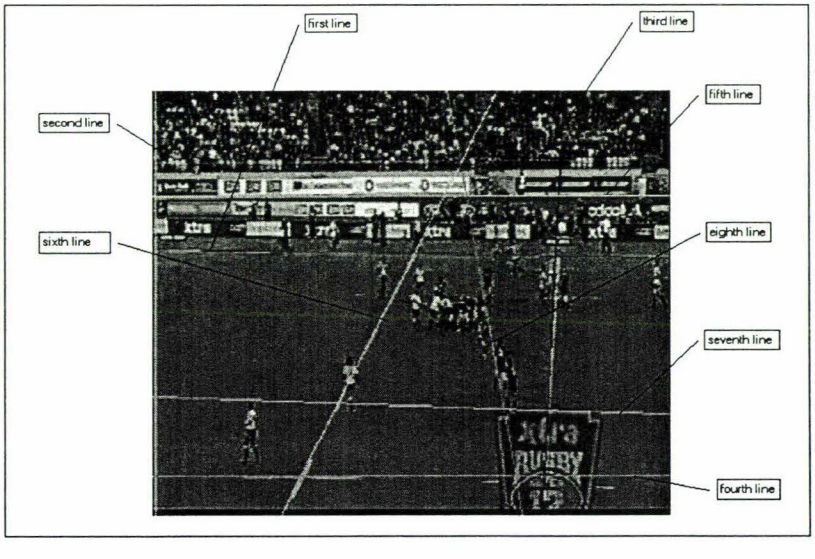


Figure 3-11 Line Identification Results

There are two variants in the processing of line identification:

- The minimum value accepted in finding the global maximum in the output matrix of Hough transform (*minValDet*). The application will stop if the global maximum is below the predefined minimum value. This parameter can be used to reduce the effect of noises, that is, if the global maximum is too small, it is caused by noise. But if this variant is set too big, some good lines may be discarded.

- The error range in which the pixels will be treated as lying in the line (*maxDist*). Obviously, not all the pixels voting to the (ρ , φ) pair in Hough space lie exactly on the line, that is, the equation $\rho = x \cdot Cos(\varphi) + y \cdot Sin(\varphi)$ is not always true for all the (x, y) pairs voting for the line. This can be caused by many reasons, such as lines being distorted or being not straight due to distortion or noise lying close to the lines caused by image segmentation. To determine whether the pixel (x, y) lies on the line (ρ , φ), the value $x \cdot Cos(\varphi) + y \cdot Sin(\varphi)$ is calculated, and compared with the ρ. If the difference between these two values is less than *maxDist*, this pixel is treated as lying on the line and will be deleted when deleting the line in the binary image. The parameter should be judged carefully, if it is too small, the same line will be detected several times, thus duplicating the work of the human operator; if it is set too big, some lines may be lost as their pixels are deleted when deleting other lines.

3.2.2.3 Line Tracking

The positions of identified lines in one frame can be used to identify the same lines in the consecutive frame. Tracking of lines is based on the fact that in continuous video recordings of rugby games, the motion of lines is continuous. So the difference in both ρ and φ between the same lines in two consecutive frames falls in a very small range. That means if a local maximum exists in (ρ , φ) in the Hough space of the previous frame, there is a local maximum somewhere very close to (ρ , φ) in the Hough space of the current frame. Table 3-2 shows an example of the information of lines identified

manually in two consecutive frames. The differences of lines in both ρ and φ in the example are within one unit.

Line No.	Frame 0		Frame 1		Differences	
	Angle	Distance	Angle	Distance	Angle	Distance
11	52	-25	51	-24	-1	1
12	96	-128	96	-129	0	-1
2	93	7	93	8	0	1
10	96	-108	96	-109	0	-1
9	24	-136	24	-137	0	-1
13	67	35	67	36	0	1
4	93	2	93	3	0	1
6	93	-8	93	-7	0	1
8	95	-78	95	-78	0	0

Table 3-2 Lines in Two Consecutive Frames

Figure 3-12 shows the flow chart of line tracking in one frame. As defined in Section 3.2.2.1, there are thirteen identifiable lines in the rugby field. All the lines are tested one by one to see if their new positions can be found automatically. If the new position of the line can be found, the new information of the line is kept, and later may be used to calculate the transform matrix. Parameters required in tracking of lines are:

- The maximum differences (in degree) allowed in angle of the line compared to the previous frame (*diffAng*) and the maximum distance (in pixels) allowed in distance of the line compared to the previous frame (*diffDist*). In polar coordinate system, a line is defined by two variants: the angle (φ) and the distance (ρ). The *diffAng* and *diffDist* define how far the line can move during two consecutive frames. The searching of the new position of a line is limited to a small range defined by these two parameters. For example, if the old position of a line is defined by (ρ_{old}, φ_{old}), the new position of the line lies in a rectangle area centred at (ρ_{old}, φ_{old}). The top-left vertex of the rectangle is (ρ_{old}-*diffDist*, φ_{old}-*diffAng*), and the right-bottom vertex of the rectangle is (ρ_{old}+*diffDist*, φ_{old}+*diffAng*). The local maximum (ρ_{new}, φ_{new}) of this rectangle area in the output matrix of the Hough transform is treated as the new position of

the line. These two parameters, diffAng and diffDist, are essential in the tracking of lines. If the range is too small, the local maximum may be lost or incorrect; if the range is too large, two closely positioned lines may be mapped to one line.

- The minimum value in the output of Hough transform allowed for automatic tracking of lines (*minValTrk*). When the local maximum in the rectangle area is found, the value is compared to *minValTrk*. If it is below this threshold, tracking of this line fails. Similar to *minValDet*, this value can be used to reduce the effect of noise. For example, due to the movement of the camera, one line has moved outside the video frame and has become invisible, but there is some noise in the new frame close to the old position of the line. The local maximum in the range is found and is close to zero due to the noise. This local maximum can be discarded by comparing to the *minValTrk* value.

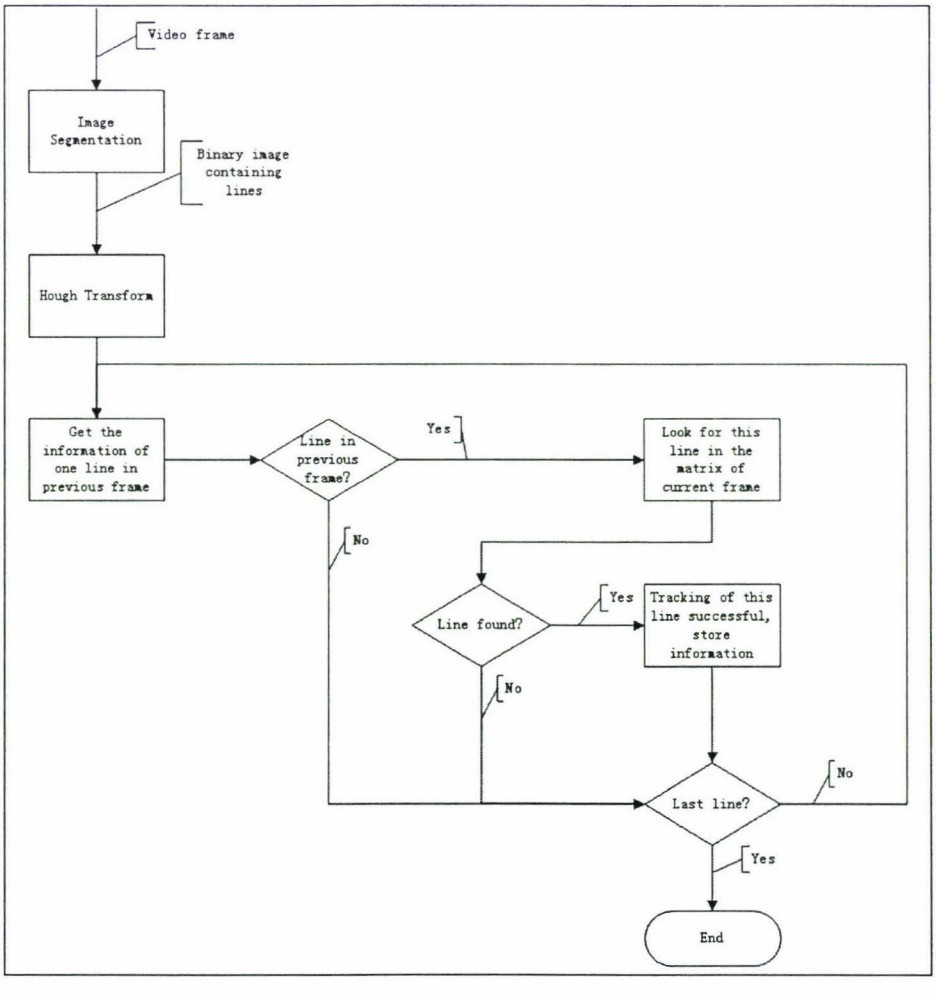


Figure 3-12 Flow Chart of Line Tracking

According to the procedure of line tracking, there are three variants for each line to be kept for future use: the angle φ, the distance ρ, and whether the line is visible in the video frame. Not all the lines visible in the video frame can be successfully detected by tracking. Some may be lost in the image segmentation stage and some may be discarded due to their values in the matrix being below *minValTrk*. These lost lines can be added back after the transform matrix has been found, thus increasing the possibility of successfully tracking all the visible lines in the next frame. The calculation of the transform matrix is introduced in the next section.

3.2.2.4 Coordinate Transform

After the lines in the video frame have been named by the human operator or identified by means of line tracking, the transform matrix used to convert the pixels in the image into the positions on the rugby field can be calculated.

As discussed in Section 2.2.3.1, perspective transform model can be used to find the mapping from camera plane to real world plane. In order to calculate the perspective transform matrix, four lines are required as reference. Searching is first performed among the lines, which their representation in the video frame are known after line detection and tracking, to find the reference lines. For the starting frame (first frame or the frame at which the processing is restarted) of the processing, the information from the previous frame is not available, human operator manually names the lines. Four reference lines (left, right, up, down) must be decided for these video frames. The algorithm of calculating the perspective transform matrix based on four reference lines is called four lines algorithm in this thesis, it has been described in Section 2.2.3.1. For video frames in which the lines are tracked using information from the previous frame, it is possible that the number of identified lines is less than four. Only using four lines algorithm would require human intervention in this situation. To reduce the number of human intervention, an algorithm has been developed to use three reference lines and the information from previous frame to calculate the perspective transform matrix. This algorithm is called three lines algorithm and is discussed later in this section. Therefore, for video frames in which the lines are tracked using information from the previous frame, at least three reference lines (two vertical and one horizontal or two horizontal and one vertical) must be found. Failure to find enough reference lines will fail the calculation of the transform matrix, thus requiring a restart of the processing. If sufficient reference lines are found, calculation of transform matrix can be performed. According to the number of reference

lines available, the calculation can base on the four lines algorithm or three lines algorithm. Figure 3-13 shows the flow chart for calculating the transform matrix.

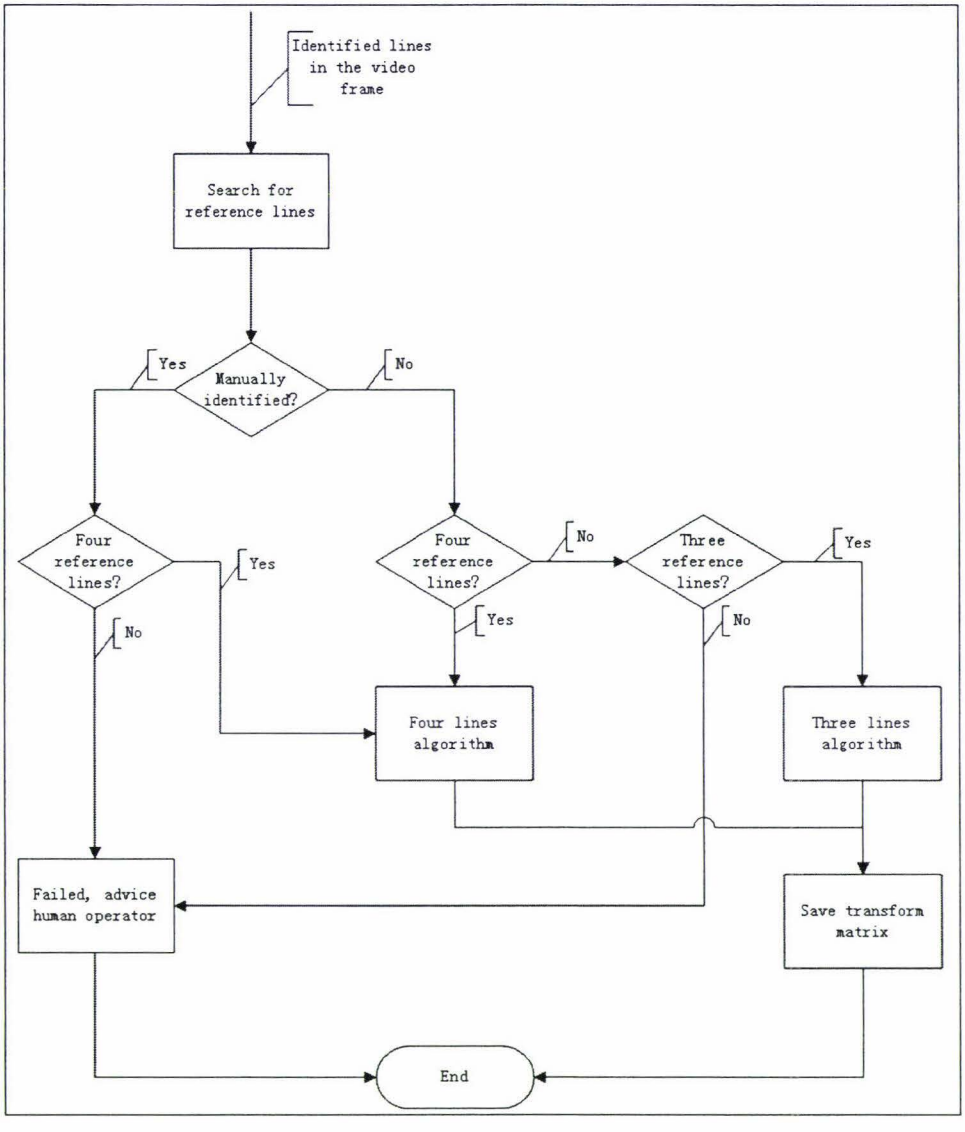


Figure 3-13 Calculation of Transform Matrix

❖ Searching for Reference Lines

One important function in coordinate transformation is to find lines as references. As described in Section 2.2.3.1, the transform matrix of a 2D perspective transform can be calculated if there are four lines, whose representation in both coordinate systems are known. As has been discussed in Section 3.2.2.1, by defining the coordinate system in the rugby field, the representations of the lines are known (refer to Table 1 for representations of lines in homogeneous coordinate system). So the selection of reference lines depends on the following questions:

- How many lines are visible in the video frame? Video frames with at least four lines (two vertical and two horizontal) are able to generate the transform matrix by itself. Video frames having three lines visible (two vertical, one horizontal or two horizontal, one vertical) may be able to find a transform matrix if the transform matrix of previous frame is known.

- What is the accuracy of the lines detected and identified? Are they of the same quality? Manually identified lines usually have a higher accuracy because if the line is not accurate, the human operator can adjust it. So at the beginning of the tracking, the identified lines are the preferred reference lines, the number of switching of reference lines should be as low as possible.

- Which line is preferred? In many cases, there are more than four identified lines. Testing of different lines as reference to calculate the transform matrix has shown that for lines with similar accuracy, a bigger reference rectangle (as formed by the four reference lines) will result in a more accurate transform matrix.

Based on the above analysis, some rules are formed for the selection of the reference lines:

- Because the lines usually have high accuracy in video frames with manually identified lines (the human operator can even manually draw the line to make refinements), the identified lines in these frames can be seen as having similar quality. Therefore, in manually identified video frames, the selection of reference lines depends on the position of the identified lines. That is, among manually identified lines, the four that form biggest rectangle are selected.

- In video frames with tracked lines, keep the same reference line if possible. A reference line will no longer be preferred if the reference line is not tracked successfully or its value in the new position in the output matrix of Hough transform is below a predefined value (*preferredRef*). To make it simple, in this project, this predefined value is set to twice the amount of *minValTrk*, that is, $preferredRef = 2 \times minValTrk$.

- If the number of reference lines after the above process is less than three, the missing reference lines are found by looking for significant lines. For example, if

the left line is missing, go through the tracked lines to see if there is a vertical line with the value in the new position in the output matrix of Hough transform that is above the predefined value (*preferredRef*). If such a line can be found, use it as the left reference line.

- If the number of reference lines after the above process is still less than three, the missing reference lines are determined by looking for tracked lines. For example, if the left line is missing, go through the tracked lines to see if there is a vertical line that has been tracked successfully. If such a line can be found, use it as the left reference line.

- Finally, if the number of reference lines is still less than three, the human operator will be asked to handle the situation. Intervention can be done to increase the possibility of successful line tracking and calculation, such as adjusting the thresholds in multi-dimensional thresholding, or changing parameters in line tracking.

❖ Four-lines Algorithm

If the selection of reference lines is successful, four lines are chosen to calculate the transform matrix. The calculation follows the steps described in Section 2.2.3.1. The four reference lines must be two horizontal lines and two vertical lines, otherwise the calculated matrix will be singular.

❖ Three-lines Algorithm

Sometimes the selection of reference lines produces the result that only three reference lines are available. This may be because there are only three lines visible in the video frame, or because other lines in the video frame are very weak or blurred by noise. Discarding these video frames would result in a big gap in the tracking of lines. Furthermore, the processing of video frames would become discontinuous and the work of the human operator would be greatly increased because the human operator has to manually name lines for discontinuous frames.

To overcome this problem, an algorithm has been developed using three reference lines based on the information about these reference lines and the transform matrix in the previous frame. Theoretically, to register two frames capturing the same rugby field, a perspective transform need to be performed. So a three-by-three transform matrix is required, which means eight degree of freedom. But because there is only a small change

in two consecutive video frames, the succeeding frame can be thought as a result of applying a combination of several simple 2D transforms to the anterior frame. By modelling the transform with an affine transform model, the motion of the frame combines translation, rotation, scaling, and shear (Section 2.2.3.2 gives an overview of affine transform). The transform matrix of affine transform has six degrees of freedom, which means only three lines are required to calculate the matrix. After registration of the consecutive frame to the anterior frame, the transform matrix for perspective transform of the consecutive frame can be calculated by convolving the affine transform matrix and the perspective transform matrix of the anterior frame. The steps for the calculation are:

Assuming the affine transform matrix is:

$$H_{affine} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & k \end{bmatrix}$$

and the representations of one line in the i^{th} and the j^{th} ($j = i + 1$) frame in homogenous coordinate system are:

$$L_i = [a_i \quad b_i \quad c_i]$$

$$L_j = [a_j \quad b_j \quad c_j]$$

so for each point and line, the equation is:

$$P_i = H_{affine} P_j; \quad \text{so:}$$

$$L_j = L_i H_{affine}; \quad \text{that is,}$$

$$[a_j \quad b_j \quad c_j] = [a_i \quad b_i \quad c_i] \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & k \end{bmatrix}$$

Two equations can be developed from this:

$$\begin{bmatrix} a_j & 0 & a_i a_j & b_j & 0 & a_i b_j & -a_i \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ t_x \\ a_{21} \\ a_{22} \\ t_y \\ k \end{bmatrix} = 0 \text{ and}$$

$$\begin{bmatrix} 0 & a_j & a_j b_i & 0 & b_j & b_i b_j & -b_i \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ t_x \\ a_{21} \\ a_{22} \\ t_y \\ k \end{bmatrix} = 0$$

For three lines, six equations can be established, and set the k to one:

$$\begin{bmatrix} a_{j1} & 0 & a_{i1}a_{j1} & b_{j1} & 0 & a_{i1}b_{j1} & -a_{i1} \\ 0 & a_{j1} & a_{j1}b_{i1} & 0 & b_{j1} & b_{i1}b_{j1} & -b_{i1} \\ a_{j2} & 0 & a_{i2}a_{j2} & b_{j2} & 0 & a_{i2}b_{j2} & -a_{i2} \\ 0 & a_{j2} & a_{j2}b_{i2} & 0 & b_{j2} & b_{i2}b_{j2} & -b_{i2} \\ a_{j3} & 0 & a_{i3}a_{j3} & b_{j3} & 0 & a_{i3}b_{j3} & -a_{i3} \\ 0 & a_{j3} & a_{j3}b_{i3} & 0 & b_{j3} & b_{i3}b_{j3} & -b_{i3} \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ t_x \\ a_{21} \\ a_{22} \\ t_y \\ k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

By solving the equation, the affine transform matrix can be formed. The perspective transform matrix used in frame j then can be calculated as below:

$$L_i = L_{field} H_i^{-1}$$

$$L_j = L_{field} H_j^{-1}$$

and

$$L_j = L_i H_{affine}$$
$$= L_{field} H_i^{-1} H_{affine}$$

so

$$H_j^{-1} = H_i^{-1} H_{affine}$$

that is

$$H_j = H_{affine}^{-1} H_i$$

In this way, the problem with video frames having only three reference lines can be solved.

❖ Storing/Showing the Results

The results of detection and tracking of lines and the perspective transform matrix need to be stored for future use. They can be used to calculate the perspective transform matrix in the consecutive frame and later to transform pixel positions in the video frame to the positions in the rugby field.

To help finding the reference lines, information of lines in the previous frame is very important. Besides the angle φ and the distance ρ, the roles of each line in the previous frame are also required in deciding the references. The role of each line in calculating the perspective transform matrix needs to be kept for future use. When superimposing the lines on the original video frame, different colors are used so that the roles of the lines can be identified in the output image. This also helps the human operator to monitor the

quality of the processing. Table 3-3 shows an example of the definition of roles and the colors used.

Role ID	*Description*	*Color*
0	Manually identified line	Cyan
1	Manually identified line, and being used as reference line	Blue
2	Tracked line	Magenta
3	Tracked line, and being used as reference line	Red
4	Failed in tracking, but in the video frame, the position of the line is calculated using the transform matrix	Black
5	Invisible, outside the video frame	N/A

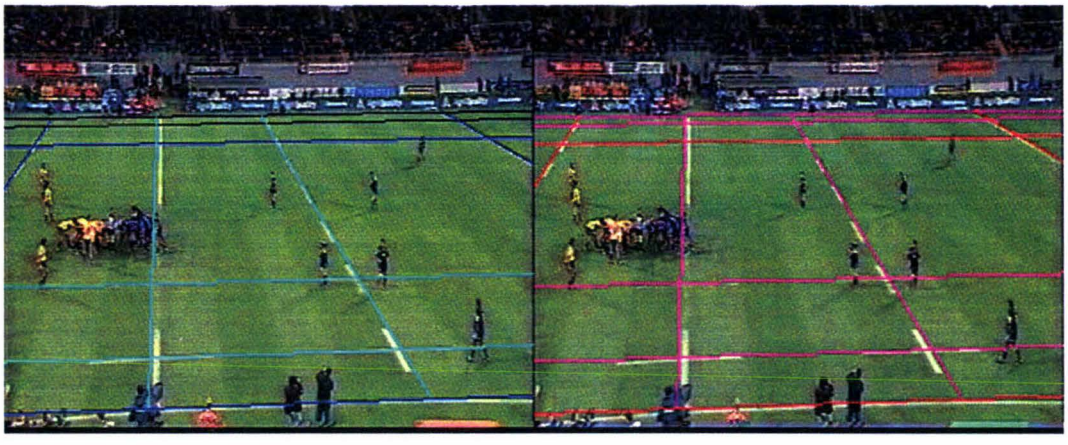
Table 3-3 Roles of Lines and Their Color Representations

3.2.2.5 Summary of Line Detection and Tracking

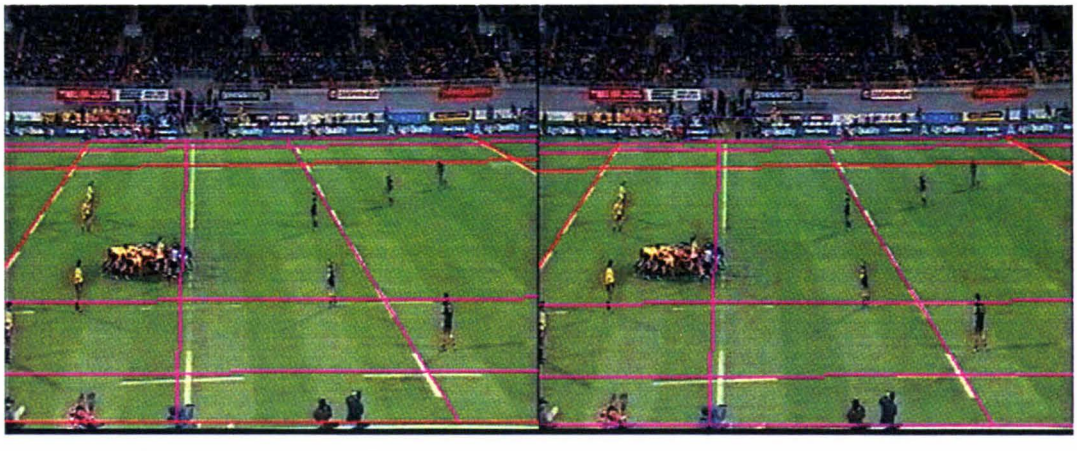
In Section 3.2.2, the algorithm to register the video frame to the rugby field is explained. The selection of the image processing techniques applied is also discussed.

Among all the image segmentation techniques, multi-dimensional thresholding is chosen to get a binary image, which contains only pixels lying on the lines. Hough transform is used to generate an output matrix, in which the lines form local maxima. In case of manually identifying lines, an iterative process, which involves Hough transform and finding global maxima, is used to retrieve the position of the lines and temporally remove the identified lines. During the auto tracking of lines, the positions of lines are retrieved by finding local maxima in the areas defined by the lines in previous frame. Reference lines are lines used to calculate the perspective transform matrix, and they are found under some predefined rules. According to the number of reference lines, different algorithms are used to calculate the transform matrix.

Figure 3-14 shows an example of the line tracking process (see Table 3-3 for meanings of colors).



(a) Frame 0, manually identified (b) Frame 1, auto tracking



(c) Frame 108, four reference lines (d) Frame 139, three reference lines

Figure 3-14 Example of Line Tracking

In frame 0, lines are named manually; the lines near the boundary are chosen as reference lines. In frame 1, auto tracking is performed to identify lines. In frame 108, still the same lines are used as reference lines. In frame 109, because the bottom touchline has moved out of the video frame, it is no longer used as reference line. The three-lines algorithm is used in this case to calculate the perspective transform matrix.

3.2.3 Testing the Algorithms

Once the algorithms will have been implemented, they will have to be tested for accuracy. The purpose of testing the developed algorithms is to find out their suitability for the task and any potential problems. The testing mainly involves research in two directions: the quality of line tracking and the efficiency of line tracking.

3.2.3.1 The Quality of Line Recognition and Tracking

The purpose of this part of the testing is to find out how accurate the outcome will be. Although the human operator will get an impression of the quality of tracking when a video frame is superimposed with the lines, it is useful to establish a more formal approach so that statistics can be developed.

➢ **What to Look For**

Many barriers exist in getting accurate measurements for the quality of the algorithms.

One problem with measuring quality of the algorithms is that testing of all the pixels belonging to the rugby field area in the video frame is not realistic. Theoretically, for every pixel that belongs to the rugby field area in the image, testing needs to be done. But this approach is not realistic. One problem is that the number of pixels belonging to the rugby field in the image is very large and hard to determine. Furthermore, it is impossible to retrieve the reference data for every pixel. For a pixel in the image, how could its true position in the field be determined? The solution could be placing reference objects in the rugby field, for example, mark vertical and horizontal lines every meter in the field. This approach is not realistic because it is impossible to draw marks on a rugby field where a game is played. Alternatively, the exact position and parameters of the camera are measured, as is done in Perš and Kovačič (2001). Due to the use of televised video clips, the parameters of the camera are unknown in this project. Therefore the testing has to be limited to pixels that have known references in the rugby field. Because only intersections of lines in the rugby field have known and fixed position (coordinates in rugby field), the testing focuses on the accuracy of the intersection of the lines in the rugby field. A human operator is required to determine the 'real' intersections of the lines. There are several features to look for:

- The deviation in one pixel. If the pixel (P') in the intersection is given, the coordinates of the pixel in the video frame are converted to position (P) in the rugby field. The distance (in meters, called point deviation, D_P) between the position P and its true position (P^t) is calculated and use as the deviation in this pixel. That is:

$$P = H^{-1}P'$$

$$D_p = \sqrt{\left(P_x - P_x^t\right)^2 + \left(P_y - P_y^t\right)^2}$$

- For one video frame, an average value (in meters, called frame deviation, *FD*) of deviations of all the tested points is calculated:

$$FD = \sum_{i=1}^{N} D_i$$

where N is the number of tested points in the frame

- For one video clip, an average value (in meters, called clip deviation, *CD*) of frame deviations of all the tested frames is calculated:

$$CD = \sum_{i=1}^{N} FD_i$$

where N is the number of tested frames in the clip

Another problem is that the testing can only be done to the final outcome of the algorithm. That means the detected deviation cannot be divided into the many steps that took place during the processing of the video frame. For example, the quality of the image segmentation is not measurable. It is not realistic to measure how many percent of the pixels in the line have been extracted successfully or to count how many percent of the deviation are contributed by the image segmentation. However, the human operator can observe the outcomes in different steps, and can give a report on the problems in processing the video frames. To get statistical data on the main barriers for successful line recognition and tracking, four problems are observed and recorded:

- Bad thresholds: caused by the thresholds selected for the multi-dimensional thresholding (image segmentation). The multi-dimensional thresholding uses thresholds to filter out lines in the image but noise always exist. The multi-dimensional thresholding technique may fail to distinguish a line from its background. Serious bad threshold problems lead to omitting lines or to wrongly detecting lines.

- Lens distortion: caused by the quality of the lens used to capture the video. One of the most common forms of lens distortion is barrel distortion. It results from the lens having a slightly higher magnification in the centre of the image than at the periphery (Bailey, 2002).

- Close reference lines: points outside the rectangle formed by the references lines may have large errors compared to those inside the rectangle. If two reference lines are too close to each other (usually this makes the rectangle very narrow), slight errors in line detection may cause big errors in points near the boundaries of the image.

- Shadows: in some video clips, part of the field is covered by the shadow of the stadium roof. It is impossible to find one single set of thresholds to filter out lines in both the light and shadow parts of the field.

An investigation on the statistic data of the problems will result in establishing the main barriers affecting the quality of the algorithm.

➢ **When and How to Measure**

The measurement of quality can be performed during or after the processing of the video clip. Because the testing requires human operator input, testing during the tracking of lines may cause the line tracking halt waiting for human operator intervention. However, testing during the processing of the video clip makes it easier to detect possible problems and make refinements. So if the time consumption is not an important issue, testing during the processing can be used to increase the quality of the output of the algorithms. Testing can also be done after the processing of the whole video clip. In this way, statistical data can be produced.

The measurement of the quality requires the intervention of a human operator. The human operator must apply the same rules for the testing to all the video clips; otherwise the measurement is meaningless. These rules include:

- Follow same routine to intervene in the line tracking process. If the human operator restarts very often, the processing will be refined in every restart, so the accuracy will become much higher.

- Intersections of lines are marked manually in the video frame. The human operator must keep the marking of the intersections of the lines as accurate as possible. In marking the intersections, errors may arise simply due to the pixel the human operator marked is not exactly at the intersection of the lines.

- The human operator also observes problems with line recognition and tracking. To get meaningful statistical data on the main barriers affecting line recognition and tracking, the human operator needs to follow the same routine during the observation.

3.2.3.2 Efficiency of Line Recognition and Tracking

The main task in assessing the efficiency of the algorithm is to measure time consumption. The end-users of the developed system are the coaches, who will want to see reports on player positions throughout the game as soon as possible. This makes the measurement of efficiency necessary because the line recognition and tracking system will be an important part of the final player tracking system. Two sets of information can be recorded:

- The overall time required for processing one video frame: The time used in manually identifying lines is not calculated as it may vary according to the human operator and the video frame. The time required for processing auto-tracked frames provides important data. This can be measured by simply calculating the time interval between two auto-tracked frames.

- The distribution of time over each sub-process of the processing: This information is very useful if the developed algorithm needs to be refined to increase the speed of processing.

3.2.4 Summary of the Proposed Solution

In Section 3.2, the proposed solution of the task is explained. There are three modules in the proposed approach. The pre-processing module prepares the input data for the line detection and tracking module. The line detection and tracking module uses image-processing techniques to calculate a perspective transform matrix for every frame in the video clip. The testing module assesses the quality and efficiency of the algorithms used in the line detection and tracking module.

3.3 Requirement Specification

Once the algorithms have been designed, an application can be developed to implement these algorithms. Based on the discussions in Section 3.1 and Section 3.2, the major requirements of the application can be summarised as:

Requirement 1. Pre-processing needs to be performed to get an image sequence from the video clip.

Requirement 2. Functions must be developed to manipulate the JPG-format image files. The application needs to be able to read, display and save JPG-format image files.

Requirement 3. An interface must be provided to allow a human operator to define the source and destination of the processing, that is, which image sequence is to be processed and where the resulting image files (the video frames superimposed by the lines) are stored.

Requirement 4. Tools are required to help the human operator to decide the thresholds used in image segmentation.

Requirement 5. The application must provide an interface for changing or loading settings used in line recognition and tracking.

Requirement 6. The application must provide an interface for changing or loading color settings used to represent the different roles of lines in the line tracking. This will allow the user to choose favourite colors for the superimposed lines in the output image.

Requirement 7. Functions and interfaces need to be developed for line detection and tracking. The functions include image-segmentation functions, Hough-transform functions, line tracking, reference-lines selection, and transformation-matrix calculation. The interfaces are for human operator to name the detected lines.

Requirement 8. A database must be designed and used to store the information produced by the line recognition and tracking process.

Requirement 9. An interface must be provided for the human operator to test the quality of line recognition and tracking.

Requirement 10. Time consumption must be recorded and displayed in the main interface of the application.

Requirement 11. Statistical data produced by the testing must be accessible. An interface must be provided for the human operator to view the results.

3.4 Summary

In this chapter, the algorithms for line recognition and tracking are introduced. Three modules form the proposed solution for the task.

The pre-processing module extracts frames from the video clip, and saves them in JPG files.

The line detection and tracking modules process video frames one by one. Several steps are involved in this stage. Multi-dimensional thresholding is selected as the technique to segment the input image. The extracted pixels in multi-dimensional thresholding are used to calculate the matrix for the Hough transform. Finding the local maxima in the matrix generates the representations of the lines. A human operator is required to name the lines detected in the first frame, and then tracking is performed to automatically follow the detected lines. The calculation of the perspective transform matrix requires the selection of three or four lines as reference. Two algorithms, based on three or four reference lines respectively, have been developed to calculate this matrix.

The testing module evaluates the accuracy of the calculated transform matrix and helps finding processing problem.

Finally, in Section 3.3, the requirements for the development of an application based on the algorithms introduced are summarised.

4 Design and Implementation of the Application

In this chapter, the implementation issues of the proposed system will be discussed. Section 4.1 briefly discusses the required hardware. Section 4.2 introduces the software tools used in developing the application. In Section 4.3, the design and implementation issues arising from each of the requirements of the application are introduced.

4.1 Hardware Requirements

Because the video clips are taken from televised rugby games, no extra hardware is required for capturing the video clips. All the image-processing functions are implemented in software, no special hardware is required by the system. However, different computer systems do have different performance in running the algorithms required for image processing. In the testing of the efficiency of the algorithms, the computer system running the application must be considered.

4.2 Software Selection

Three categories of software are required for this application. One is the image-processing library. The next is the programming language, that is, the development environment of the application. The third is the database system used in the application.

4.2.1 Image-processing Library Selection

Using an existing image processing library will greatly reduce the work of programming and at the same time improve the performance of the application. In this section, two image-processing libraries will be introduced: the Intel Image Processing Library and the MATLAB Image Processing Toolbox.

- **Intel IPL**

The Intel Image Processing Library (2002) (IPL) is a set of C functions for performing typical image-processing tasks. The library functions ensure high performance when run on Intel processors, especially on those with MMX™ technology and the latest processor generations. The image-processing library now has been integrated into the Intel Integrated Performance Primitives (2004) (IPP). The latest version of IPP (4.0, as on July 2004) provides functionality including general signal, image, speech, graphics, text

strings and audio processing, vector manipulation and matrix mathematics, as well as more sophisticated primitives for construction of audio, video and speech codecs such as MP3 (MPEG-1 Audio, Layer 3), MPEG-2, MPEG-4, H.263, JPEG, JPEG2000, GSM-AMR* and G.723, plus computer vision.

The IPP in the image-processing domain (called Intel Integrated Performance Primitives for Image Processing, IPPI) provides functions designed to perform operations with images, video, and video coding. They can be grouped by functionality. Some of the categories are: image geometric transforms, image arithmetic and logical operation function, threshold and compare functions, and filtering functions.

The main benefits of selecting IPL as image processing library would be the performance and the compatibility with Intel CPUs.

- **MATLAB Image-processing Toolbox**

MATLAB (2002) stands for MATrix LABoratory. It is a high-performance numerical computation and visualization software developed by MathWorks, Inc. MATLAB integrates matrix computation, numerical analysis, signal processing, data analysis, and image processing in an easy to use environment. It also features a family of application-specific solutions called toolboxes. Toolboxes are libraries of MATLAB functions that customize MATLAB for solving particular classes of problems.

The basic data structure in MATLAB is the array, an ordered set of real or complex elements. This object is naturally suited to the representation of images, real-valued ordered sets of color or intensity data. MATLAB stores most images as two-dimensional arrays (i.e., matrices), in which each element of the matrix corresponds to a single pixel in the displayed image. This convention makes working with images in MATLAB similar to working with any other type of matrix data, and makes the full power of MATLAB available for image processing applications. The functions in image processing toolbox of MATLAB are grouped into different categories, such as: image input, output and display, image analysis and statistics, image enhancement and restoration, and linear filtering and transforms.

MATLAB has two different methods for executing commands: interactive mode and batch mode. In interactive mode, commands are typed (or cut-and-pasted) into the 'command window'. In batch mode, a series of commands are saved in a text file (either using MATLAB's built-in editor, or another text editor such as Notepad) with a '.m'

extension. The batch commands in a file are then executed by typing the name of the file at the MATLAB command prompt. The MATLAB built-in editor also supports a debugging tool, where commands can be executed step by step.

Decision

MATLAB's image processing toolbox was chosen as the library used in this project. Compared to Intel's IPPI, MATLAB's image processing toolbox has several advantages. First of all, it is easy to use. While working in the interactive mode, functions can be tested by just typing in the command, thus saving lots of programming time. Second, checking the values of elements in the matrix in the MATLAB environment is easy, which means the values of the pixels in the image can be viewed without debugging tools. Third, there are lots of predefined functions in the image processing toolbox, which are ready to use. Fourth, MATLAB is also a programming language. It provides several flow control constructs, such as *for* loops and *if* statements. The developed algorithms can be tested by writing a MATLAB script and can be executed in the MATLAB batch mode. The debugging tool helps in refining the algorithms.

4.2.2 Programming-Language Selection

Two programming languages are used in AnalySports Ltd: Delphi and C++. Therefore the selection of the development environment for this project focused on Borland Delphi (2001) and Microsoft Visual C++ (1998). The most important feature considered in choosing the development environment was the ease of use with the MATLAB library.

- **Delphi**

Borland Delphi is a powerful visual programming environment, suitable for beginners and professional programmers alike. Delphi's roots lie in Borland's Turbo Pascal, introduced in the mid-1980s. Object Pascal, the object-oriented extensions to Pascal, is the underlying language of Delphi. The Visual Component Library, or VCL, is a hierarchy of Object Pascal objects that allows designing applications. A better way of describing Delphi is as an Object Pascal-based visual development environment (Gajic, n.d.).

Delphi can be used to easily create self-contained, user friendly, highly efficient Windows applications in a very short time, with a minimum of manual coding. One of the major strengths of the Borland compilers is the support for database applications, based on Borland's Database Engine (BDE), which allows uniform use of databases held locally

in a variety of formats (most usefully as Paradox databases), and also held on other computers, and accessed using Standard Query Language (SQL) techniques (Programming with Borland Delphi, n.d.).

Delphi provides tools to develop, test and deploy Windows applications, including a large number of so-called reusable components. As well as the components in the VCL, there are many components, which can be obtained to provide more advanced functions. Some are sold through normal commercial outlets, and others are available as freeware or shareware files on web sites (Jacobson, 2002).

Calling MATLAB from Delphi

Component Object Model, or COM, is a set of object-oriented technologies and tools that allow software developers to integrate application-specific components from different vendors into their own application solution (MATLAB, 2002). COM support in MATLAB enables the programmer to interact with 'contained controls' (that is, running the control in the MATLAB process address space) or server processes, or to configure MATLAB as a computational server controlled by the client application programs.

Delphi provides a large number of components, language features and classes for writing COM programs, which can be used to call MATLAB functions and communicate with a MATLAB server. The Delphi approach to COM is thoroughly component oriented, which means calling COM server can be as easy as dragging-and-dropping a component to the user's form. Figure 4-1 shows the structure of a Delphi application communicating with a MATLAB Automation Server.

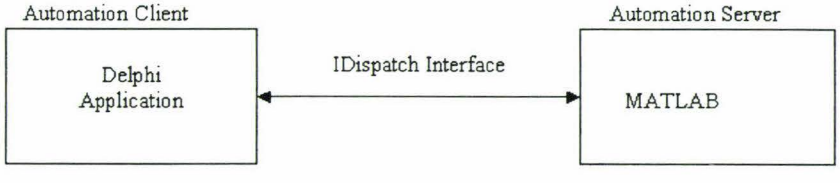


Figure 4-1 Calling MATLAB from Delphi

MATLAB operates as the Automation server in this configuration. It can be launched and controlled by Delphi application that is an Automation Controller.

- **Microsoft Visual C++**

The C++ language was developed by Bjarne Stroustrup of AT&T Bell Laboratories in the early 1980's, and is based on the C language (Pooley, n.d.). Microsoft Visual C++ is part of Microsoft's software development suite: Visual Studio. It is an Object-Oriented C++ compiler from Microsoft and is used as a tool for programming in an integrated development environment (IDE). Being the most common C++ development environment, Visual C++ has become the industry standard for C++ development on Windows and for Windows.

MATLAB's VC++ Add-in

The MATLAB compiler takes MATLAB M-files as input and generates C or C++ source code as output. There are several kinds of source code such as C++ stand-alone applications, C shared libraries (DLLs), as well as COM objects the MATLAB compiler can generate. Stand-alone MATLAB applications take advantage of the mathematical functions of MATLAB, yet do not require that the user owns MATLAB. Stand-alone applications are a convenient way to package the power of MATLAB and to distribute a customized application to the users. Using the compiler, algorithms developed in MATLAB performing specialized calculations can be converted to C shared libraries (DLL on Windows) or C++ static libraries. The algorithms then can be integrated into a C/C++ application. After the C/C++ application has been compiled, the MATLAB algorithms can be used to perform specialized calculations from the C/C++ program.

MathWorks provides a MATLAB add-in for the Visual Studio development environment that makes it easy to use the MATLAB compiler within Microsoft Visual C/C++ (MSVC) integrated development environment (IDE). The MATLAB add-in for Visual Studio greatly simplifies using M-files in the MSVC environment. The add-in automates the integration of M-files into Visual C++ projects. It is fully integrated with the MSVC environment so that the user can create stand-alone MATLAB applications or libraries in a similar way of creating a normal C++ project. Figure 4-2 shows the project wizard for the MATLAB add-in in the Visual C++ IDE.

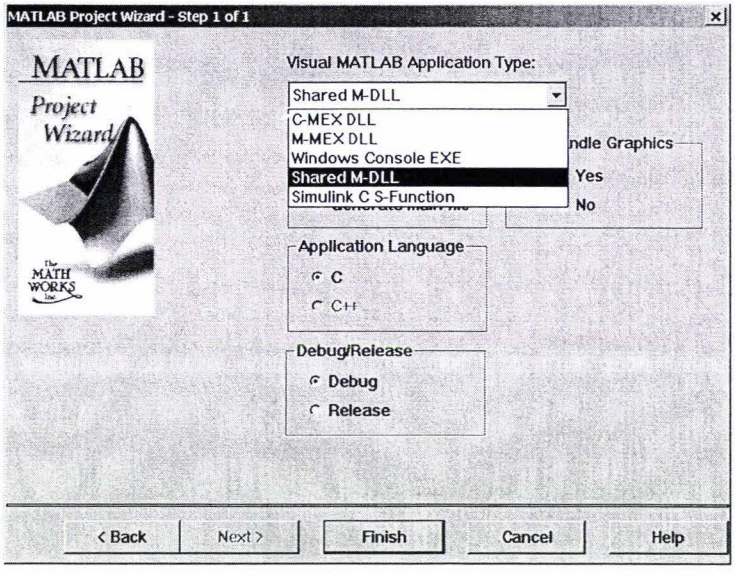


Figure 4-2 MATLAB Project Wizard

By choosing the 'Windows Console EXE' in the project wizard, a stand-alone application can be developed in Visual C++ using an M-file. If the user chooses 'Shared M-DLL', the M-file is converted into a DLL file, which can be integrated into other C/C++ application.

Decision

Microsoft Visual C++ was selected as development environment for this project. Compared to Delphi, Visual C++ provides a more suitable interface to MATLAB. By choosing VC++, the algorithms can be developed and tested in MATLAB without worrying about developing the interface. When the algorithms are ready, the MATLAB compiler can be used to convert the algorithms into DLLs, which are then integrated into the C++ application. Further more, the MATLAB VC++ add-in makes it simple to create the DLLs and add the DLLs to the VC++ application.

4.2.3 Database Selection

A Microsoft Access (2000) database was selected and is currently used as the database system for this project. Compared to other database systems, MS Access has several advantages:

- MS Access is a single-file database system, which means moving or copying the database can be easily achieved. This also helps when deploying the developed application.

- MS Access is commonly available in PCs. Being part of the Microsoft Office suite, Microsoft Access is installed in many PCs. Further more, this saves time and work for setting up the database system.

- In the early stages of the development, the design of the application and the database often changes. This requires a convenient interface for the database. MS Access allows the user to alter the structure of the database or access the data in the database in a windows GUI.

However, in the later stages of the development, MS Access database may become inappropriate for this project for two reasons: firstly, Microsoft Access requires a licence to install the database system; secondly, MS Access lacks native multithread support, which will be essential when multiple PCs are running the application (to share processing and therefore save processing time) and access the same database. The MS Access database system could then be changed to other database systems, such as MySQL.

4.3 Design and Implementation Issues

This section covers all the details of design and implementation for the application. The issues in designing and implementing the requirements introduced in Section 3.3 are explained.

4.3.1 The Structure of the Application

As shown in Figure 4-3, in developing the application, seven projects are defined and placed in one workspace in Visual C++.

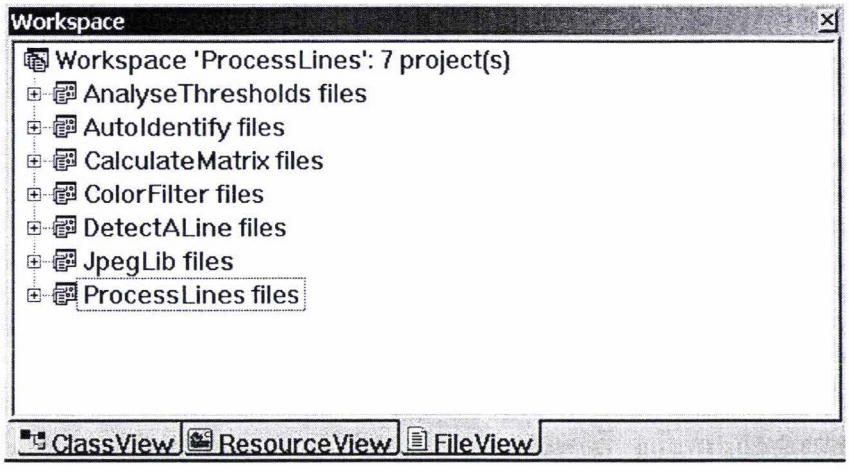


Figure 4-3 Workspace of the Application

'ProcessLines' is the main project, which contains the interfaces and most functions of the application. 'JpegLib' is a project containing the library for displaying, reading, and writing JPG files. The other five projects are created using MATLAB's add-in for VC++. They are used to generate libraries (DLLs) containing algorithms developed in MATLAB.

The main project

The main project in the workspace (*ProcessLines*) is created using the application wizard in VC++. It is a Single Document Interface (SDI) application, which means there are several classes created automatically by VC++. Three of the classes are most important in the project:

- The CMainFrame class is the window frame. It holds the menu, tool bar, and the status bar. A specific segment of code has been developed in handling the resize event of this class, so as to prevent the window being too small.

- The CProcessLinesDoc class is where the data structures are held and manipulated. The CProcessLinesDoc class responds to inputs from the CMainFrame class to perform certain tasks, such as start/pause/stop the processing, playback the image sequence, and pop up the statistics dialog. It also passes the display information to the CProcessLinesView class.

- The CProcessLinesView class displays the visual representation of the CProcessLinesDoc class. The CProcessLinesView class is derived from the CFormView class. A slider bar is placed in the form to show the progress of the

processing and to control the displayed frame in case the playback mode is selected.

Other classes in the project include the CProcessLinesApp class, which creates all the other components in the project and passes all the events to relevant classes, the dialog classes used as interfaces, such as the CColorSettingDlg class, and third-party libraries, such as the JPEG library used to read and write JPG files. The main interface of the application is shown in Figure 4-4.

In the interface, the menu and the tool bar allow the user to control the running of the application. These controls include the open/save button used to specify the source and destination, the play/pause/stop buttons, used to control the progress of the processing, the settings button, which is used to pop up the change settings dialog, the playback button, which is designed to toggle between playback and normal mode, the statistics button, which is used to pop up the statistics dialog, the database button, which allows the user to change the database, and the help buttons. In the main part of the interface, the input and output images are displayed. A slider bar is placed under the images, and indicates the progress of the task during processing. In playback mode, the slider bar can be used to select a frame from the video clip for display. The status bar in the interface displays information about the application, including the current status of the application, the source image being processed, the number of processed frames, the total number of frames in this task, and the time consumption.

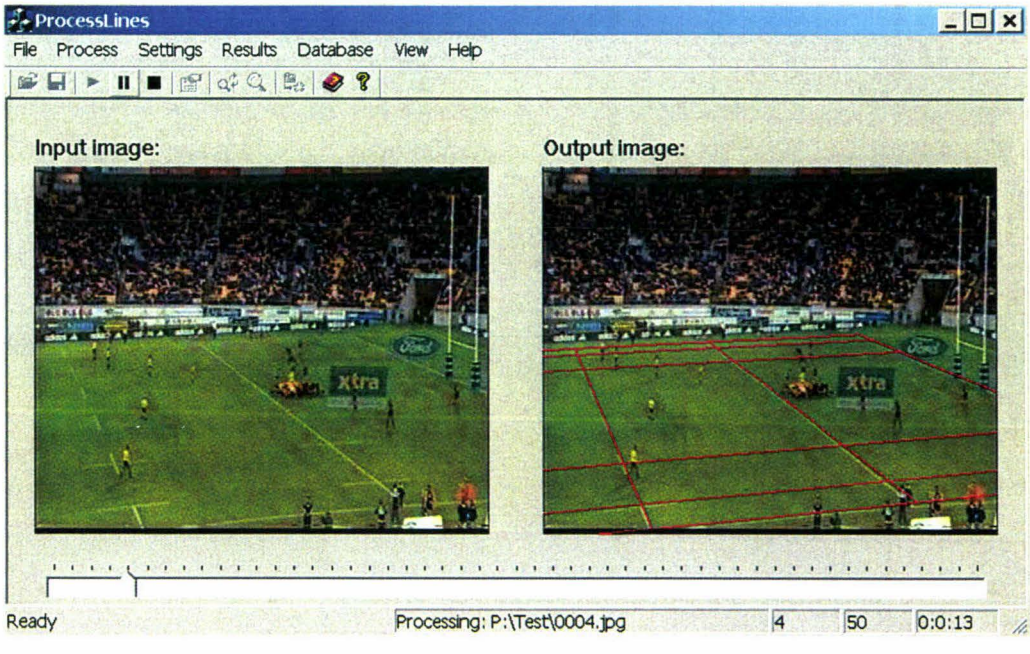


Figure 4-4 Main Interface of the Application

4.3.2 Design Decisions and Implementation Issues

In this section, the design decisions and implementation issues are explained addressing the requirements described in Section 3.3.

Requirement 1. Implementation of the pre-processing module.

Design Decision: The video clips are converted into sequences of JPG-format image files using video or image processing tools.

Implementation Issues: The pre-processing module of the proposed system can be implemented independently, that is, outside the application developed in VC++. This is because the final player tracking system will eventually use video clips as input, thus a function will be introduced to the application to read the MPEG file and extract frames from it. This function will then substitute the pre-processing module.

An application called 'VirtualDub' (Lee, 2003) is used to extract the video frames from the video clip. VirtualDub is a video capture/processing utility for 32-bit Windows platforms, licensed under the GNU General Public License (GPL). It is streamlined for fast linear operations over video. VirtualDub is mainly geared toward processing AVI files, although it can read MPEG-1 and also handle sets of bitmap (BMP) images.

The video clip in this project is read into VirtualDub, and then saved as image sequence in BMP format. Figure 4-5 shows the interface for the conversion.

The BMP-format image sequence is then converted to JPG-format image sequence to save disk space. There are lots of tools that can be used to perform this task. In this project, the MATLAB image-processing toolbox has been used for this task. A small M-file has been developed in MATLAB and converted to a stand-alone application using MATLAB's compiler. Figure 4-6 shows the interface of the application.

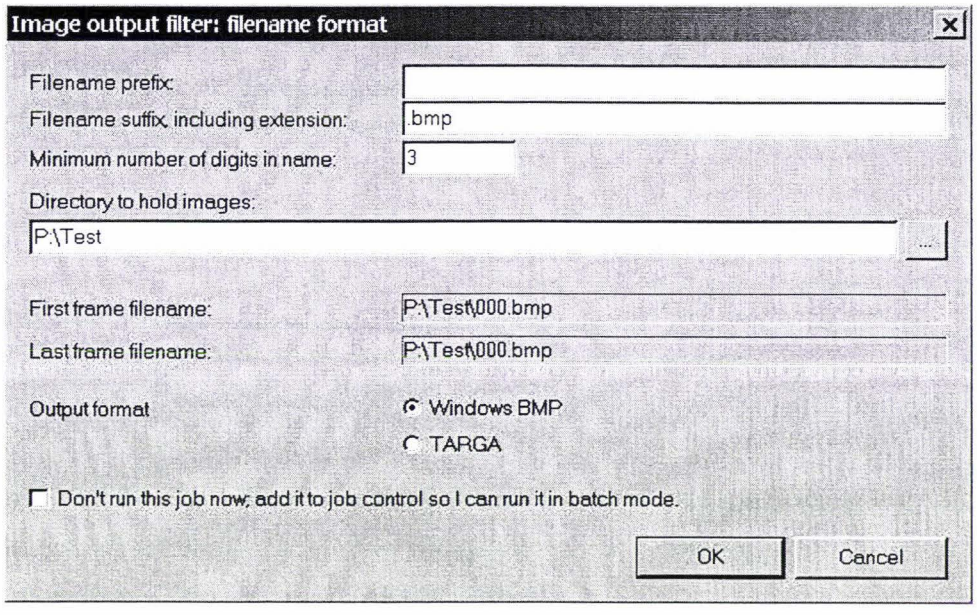


Figure 4-5 Convert MPEG to Image Sequence

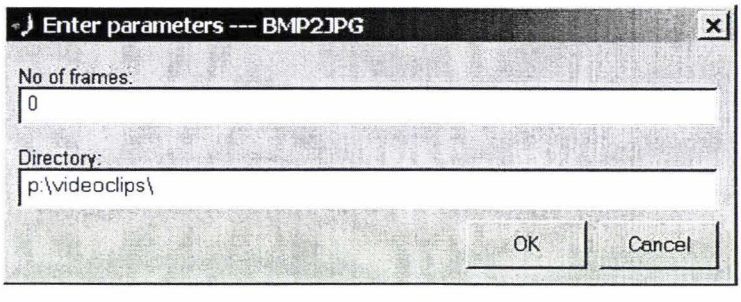


Figure 4-6 Convert BMP Sequence to JPG Sequence

Requirement 2. The application needs to be able to handle JPG files.

Design Decision: The application is able to read, display and write JPG format files.

Implementation Issues: The basic JPEG standard defines many options and alternatives for the coding of still images of photographic quality. Its file format was originally created by Eric Hamilton in 1992 and called the Jpeg File Interchange Format (JFIF) (Hamilton, 1992). When Eric Hamilton described the JFIF, he did not define the exact details of the implementation. The most commonly used implementation for encoding and compression JPEG images is that produced by Independent JPEG Group or IJG. Other implementations are the Pegasus Jpeg Library, and the version used in Photoshop.

In this project, the implementation produced by the IJG is used. IJG is an informal group that writes and distributes a widely used free library for JPEG image compression. Chris Losinger (1997) released a free class called JpegFile, which is based on VC++ 6.0 project of the IJG's V.6a JPEG source. The JpegFile class is a wrapper around the Independent JPEG Group library. It provides simple functions for reading and writing JPEG files. Thanks to Chris Losinger's work, the functions used in this project for accessing the JPEG images are as simple as

```
inFrame=JpegFile::JpegFileToRGB(strFileName,&uiW,&uiH);
```

Requirement 3. An interface needs to be provided for a human operator to define the source and destination of the processing.

Design Decision: In order to specify the source and destination of the processing, a file-open dialog is used to allow the user to choose the start frame of the task. A folder-selection dialog is used to allow the user to select a folder for storing the output images.

Implementation Issues: The file-open dialog uses the standard 'CFileDialog' class in MFC. The folder-selection dialog uses a third-party class, called 'CFolderDialog' (Hakobyan, 2002), which allows the user using Win2000 to create a new folder in the interface. Figure 4-7 shows the examples of the file-open dialog and the folder-selection dialog.

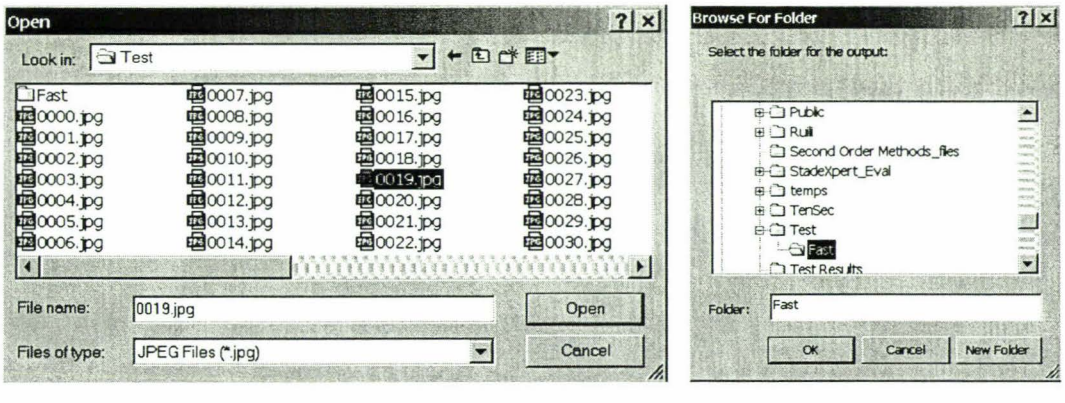


Figure 4-7 The File-open and Folder-selection Dialogs

After the starting frame is specified, the program read the file names of all the consecutive frames and calculates the total number of frames. When the start button in the main interface of the application is clicked, the application starts processing the frames

until the user stops it or the end of image sequence is reached. Below is the pseudo-code for the processing.

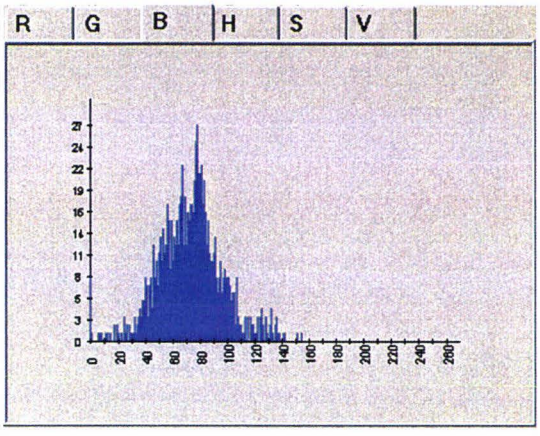
```
Select_First_Frame;
Select_Destination_Folder;
Success = FALSE;
While NOT(EndOfSequence OR UserStop)
BEGIN
      While NOT(EndOfSequence OR UserStop OR Success)
            Success = Manually_Identify_One_Frame;
      While NOT(EndOfSequence OR UserStop) AND Success
            Success = Auto_Tracking_One_Frame;
END
```

Requirement 4. An interface must be provided to help the user finding and setting the thresholds in image segmentation.

Design Decision: Figure 4-8 shows an example of the interface. In Figure 4-8 (a), the human operator can mark lines in the video frame using the drawing tool provided. Right-clicking the marked line will pop up a context menu, which can be used to delete unwanted lines. The histograms of the marked lines in both RGB and HSV color space can be drawn as shown in Figure 4-8 (b). In Figure 4-8 (c), text boxes are used to accept the thresholds. After applying multi-dimensional thresholding, the result (a binary image) is shown to the human operator, as demonstrated in Figure 4-8 (d).

Implementation Issues: The interface used to set the thresholds for multi-dimensional thresholding is shown in Figure 4-9, which combines the functions defined in Figure 4-8.

(a) Lines marked by the user on the video frame, user can delete a line by right-clicking on it



(b) Histograms calculated, clicking the tabs will change the color space

	Min	Max		Min	Max
R:	120	170	H:	0.18	0.25
G:	150	200	S:	0	1
B:	80	130	V:	0	1

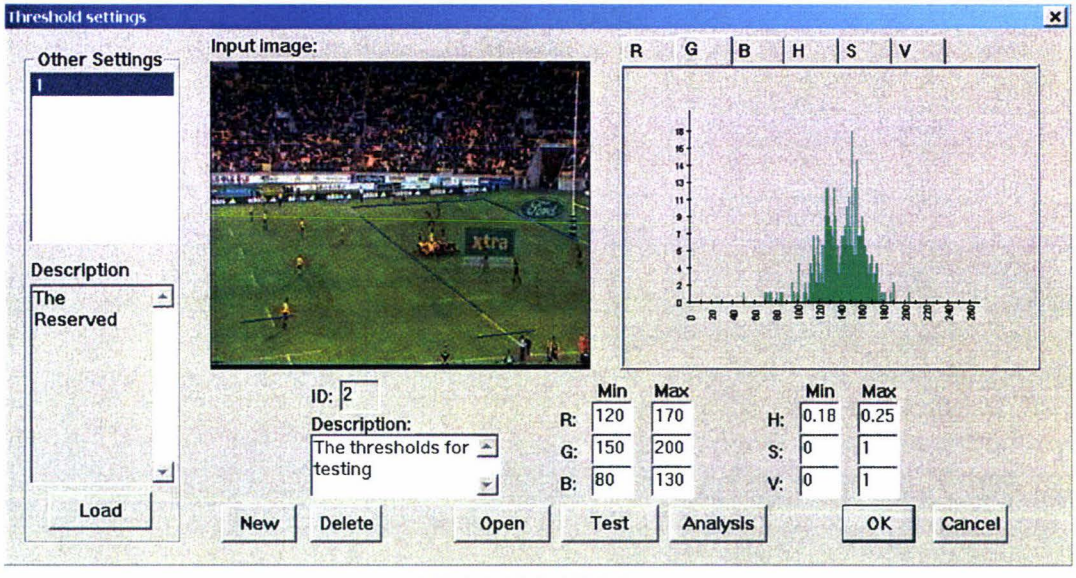
(c) The thresholds determined by the user

(d) Test outcome generated

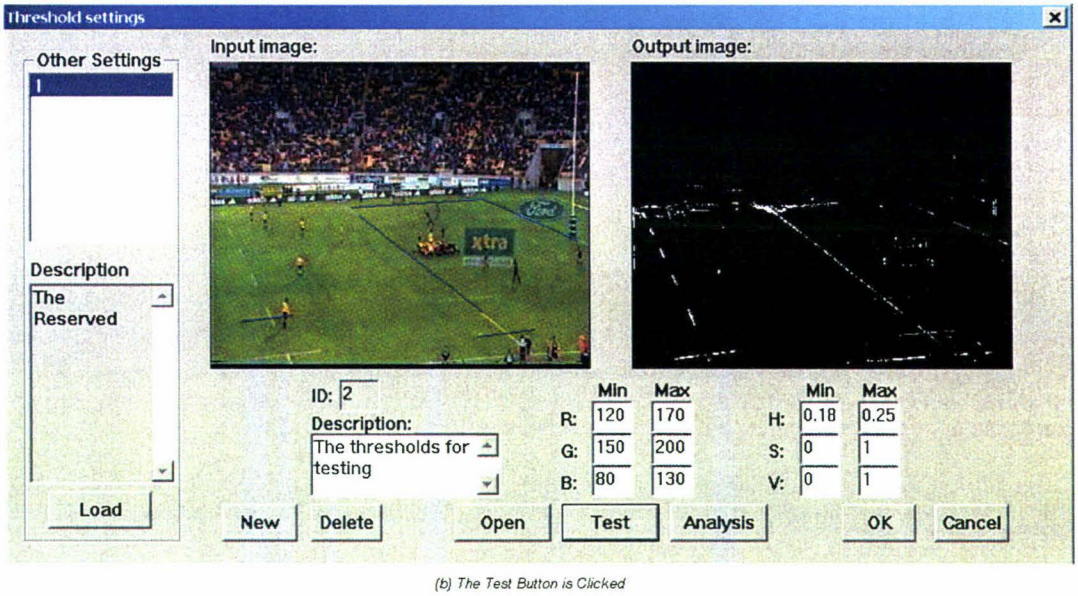
Figure 4-8 Finding the Thresholds

As shown in Figure 4-9, in the left hand side of the interface, a list box is used to show the IDs of other sets of thresholds in the database (database structures are explained in Requirement 8), the edit box below it shows the description of the highlighted set of thresholds. By clicking the 'Load' button, the highlighted set of thresholds can be loaded. In the right hand side of the interface, the current set of thresholds is displayed and can be modified. The 'New' and 'Delete' buttons will create or delete a set of thresholds respectively. The 'Open' button allows the user to select an image file to be used in testing the specified thresholds. The 'Test' button calls the multi-dimensional thresholding function developed in MATLAB to perform image segmentation using the thresholds defined in the interface. The result is then displayed in the top-right corner of the interface. The 'Analysis' button is used to calculate the histogram of the marked pixels in the input image. To make the interface tidy, the output of the histogram function is also displayed in the top-right corner of the interface. Clicking the 'Test' and 'Analysis' button

will allow the display to toggle between the histogram and the output of the multi-dimensional thresholding.



(a) The Analysis Button is Clicked



(b) The Test Button is Clicked

Figure 4-9 The Interface for Selecting Thresholds

The function to calculate the histogram is also implemented in MATLAB. The process of calling functions defined in MATLAB is explained below using the histogram function as an example.

- Assume that the function definition in MATLAB is:

```
function [R, G, B, H, S, V] = analyColor(I, P);
```

where '*function*' is the keyword indicating the start of the function definition line, which is followed by output parameters *R*, *G*, *B*, *H*, *S* and *V*. '*analyColor*' is the function name, which is followed by the input parameters *I* and *P*. The variable *I* holds the input image, and the variable *P* is an *N by 4* matrix. *N* denotes the number of lines marked in the image, and 4 represents the coordinates of the vertices of the lines. The output parameters are vectors containing the results of the calculation in different color spaces. For example, the histogram of the Red component is stored in a vector containing 256 (the value of the Red component is between 0 and 255) elements.

- A project called '*AnalyseThresholds*' is developed using the MATLAB add-in for VC++. This project is created from the m-file containing the *analyColor* function. This function is put in the same workspace as the main project (*ProcessLines*). The output directory of the '*AnalyseThresholds*' project is set to where the executable file of the '*ProcessLines*' is, so that the DLL can be accessed by the executable file.

- By including the header file of the '*AnalyseThresholds*' project, the MATLAB function can be called from the Visual C++ application. Below is the example of code to call the function.

```
/*include the header file of the MATLAB project*/
#include "AnalyseThresholds\AnalyseThresholds.h"
...
...
      double *pr = NULL; // declare pointer to mxArray
      /* Declare matrices and initialize to NULL */
      mxArray *I = NULL;
...
      mxArray *V = NULL;

      /* Create the matrices and assign data to them */
      mlfAssign(&I, mxCreateNumericArray( ndim,
            dims,
            mxDOUBLE_CLASS,
            mxREAL));
```

```
      pr = mxGetPr(I);
      memcpy(pr, pDb, bytes_to_copy);
      delete [] pDb;

      /*prepare the input data*/
      double *lines = new double[4*liCount];
      for (int i=0;i<liCount;i++)
      {
            lines[i] = ((CLine *) m_oaLines[i])->m_x1;
      …
      }
      mlfAssign(&P, mlfDoubleMatrix(liCount, 4, lines,
NULL));
      delete [] lines;

      /*Calling the function developed in MATLAB*/
      R = mlfAnalycolor(&G, &B, &H, &S, &V, I, P);

      /*Copy the values back to VC++*/
      pdR = (double *) new double[257];
      memcpy(pdR, mxGetPr(R), 257*sizeof(double));
      …
      pdV = (double*) new double[102];
      memcpy(pdV, mxGetPr(V), 102*sizeof(double));

      /*Release the variable*/
      mxDestroyArray(R);
      …
      mxDestroyArray(P);
```

Requirement 5. The application must provide an interface to allow changing settings or selecting an existing set of settings from the list.

Design Decision: Figure 4-10 shows the interface (annotated) designed in the application to change or load settings. Also from this interface, the user will be able to access the interfaces to change thresholds and colors as described in Requirement 4 and Requirement 6.

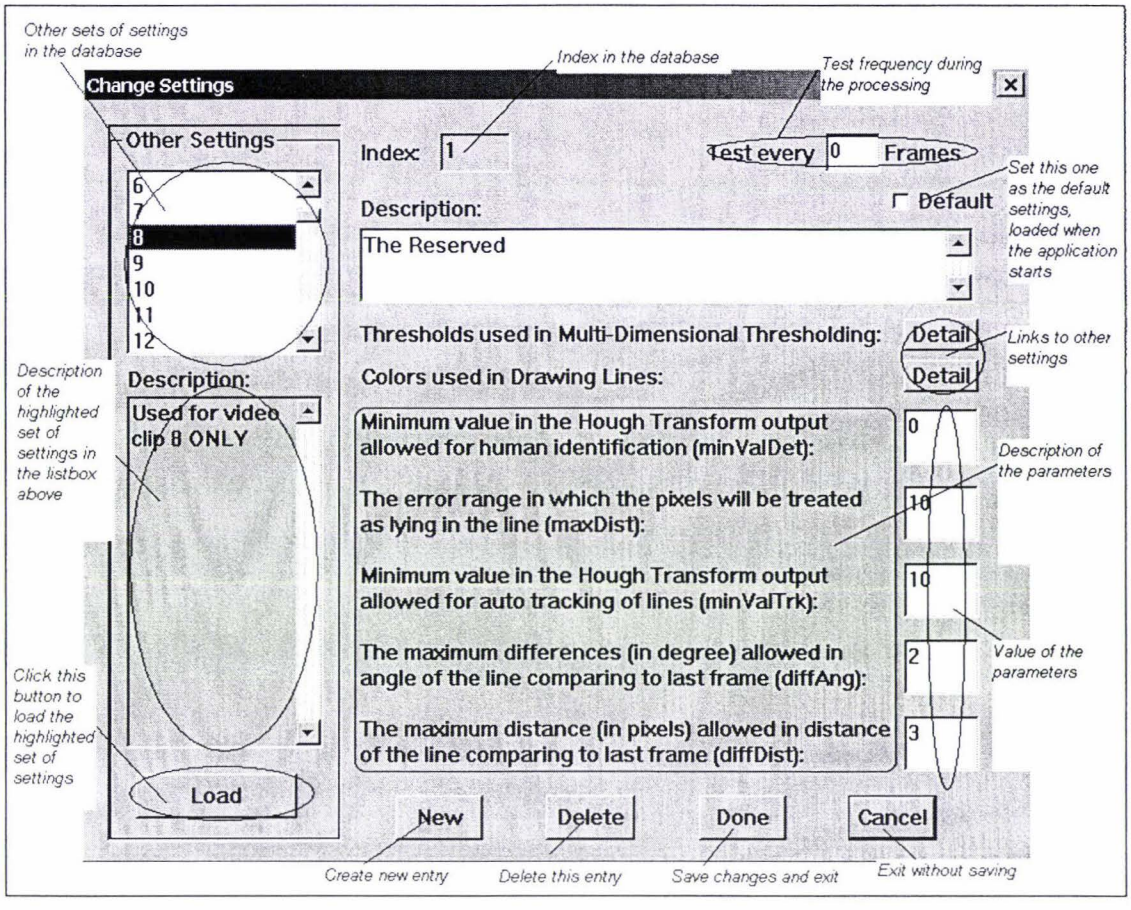


Figure 4-10 Annotated Interface of Changing Settings

The change settings interface consists of two parts. On the left hand side of the interface, other existing settings in the database (in the Settings table, descriptions of tables in the database can be found in Requirement 8) are listed and the edit box is used to display the description of the highlighted set of settings in the list box. The 'Load' button allows the highlighted set of settings to be loaded and used in current task.

Implementation Issues: According to the design of the change settings interface (Figure 4-10), a form in the main project (*ProcessLines*) has been developed. Clicking the 'settings' button in the tool bar of the main interface can pop up this dialog. When the application is loaded, the default settings are loaded. If there is no default setting defined in the Settings table of the database (database structures are described in Requirement 8), the settings with the ID equals to 1 is loaded. A restriction has been set up in the code to prevent deleting the first entry (ID=1).

Requirement 6. An interface must be provided to change the colors associated with different roles of the lines in the output image.

Design Decision: Figure 4-11 shows the design of the color settings interface. This interface contains input fields to define the colors and a test image to demonstrate the outcome.

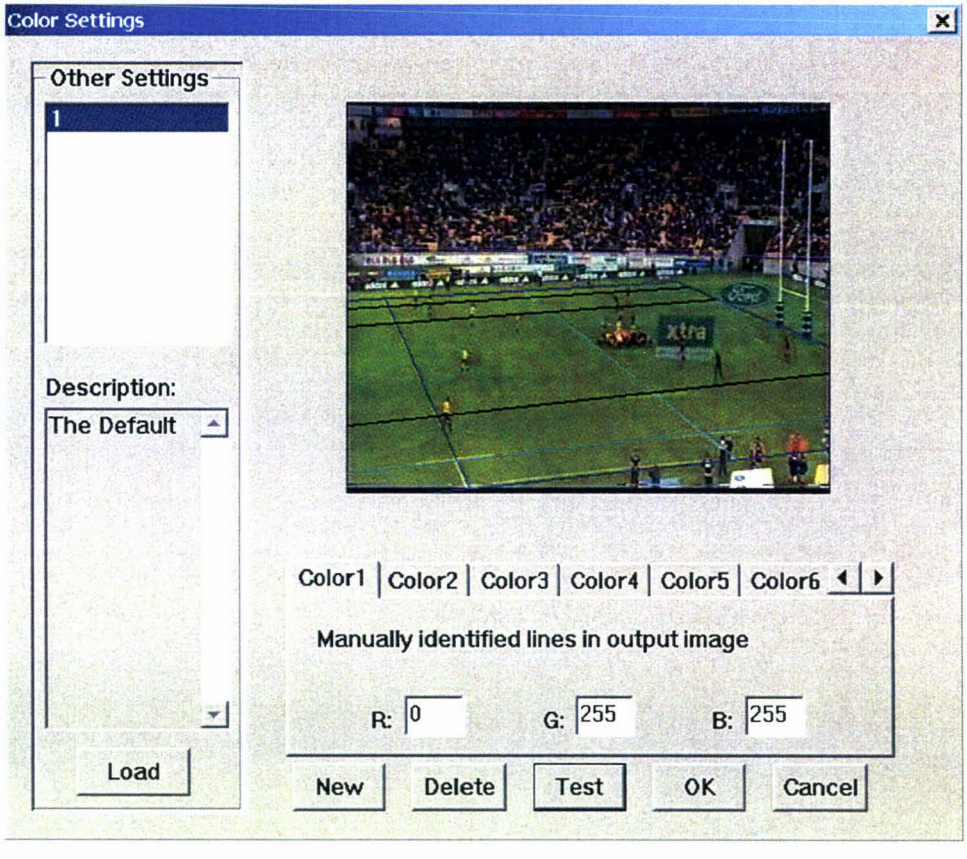


Figure 4-11 Color Settings for Different Roles of Lines

The color settings interface has a similar structure to that of the interface for setting thresholds. At the left hand side of the interface is a facility for loading predefined settings. On the right hand side, a tab control is used to allow the user to change the color setting, and the 'Test' button will display the lines on the image using the settings defined in the tab control.

Implementation Issues: The color-setting dialog has been set up according to the design of its interface (Figure 4-11). A static image is used to demonstrate the effect of the color scheme. The positions of the lines superimposed to the image are fixed; the color used to draw the lines will change according to the value in the tab control. Similar to the first

entry in the Settings table, the first entry in the Colors table is protected in the code as the reserved default color scheme.

Requirement 7. Functions and interfaces need to be developed for line detection and tracking.

Design Decision: An interface will pop up when the human operator is required to name the detected line manually. Figure 4-12 shows the design of the interface (annotated) to present a detected line to the human operator. The human operator is also provided with a drawing tool so that when the detected line is not accurate, modification can be done. The check box in the interface allows the user to choose which line to be used, the detected or the one manually drawn.

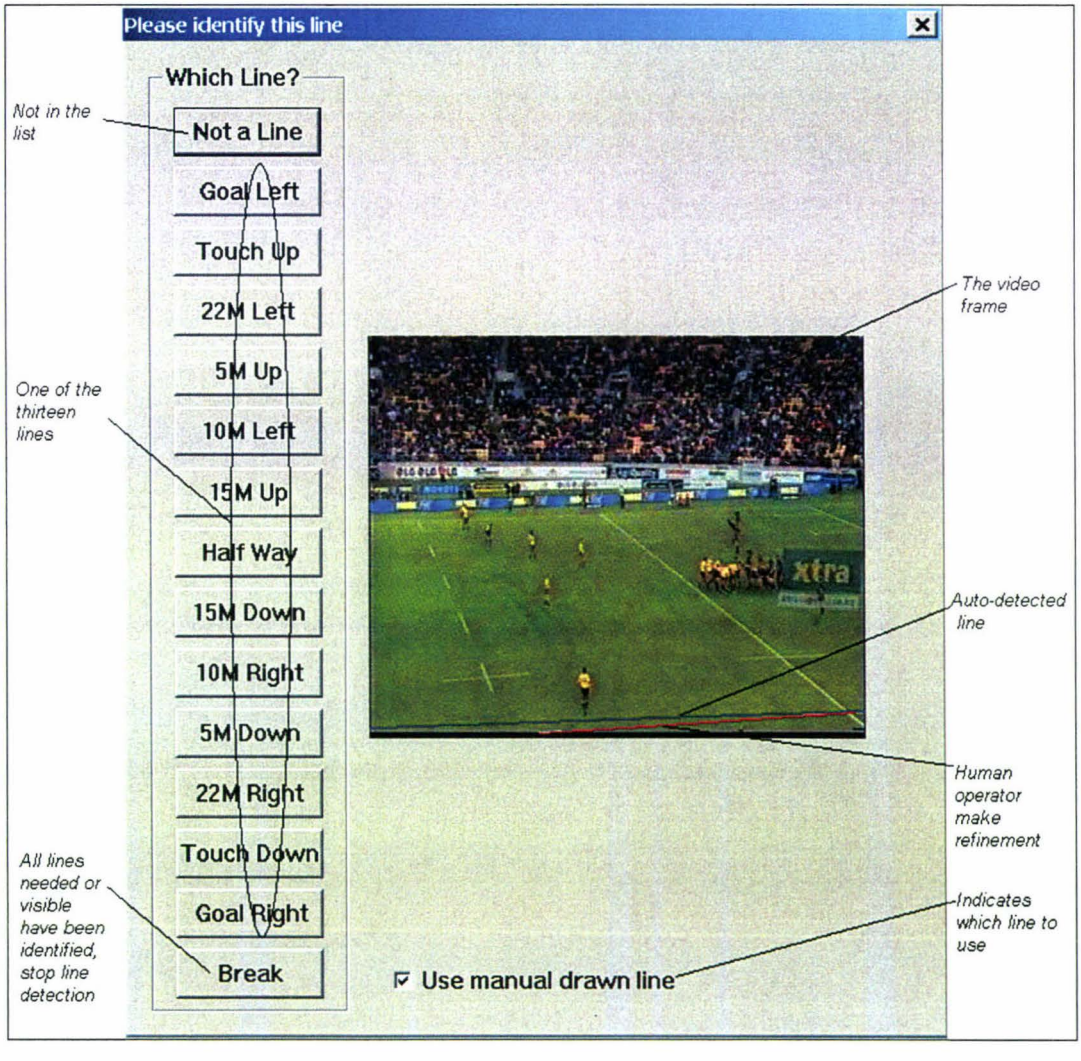


Figure 4-12 Annotated Interface of Manually Identify a Line

Implementation Issues: Functions for processing the frames are implemented in both MATLAB and Visual C++. Using the process of manually identifying a video frame as an example, the implementation of the frame-processing functions can be explained as:

- The source image is read to the memory, using the technique described in Requirement 2.

- Multi-dimensional thresholding is applied to the input image to get a binary image. This is done using a function developed in MATLAB. This function is in the project *ColorFilter* in the VC++ workspace. The syntax of the function is:

```
function BW = multithreshold(I, T)
```

  where the *I* is the input image, the *T* is the thresholds, and the *BW* holds the output binary image.

- Hough transform is performed to get the matrix in Hough space. This is done in VC++ based on the *radon* function provided by the MATLAB in the file *radonc.c* under the image toolbox folder.

- The global maximum is found using a function developed in MATLAB. The function is located in the project *DetectALine*. The syntax of the function is:

```
function  [J, bwJ, Finish, Angle, Dist] =
          GetALine(I, bwI,  T, xpl, minValDet,
          maxDist, colorR, colorG, colorB)
```

  where the *I* is the input image, the *bwI* is the binary image, the *T* and *xpl* are output of the Hough transform, the *minValDet* and *maxDist* is parameters to detect the line, the *colorR*, *colorG*, *colorB* define the color components used to draw the detected line in the image. *J* is the image with the detected line superimposed upon it. *bwJ* is the binary image where the pixels lying in the detected line have been turned off. *Finish* denotes whether a line has been detected. *Angle* and *Dist* are the parameters defining the detected line.

- In order to identify the detected line, a dialog has been developed in the main project of the application to realize the interface designed in Figure 4-12. The reference lines then are selected from the identified lines. This is also implemented in VC++.

- A function developed in MATLAB is used to calculate the perspective transform matrix. This function is in the project *CalculateMatrix*. The syntax of the function is:

```
function [J, lineLogJ, M, Failed] =
                getMatrix(I, lineLogI, left, right,
                up,  down, colors);
```

  where *I* is the input image, *lineLogI* contains the information of known lines, *left, right, up* and *down* are indices of the reference lines, *colors* defines the colors used to draw the lines in the image according to their roles. *J* is the image with the visible lines superimposed upon it, *lineLogJ* contains information on all the lines in this frame, *M* is the perspective transform matrix, *Failed* denotes whether the calculation was successful.

- The database functions (described in the Requirement 8) helps to save the result data to the database.

Using the functions listed above, a video frame can be processed by manually identifying the lines. Most of the functions are also used in processing of video frames by auto tracking of lines. An extra function is developed in this case to choose between the four-lines algorithm and the three-lines algorithm. This is a MATLAB function and is contained in the project *AutoIdentify*.

Requirement 8. A database must be designed and used to store the information produced by the line recognition and tracking process.

Design Decision: A database has been designed to store the information relating to the line recognition and tracking process. According to the algorithms described in Section3.1 and Section 3.2, seven tables have been designed.

➢ **VideoFiles Table**

The VideoFiles table is used to store general information about the video clips. One record in this table corresponds to one processing task. Table 4-1 shows the design of the table.

Field Name	*Data Type*	*Description*
ID	AutoNumber	Primary key, one for each process task
FileName	Text	The path of the source image sequence
StartTime	Date/Time	The first time the task has been started
TimeUsed	Number	Time used in seconds
StartFrame	Number	Number of the first frame
Processed	Number	How many frames have been processed
OutputName	Text	Path of the output sequence
FileDigits	Number	Number of digits in the file name

Table 4-1 Design of VideoFiles Table

Note that the 'StartTime' only records the first time the task is started. As one task may be stopped and resumed later, this value is not a suitable measurement for time consumption. Instead, the 'TimeUsed' field is an accumulate field, that means if the task is resumed, the time spent is added to the original time. The 'Processed' field is also updated in case of a task been resumed.

➢ **Thresholds Table**

The Thresholds table contains the thresholds used in the multi-dimensional thresholding. Table 4-2 shows the design of the Thresholds table.

In the table, the range of thresholds in the RGB color space is 0 ~ 255. The range of thresholds in HSV color space is 0 ~ 1. Storing the thresholds in a stand-alone table simplifies the structure of the Settings table, and at the same time makes it possible to reuse the settings of thresholds.

Field Name	*Data Type*	*Description*
ID	AutoNumber	Primary key, one for each set of thresholds
Description	Text	Description of this set of thresholds, help reusing the thresholds
LoR	Number	12 parameters specifying the thresholds
HiR	Number	
LoG	Number	
...	...	
LoV	Number	
HiV	Number	

Table 4-2 Design of Thresholds Table

- **Colors Table**

The Colors table specifies the colors used in the output image to represent different roles of the lines superimposed to the video frame. The default value is set as described in Table 3-3. Similar to the Thresholds table, using an independent table to store the Colors information helps reusing the predefined color sets. Table 4-3 shows the design of the Colors table.

Field Name	*Data Type*	*Description*
ID	AutoNumber	Primary key, one for each set of colors
Description	Text	Description of this set of colors, help reusing the colors
R1	Number	15 parameters specifying five RGB colors
R2	Number	
R3	Number	
R4	Number	
R5	Number	
G1	Number	
...	...	
B5	Number	

Table 4-3 Design of Colors Table

- **Settings Table**

The Settings table contains all the parameters required in the processing. Table 7 shows the structure of the Settings table. The ThresholdsID and the ColorsID refer to entry in the Thresholds and Colors table respectively.

Field Name	*Data Type*	*Description*
ID	AutoNumber	Primary key, one for each set of settings
Description	Text	Description of this set of colors, help reusing the colors
ThresholdsID	Number	ID in the Thresholds table
ColorsID	Number	ID in the Colors table
minValDet	Number	As described in Section 3.2.2.2
maxDist	Number	As described in Section 3.2.2.2
minValTrk	Number	As described in Section 3.2.2.3
diffAng	Number	As described in Section 3.2.2.3
diffDist	Number	As described in Section 3.2.2.3
testFreq	Number	The frequency of testing, 0 means no testing during the processing
ColorR	Number	The color used to display the detected line to the human operator for identification
ColorG	Number	
ColorB	Number	
isDefault	Yes/No	Default setting is the one loaded when the application initializes; if no default setting found, the application loads settings 0 (predefined in the database)

Table 4-4 Design of Settings Table

- **ProcessResults Table**

The ProcessResults table is where the result of the processing is stored. Every processed frame has a corresponding record belonging to this table. Table 4-5 shows the structure of the ProcessResults table.

The ProcessResults table contains a VideoFileID field, which is used to identify the source of the input image. Because the settings can be changed during the processing of the video clip, each entry of the ProcessResults table (video frame) contains a SettingsID, associating one set of parameters with the video frame.

Field Name	*Data Type*	*Description*
ID	AutoNumber	Primary key, one for each frame
VideoFileID	Number	The process task (ID field in VideoFiles table)
SettingsID	Number	The index of settings used for this frame (ID field in Settings table)
FrameNo	Number	The frame no
NoOfLines	Number	How many lines used as reference, -1 means manually; 0 means failed
lineLog11	Number	A 13 x 3 matrix, used to store the information of lines 13 lines referred to in hexadecimal notation 1..D 3 parameters (Angle, Distance, and Role) referred to 1..3
lineLog12	Number	
lineLog13	Number	
lineLog21	Number	
...	...	
lineLogA1	Number	
lineLogA2	Number	
lineLogA3	Number	
...	...	
lineLogD3	Number	
matrix11	Number	A 3 x 3 matrix, used to store the perspective transform matrix of the video frame
matrix12	Number	
matrix13	Number	
matrix21	Number	
...	...	
matrix33	Number	

Table 4-5 Design of ProcessResults Table

- **TestResults Table**

The TestResults table contains frame-level information of the testing. Each tested frame has a corresponding entry in this table. Table 4-6 shows the structure of the TestResults table. The values in 'NoOfPoints', 'MaxError', 'MinError', and 'MeanError' can also be retrieved querying the TestPoints table, the reason they are duplicated in this table is to make it easier to gather the clip-level information of the testing.

Field Name	*Data Type*	*Description*
ID	AutoNumber	Primary key, one for each video frame
FrameID	Number	The ID field in ProcessResults table
NoOfPoints	Number	Number of tested points in the video frame
MaxError	Number	Max of the Deviations in the tested points
MinError	Number	Min of the Deviations in the tested points
MeanError	Number	Mean of the Deviations in the tested points (*DF*)
BadThresholds	Yes/No	Human operator observes the problem causing the deviation in the video frame
LensDistortion	Yes/No	
RefLinesTooClose	Yes/No	
Shadow	Yes/No	

Table 4-6 Design of TestResults Table

➢ **TestPoints Table**

The TestPoints table contains the pixel-level information of the testing. Table 4-7 shows the structure of the TestPoints table. Theoretically, the 'Calculated_x', 'Calculated_y', and the 'Deviation' fields could be derived by applying the transform matrix, which is stored in the ProcessResults table. The decision to explicitily state these values has been made because this will make the calculation of the frame-level testing information much faster.

Field Name	*Data Type*	*Description*
ID	AutoNumber	Primary key, one for each tested point
TestID	Number	The ID field in TestResults table
Pixel_x	Number	The coordinates of the marked pixel in the coordinate system of the video frame
Pixel_y	Number	
Calculated_x	Number	The coordinates of the marked pixel in the coordinate system of the rugby field
Calculated_y	Number	
Field_x	Number	The ideal coordinates of the marked pixel in the coordinate system of the rugby field
Field_y	Number	
Deviation	Number	The deviation calculated by measuring the distance between the Calculated and Field position

Table 4-7 Design of TestPoints Table

Relationships

According to the design of the tables in the database, the relationships between tables in the database can be summarised as in Figure 4-13.

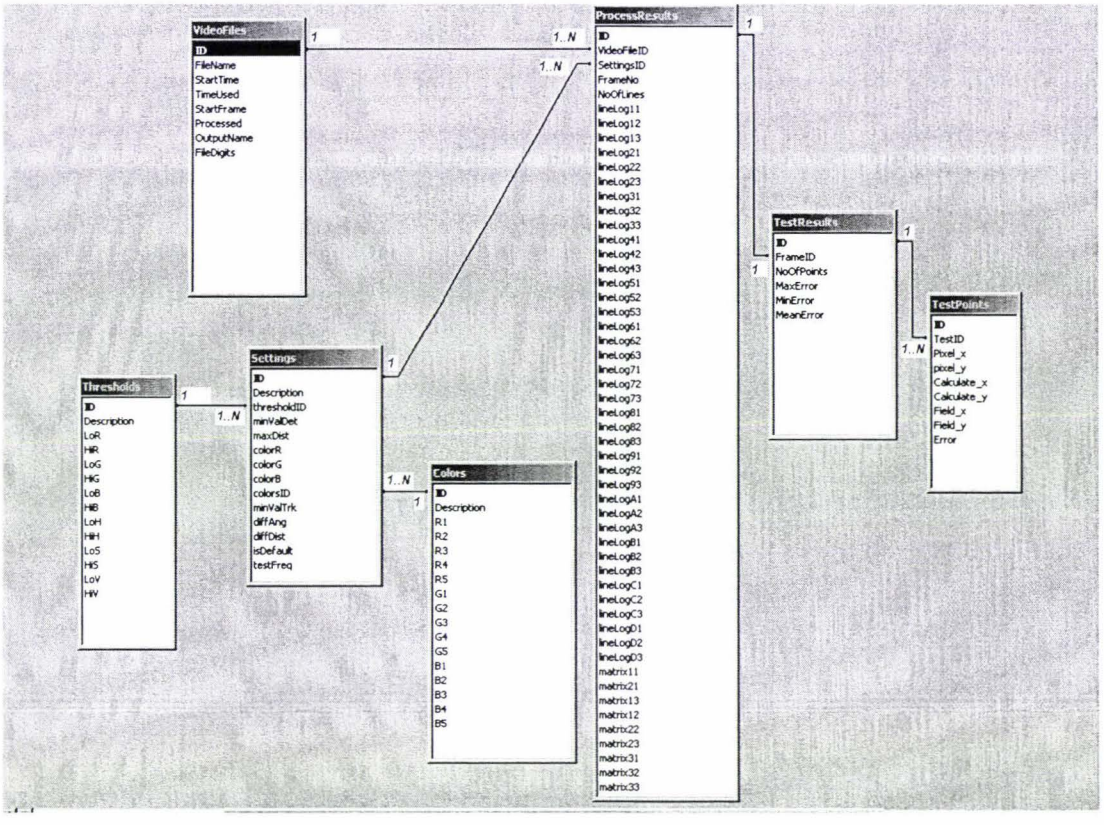


Figure 4-13 Relationships in Database

Implementation Issues: As MS Access database is currently used in the application. Changing a database simply means to select another database file because MS Access is a single-file database. The default database loaded by the application is the one in the working directory of the application --- data.mdb. To change to another database, a file open dialog is used. The new database is selected by the user and loaded by the application (this includes the change of default settings to the one in the new database).

To access the data in the database, ActiveX Data Objects (ADO) is used. ADO is the strategic application-programming interface (API) to data and information. ADO provides consistent, high-performance access to data and supports a variety of development needs, including the creation of front-end database clients and middle-tier business objects that use applications, tools, languages, or Internet browsers (Chand, 2000). It is the fastest method to access various kind of databases. A third-party library (Antollini, 2003) is used in this project to communicate with the database. The library

was released by Carlos Antollini in 2003 in order to make it easy to work with ADO. Two classes are used in this project: the CADODatabase class and the CADORecordset class. The CADODatabase class has a set of functions that corresponds to the _ConnectionPtr class in MFC, such as the Open function and the SetConnectionMode function. The CADORecordset class has a set of functions that corresponds to the _RecordsetPtr class, such as the AddNew function and the Delete function. An example of using the CADODatabase and CADORecordset class is demonstrated as below.

```
////database object
CADODatabase m_pDb;
////recordset object
CADORecordset m_pRs;
////connection string
strConnection = _T("Provider=Microsoft.Jet.OLEDB.4.0;
                                    Data Source=data.mdb");
////connect to database
if(m_pDb.Open(strConnection))
{
      m_pRs = CADORecordset(&m_pDb);
      CString strTmp;
      strTmp.Format("select * from VideoFiles where ID
                                    = %d;", m_iVideoFileID);
      ////run a query, open recordset
      if (m_pRs.Open(strTmp,
                        CADORecordset::openUnknown))
      {
            double dSec;
            ////get a value by field id
            m_pRs.GetFieldValue(3, dSec);
            if (dSec==0)
                  dSec = (COleDateTime::GetCurrentTime()
                        - m_oledtStart).GetTotalSeconds();
            else
                  dSec +=(COleDateTime::GetCurrentTime()
                        - m_oledtStart).GetTotalSeconds();
            ////allow editing the recordset
            m_pRs.Edit();
```

```
                ////change value in recordset
                m_pRs.SetFieldValue(3, dSec);
                ...
                ////update change to database
                m_pRs.Update();
        }
        ////close recordset and database
        m_pRs.Close();
        m_pDb.Close();
}
```

In the above example, the variable strConnection is used to denote the connection string to be used when opening a database. If in the future development of the application, another database system is going to replace the Microsoft Access database, the only change needed is a different connection string. For example, to connect to a remote MySQL database system, the connection string would be

```
strConnection = _T("Provider=MySQLProv; server =
                                remotehost; DB = test");
```

More information on the connection string can be found on Antollini (2002).

Requirement 9. The quality of the line recognition and tracking needs to be measured.

Design Decision: An interface is required in order to measure the quality of the line recognition and tracking. In this interface, the user can click on the input video frame to mark the pixel, then identify which horizontal and which vertical line the pixel is lying on. The deviation is then calculated using the perspective transform matrix. The calculated deviation for this pixel is displayed on the image and the maximum, minimum and mean deviations in this video frame will also be calculated. Figure 4-14 shows the design of the interface.

Left clicking in the input image area will mark a pixel on the input video frame. The coordinates of the pixel can be calculated. Then by right clicking the marked pixel, a context menu pops up, in which the human operator can identify which intersection it is.

By selecting the 'Calculate' menu, the coordinates of the pixel are converted to the coordinates in the rugby field and displayed in a pop-up window. By observation, the human operator can estimate a range of coordinates in which the pixel should be converted to. In this way, human operator can have a rough impression of the accuracy of the pixels not lying in the intersections of the lines by comparing the calculated coordinates and the estimated coordinates. The 'Erase' menu allows the human operator to delete a marked pixel in cases of wrong marking. The check boxes under the input and output images allow the human operator to enter the observation result of what is the problem in the processing of this frame. The last line in the interface shows the statistical data of the video frame.

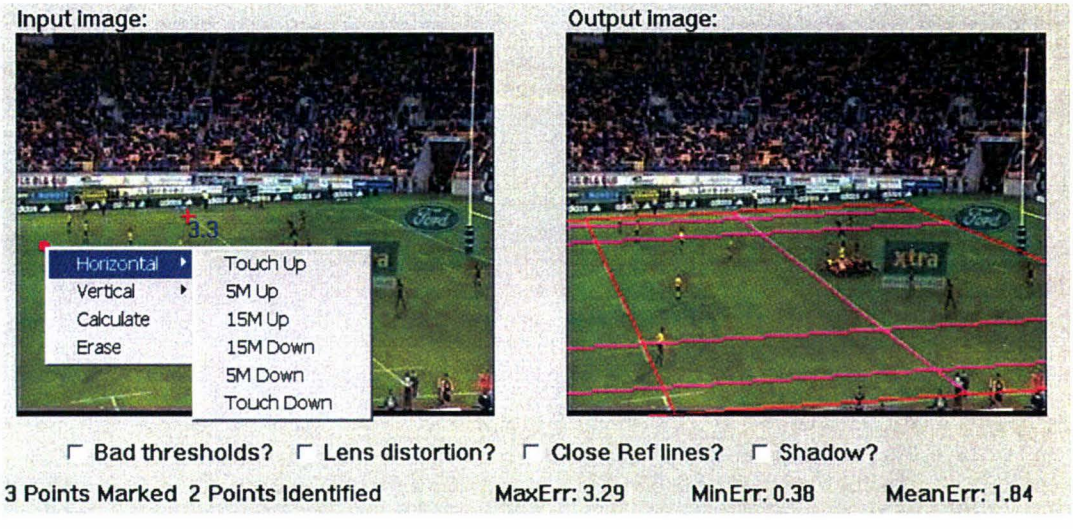


Figure 4-14 Interface for Testing Video Frame

Implementation Issues: To assess the quality and observe potential problems with the processed frame, a dialog has been developed to perform the tasks in accordance with those defined in Figure 4-14.

Requirement 10. Time consumption must be recorded and displayed.

Design Decision: A timer can be used to show the time consumption. The timer will start at the same time as the processing starts. If the human operator pauses the processing, the timer will also halt. When the processing is finished or stopped, the timer will also stop. After that the time consumption is written to the database, and the timer is reset.

Implementation Issues: A timer is placed in the status bar of the main interface, as can be seen in Figure 4-4.

Requirement 11. The human operator must be able to access the results of the processing easily.

Design Decision: A playback mode is designed so that the input and output image sequences can be viewed like playing a video. A slider can be placed in the main interface of the application. The human operator can access a video frame by changing the position of the slider.

To help analysing the outcome, an interface should be provided. Because a lot of statistical data need to be presented, multiple data pages are designed. A tab control is designed so that the statistics can be well organised. Figures 4-15 to 4-18 show the design of the interface for statistical data.

Figure 4-15 shows the general information of the task is shown. The information is retrieved from the VideoFiles table in the database.

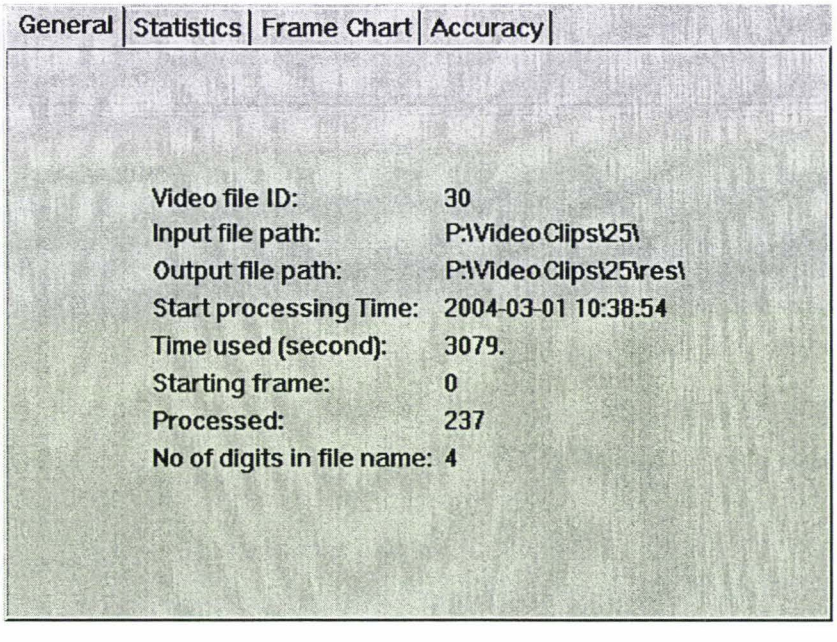


Figure 4-15 Statistics Interface --- General Information

Statistical data summarising the outcome of the task are presented as shown in Figure 4-16. In the 'Processed Frames' section, two kinds of information are given: the proportion of frames undergoing different line identification methods, and the proportion

of roles of the lines in the video clip. In the 'Tested Frames' section, statistics in tested frames in the video clip are given.

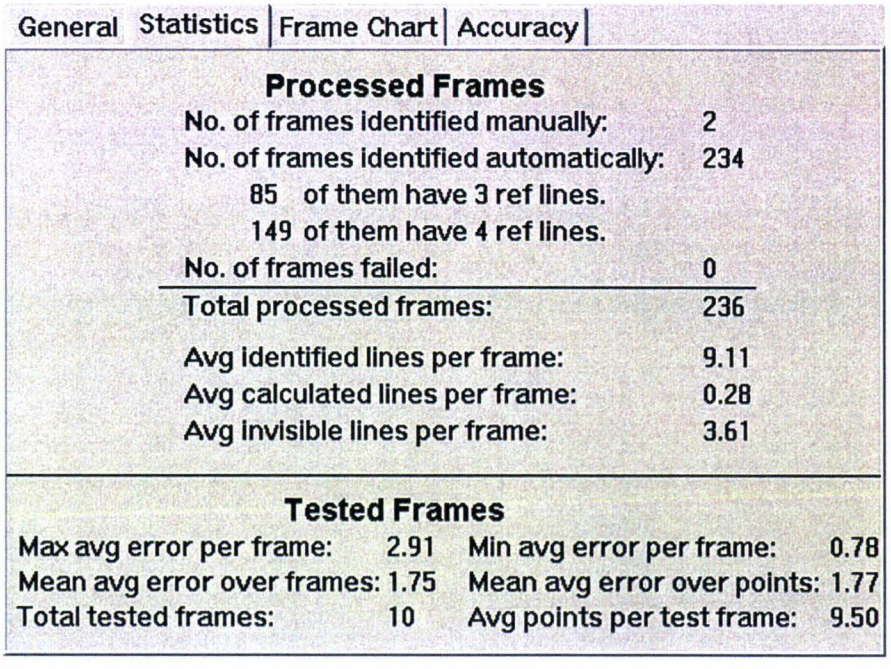


Figure 4-16 Statistics Interface --- Statistics

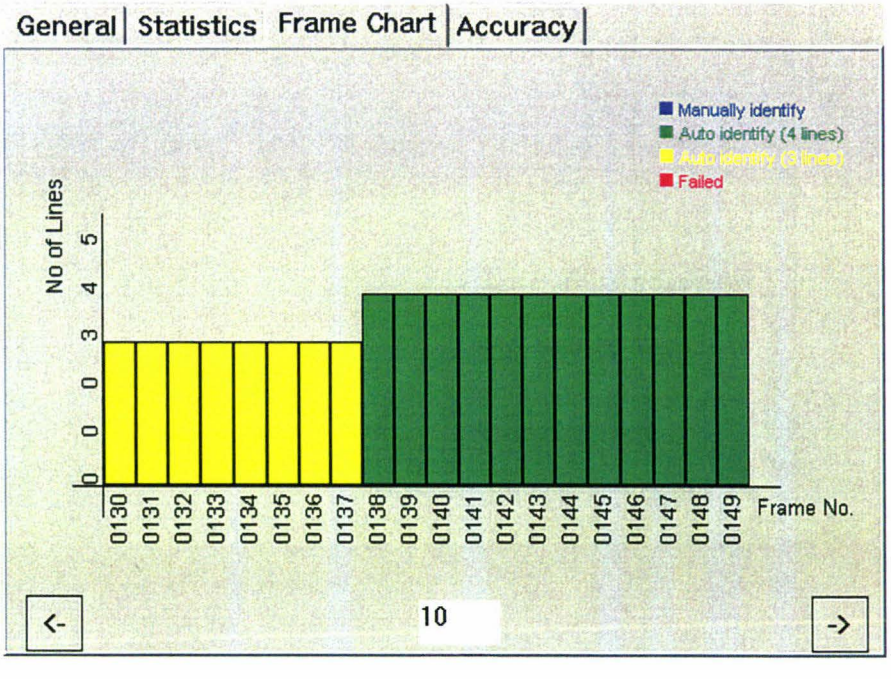


Figure 4-17 Statistic Interface --- Frame Chart

In Figure 4-17, a bar chart is used to represent the number of reference lines of each frame in the video clip. The bars are aligned in the ascending order of frame number.

Only 20 bars are displayed in one screen, so the '<-' and '->' buttons are used to move the bars left and right in case there are more than 20 frames in the task. The edit box between the two arrows is used to define the moving speed of the bars. The bars are drawn in different colors revealing the different number of reference lines. As the user may want to see more information on a certain video frame, double-clicking the bar representing the frame will open the testing interface of that video frame.

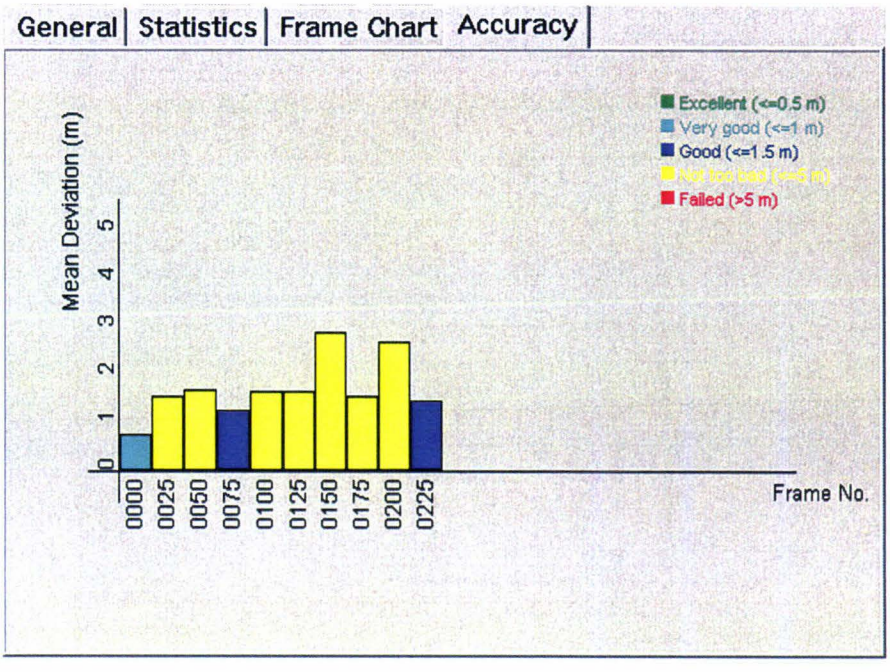


Figure 4-18 Statistic Interface ---- Accuracy

In Figure 4-18, the mean deviation of the video frame (DF) is displayed in a bar chart. Different colors are used to represent different ranges the mean deviations fall in. Double-clicking the bar will also open the testing interface of that frame.

Implementation Issues: A playback button is put in the tool bar of the main interface, which when set, will play the image sequence frame by frame.

A dialog is created in the main project to show the statistical data to the user. Figure 4-19 shows the interface of the dialog. It is derived from Figures 4-15 to 4-18.

One list box has been added to the left of the interface to allow viewing the statistical results of other tasks. The Reload button is used to refresh the display in cases where some frames in this video clip have just been tested. When testing the quality of processing for one video clip, manually clicking in the frame chart to bring out the testing

dialog is not convenient, so the Test button is added. When it is clicked, a testing dialog will pop up automatically for frames in regular intervals (currently the interval is set to 25). The Resume button allows the user to continue an unfinished task.

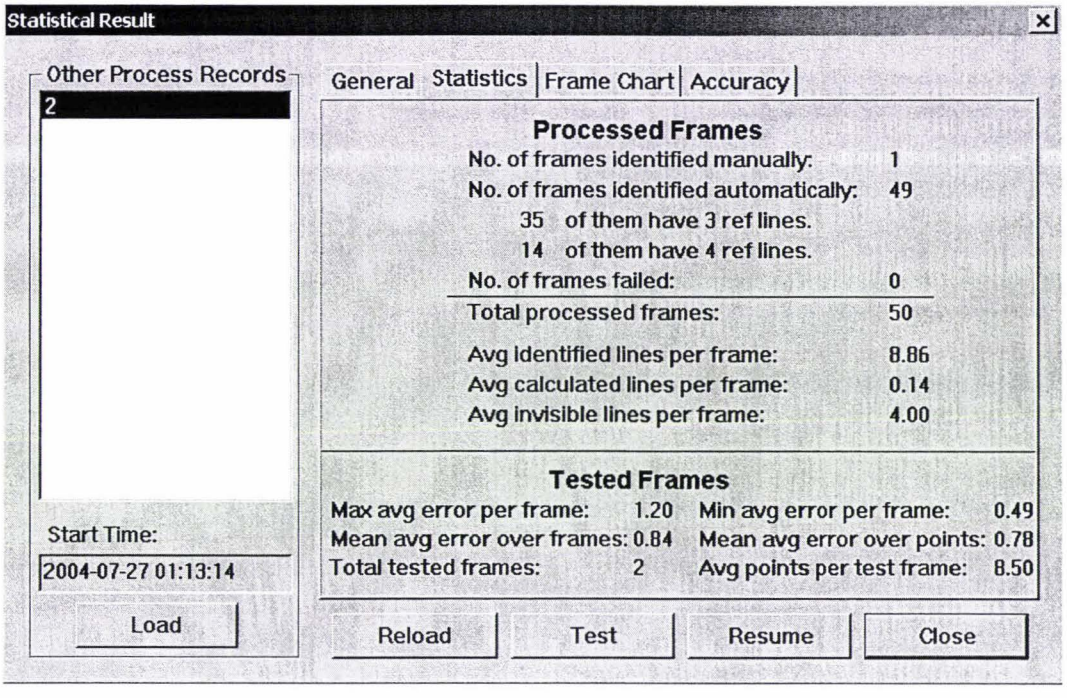


Figure 4-19 The Statistical Result

4.4 Summary

In this chapter, the design and implementation issues of the application were discussed.

The MATLAB image-processing toolbox was selected as the library for developing and testing the algorithms. Visual C++ was chosen as the development environment for the application as MATLAB add-in tools can be used to create DLLs, which can be integrated into the VC++ application easily. Microsoft Access was selected as database system as it is simple and has a user-friendly interface.

Finally the design decisions and implementation issues addressing the requirements were discussed.

5 Experimental Results

In order to assess the quality and efficiency of the developed algorithms and application, tests have been performed. In this chapter, the pre-conditions, steps, and the results of the testing are discussed.

5.1 Background

Tests have been performed in order to find out how good the application is and what the main requirements for the future development of the project are. According to Section 3.2.3, these tests mainly focused on two parts: the quality of line tracking, that is, what is the mean deviation between calculated reference point and the point identified by a human observer; the efficiency of the application, that is, the speed of the processing.

Fifty video clips were selected by AnalySports to be used in testing the application. The selected video clips cover a range of different conditions: stadium, lighting, camera movements, and so on. Each video clip covers a continuous segment of a rugby game. Most of the video clips are long shots, which means the camera is taking a global view of the field. Thus it is possible to find enough reference lines to calculate the transformation matrix. All video clips are in MPEG-1 format.

5.2 Steps for Line Processing

For every video clip, the following steps are performed to generate the output information.

- The application only accepts image sequence as input. Therefore the pre-processing was performed, and the time for the pre-processing was not logged, as it is not implemented as part of the application.

- Finding good thresholds is essential to successfully detect the lines in the video clips. Before processing was started, a new set of thresholds was created for each video clip. In the thresholds-selection interface, the human operator marked lines in the video frame. The application then generated histograms of color components of the selected pixels. The human operator selected a set of suitable values as thresholds to be used in the color filter.

- After the processing was started, the human operator identified lines when needed, stopped the processing and modified the settings if necessary. The operator tried to apply the same stopping-rule to all the video clips, that is, to stop the processing manually only if the calculated matrix was obviously wrong.

- To limit the processing duration, only the first 200 frames of each video clip were processed.

5.3 Steps for Assessment

To assess the quality of processing, sample images were selected from the processed frames. For each selected frame, test points were marked manually and the "mean deviation" (*FD*) of that frame was calculated. For each video clip, the average of the frame-level "mean deviations" (*CD*) was calculated as the key data representing the overall quality of the processing.

To assess the efficiency of the application, the time consumption for the processing of frame sequences was recorded. Based on this information, the time consumption to process a complete recording of a rugby game can be estimated.

The following steps were performed to generate the test results.

- The human operator marked intersections of lines on the image as reference points. The application calculated the deviation on these points (D_P), and calculated the mean deviation for that frame. The mean deviation is then saved to a database by the application.

- For each video clip, the processing quality of every 25^{th} frame was tested. As 200 frames are processed, nine frames were tested for each video clip.

- For each video clip, the average and maximum of the mean deviations for all reference frames were calculated by querying the database.

5.4 Results

❖ Overall success rate

Among the 50 video clips, the application processed 34 (68%) video clips successfully. The application failed to process 16 (32%) of the 50 video clips, among them:

- 4 video clips failed because there are large areas of shadows on the field. It is impossible to find one set of thresholds for both dark and light areas.

- 11 video clips failed because good thresholds for the color filter could not be found. This was caused by the quality of the video clips. Most of these video clips have poor quality (that is, poor lighting conditions, bad weather, faint lines, and so on), which raised the difficulties in finding good thresholds.

- 1 video clip failed because of camera zooming, and not being able to find 4 reference lines.

❖ Statistics in the maximum and average mean deviation

The application successfully processed the required 200 frames for 34 video clips. The average value of deviation between calculated and observed coordinates over all tested points is 2.25 meters. 22 of 34 video clips have an average of "mean deviation" of less than 2 meters. Appendix A gives the detailed values for the test result of the fifty video clips.

❖ Time consumption

The performance of the application is different in different computers.

- In an ASUS laptop with Mobile Intel® Celeron™ CPU 1200 MHZ, 256M Ram (shared with display memory), running Microsoft Windows 2000 Professional (Service Pack 4), on average, the application takes 4.1 seconds to process one frame.

- In an Advantage desktop with Intel® Pentium® 4 CPU 2.40 GHZ, 504M Ram, running Microsoft Windows XP Professional (Service Pack 1), on average, the application takes 3.5 seconds to process one frame.

5.5 Analysis of Processing Problems

When assessing a video frame, the human operator is asked to manually observe which problem causes the error. As described in Section 3.2.3.1, four problems are identified and logged.

Among these four problems, the bad thresholds problem is one of the main reasons that cause the processing to fail. There would be a number of ways to overcome this problem. For example, one simple way would be to use video clips of better quality. Another way would be to change the line detection technique, use another image segmentation technique or a combination of several techniques to obtain a better detection of lines in the field.

Lens distortion is very common in the video clips. Among the 34 successful video clips, 21 have serious lens distortion problems. Normally this will not make the processing fail, but will bring extra errors to the process results. The average value of the 21 video clips' average mean deviations is 2.64 meters. To solve this problem, camera calibration could be performed before the processing of the video frames.

12 of the 34 successfully processed video clips have close-reference-lines problems, causing large errors in the process results. In these cases the maximum of the mean deviation is usually large with the average of the maximum of mean deviation over these 12 video clips being 4.35 meters. This problem could be solved by adding a guard to the application to detect such close reference lines, by performing extra checking on the reference lines and by asking the human operator to make corrections if needed.

Shadow is another serious problem. Among all the video clips, there are 6 video clips that have a shadow problem and 4 of them failed because neither of the two parts of the field contains 4 detectable lines. The application can find enough reference lines for the other 2 video clips, but both have an average-mean-deviation of larger than 5 meters. One possible way to solve this problem is to introduce multiple sets of thresholds to the color filter, combine the outputs of different sets of thresholds and use the combination as input into the Hough transformation.

The time consumption of the application is quite large; the line identification has a time factor of about 100 (that is, for a one-second video clip, it will take about 100 seconds to

process), which means it will take a long time to process one rugby game on one computer. One way towards reducing processing times would be to upgrade the hardware. A faster PC or multiple PCs could be used. Another very promising approach would be to optimize the algorithms so that not every frame in the video clip will be processed. Instead, the application would skip several frames if it detects that there is little camera movement and therefore the reference lines would change only little. If only 1 in every 25 frames were processed, and four computers could be used to run the application simultaneously for different sections of a rugby game, it would be possible to process the video of one game in less than two hours.

5.6 Summary

In this chapter, the tests performed to assess the quality and efficiency of the developed algorithms and application were introduced.

Sections 5.1 to 5.3 introduce the steps for processing and testing the video clips. The test results on the quality of the algorithm and efficiency of the application are covered in Section 5.4. In Section 5.5, the observation results made by the human operator on the problems in the processing are introduced and suggestions for improvements are made.

6 Conclusions and Future Work

6.1 Conclusions

Many computer systems have been developed to help coaches and trainers to analyze the performance of the athletes and to develop tactics. Video analysis usually plays an import role in such systems. The activities of the athletes in the video recordings can be observed and logged to the computer by human operator. But the positions of the athletes are hard to measure, especially in video recordings taken in uncontrolled environments. The major difficult comes from the fact that in uncontrolled environments, the camera's position and parameters are unknown and non-fixed. With a moving camera, problems arise in many aspects, such as the detection of the moving objects and the registration of the detected object from the video frame to the sports field.

The aim of the wider project related to this thesis is to develop a player tracking system. Video recordings from televised rugby games are used as input data. The computer system would process the video recording and produce information on the positions of the players in the rugby field. As the camera used is moving to follow the hotspots in the rugby field, the motion detection and tracking techniques for static background video recordings cannot be used in this situation. Therefore, field characteristics are used as reference in measuring the positions of the players. The field must be identified for every video frame because its characteristics in relation to the video frames are not fixed. In order to reduce the time consumption in recognizing the field characteristics manually for every frame, detection and tracking systems must be developed to semi-automatically track the field characteristics. This one-year thesis focuses on the developing of such a line detection and tracking system.

An application containing line detection and tracking algorithms has been developed to process the televised video recordings of rugby games. With this technique, a transformation between the coordinates in the recorded image and the coordinates in the real rugby field can be performed. This establishes a very important part of the overall player-tracking project. The application also contains components for changing parameters in the processing and storing data in the database. Testing for assessing the quality of the line tracking has been undertaken and shown very positive results for the number of videos tested. Problems stemming largely from the light and distortion characteristics of the video recordings have been identified and solutions suggested.

This work provides a major step along the way to player tracking. The identified lines provide the reference points to calculate the players' positions on the rugby field. Player recognition/tracking can be added to the current application following much the same pattern as line tracking.

6.2 Future Work

This project can be improved or developed further in many directions.

Work can be done to improve the algorithms using image segmentation techniques. The developed application uses multi-dimensional thresholding to detect pixels on the lines. This image segmentation technique is fast and simple, at the same time, unlike edge-based image segmentation techniques, it only responds once to a line in the image. Yet it may fail to distinguish two colors, as the shape of the filter used in it is always cuboids. The biggest problem with edge-based image segmentation techniques is that they will also responds to noise (such as advertisements and audiences). Therefore they can only be used in combination with color-based image segmentation techniques (for example, thresholding or multi-dimensional thresholding). An attempt has been made to combine an edge-based image segmentation technique with a color-based image segmentation technique (thresholding) in Section 3.2.2.2. The result shows that only filtering in one component of the color space (Hue) is insufficient. Further research on adding filters in the other color components (that is, changing from thresholding to multi-dimensional thresholding) could be performed.

The lens distortion problem remains untreated in this project due to the complexity of this problem and the lack of control of the capturing environment. Typically, camera calibration is done to correct lens distortion. But in this project, as televised video recordings are used, performing camera calibration is not feasible. Research can be performed in this direction, as dealing with lens distortion effects would greatly increase the quality of the processing.

AnalySports Ltd. can now work towards integrating this application into their current systems to provide position information of the players. This will require to manually identifying the players positions in a long shot video recording and the synchronization of long shot and medium/close-up shots, as used in their current video analysis system. By examining the players' activities together with the players' positions, as well as the nearby players' positions, the coaches will be better able to assess the players'

performance. Furthermore, as all the players' positions are known, the perspective view of the rugby game can be converted to a bird eye view. The players' movement trajectories can be drawn in the bird eye view, which will be very helpful in analysis and development of team tactics.

Based on the line detection and tracking application, the player recognition and tracking system can be implemented. When the player positions are found in the video frame, the perspective transform matrix produced in the line detection and tracking system can be used to automatically convert the player positions in the image to positions in the rugby field.

References

Antollini, C. (2002). ADO Connection Strings. Retrieved (December 18, 2003) from The Code Project on the World Wide Web: http://www.codeproject.com/database/connectionstrings.asp

Antollini, C. (2003). A set of ADO Classes (Version 2.10) [Computer software]. Retrieved (December 18, 2003) from The Code Project on the World Wide Web: http://www.codeproject.com/database/caaadoclass1.asp

Bailey, D. G. (2002). A New Approach to Lens Distortion Correction. *Proceedings of Image and Vision Computing New Zealand 2002*, pp 59-64. Auckland: University of Auckland.

Baxes, Gregory. A. (1994). *Digital Image Processing: Principles and Applications*. New York: John Wiley & Sons, Inc.

Borland Delphi (Version 6.0) [Computer software]. (2001). USA: Borland Software Corporation.

Chand, M. (2000). Microsoft Data Access Components (MDAC). Retrieved (July 18, 2004) from Mindcracker on the World Wide Web: http://www.mindcracker.com/mindcracker/c_cafe/database/mdac.asp

Ekin, A. and Tekalp, A. M. (2002). A Framework for Tracking and Analysis of Soccer Video. *Visual Com. and Image Proc. (VCIP)*, San Jose: SPIE proceedings. Retrieved (September 20, 2003) from University of Rochester on the World Wide Web: http://www.ece.rochester.edu/users/ekin/papers/CPapers/vcip02_ekin.pdf

Frédéric, P. (2003). An Introduction to Digital Image Processing. Retrieved (October 30, 2003) from GameDev.net on the World Wide Web: http://www.gamedev.net/reference/articles/article2007.asp

Gajic, Z. (n.d.). Delphi versions: from 1 to 7 and Counting. Retrieved (July 20, 2004) from About Inc on the World Wide Web: http://delphi.about.com/cs/azindex/a/dhistory.htm

Hamilton, E. (1992). JPEG File Interchange Format (version 1.02). Technical report, C-Cube Microsystems. Retrieved (June 19, 2004) from ftp://ftp.uu.net/graphics/jpeg/jfif.txt

Haralick, R. M. and Shapiro, L. G. (1985). Survey Image Segmentation Techniques. *Computer Vision, Graphics, and Image Processing, An International Journal, 29(1), 100-132.*

Hakobyan, A. (2002). Folder Selection Dialog Class [Computer software]. Retrieved (December 18, 2003) from The Code Project on the World Wide Web: http://www.codeproject.com/dialog/cfolderdialog.asp

Holdaway, C. (2002). *Tracking using Image Processing Techniques.* Unpublished Honours report, Massey University, New Zealand.

Intel Image Processing Library (Version 3.1) [Computer software]. (2002). USA: Intel Corporation. Retrieved (April 8, 2003) from Intel Corporation on the World Wide Web: http://developer.intel.com/software/products/perflib/ijl/

Intel Integrated Performance Primitives (Version 4.0) [Computer software] (2004). USA: Intel Corporation. Retrieved (August 20, 2004) from Intel Corporation on the World Wide Web: http://developer.intel.com/software/products/ipp/

Jacobson, J. (2002). Delphi Vs Visual C++. Retrieved (July 24, 2004) from http://www.programmazione.it/index.php?entity=earticle&idArticle=675&idArea=1

Leavers, V.F. (1992). *Shape Detection in Computer Vision Using the Hough Transform.* London: Springer-Verlag.

Lee, A. (2003). VirtualDub (Version 1.53) [Computer software]. Retrieved (March 30, 2003) from http://www.virtualdub.org/

Loomis, J. (1997). 2D Transformations. Retrieved October 19, 2003 from University of Dayton on the World Wide Web: http://www.udayton.edu/~cps/cps460/notes/2dtrans

Losinger, C. (1997). JpegFile - free JPG (and BMP) reading and writing code. Retrieved (December 12, 2003) from Smaller Animals Software on the World Wide Web: http://www.smalleranimals.com/jpegfile.htm

Luo, D. (1998). *Pattern Recognition and Image Processing*. Chichester: Horwood Publishing.

Mai, L. C. (2000). Introduction to Computer Vision and Image Processing. Retrieved October 20, 2003 from Training of Computer Specialists on the World Wide Web: http://www.netnam.vn/unescocourse/computervision/computer.htm

MATLAB (Version 6.5) [Computer software]. (2002). Massachusetts: The MathWorks, Inc.

Microsoft Access (2000, Version 9.0.3821 SR-1) [Computer software]. (1999). USA: Microsoft Corporation.

Microsoft Visual C++ (Version 6.0) [Computer software]. (1998). USA: Microsoft Corporation.

Ohlamder, R., Price, K., and Reddy, D. R. (1978). *Computer Graphics and Image Processing, An International Journal, 8(1), 313-333.*

Ohta, Y., Kanade, T. and Sakai, T. (1980). Color Information for Region Segmentation. *Computer Graphics and Image Processing, An International Journal, 13(3), 222-241.*

Parker, J. R. (1997). *Algorithms For Image Processing and Computer Vision.* USA: John Wiley & Sons, Inc.

Perš, J. and Kovačič, S. (2001). Tracking People in Sport: Making Use of Partially Controlled Environment. *Computer Analysis of Images and Patterns: 9th International Conference Proceedings.* Retrieved (September 20, 2003) from Machine Vision Group on the World Wide Web: http://vision.fe.uni-lj.si/docs/JanezP/caip2001.pdf

Pooley, R. (n.d.). A C++ Introduction. Retrieved (July 20, 2004) from Heriot-Watt University on the World Wide Web: http://www.cee.hw.ac.uk/~rjp/Coursewww/CPPwww/

Programming with Borland Delphi. (n.d.). Retrieved (July 24, 2004) from The University of York on the World Wide Web: http://www.york.ac.uk/services/cserv/help/programming/pages.sup/delphi5.htm

Ritter, G. X. & Wilson, J. N. (1996). *Handbook of Computer Vision Algorithms in Image Algebra.* Florida: CRC Press.

Roth, G. (2002). 2D Transformations. Retrieved October 19, 2003 from Carleton University on the World Wide Web: http://www.cv.iit.nrc.ca/~cs410/downloads/twodtran.ppt

Russ, J. C. (1995). *The Image Processing Handbook* (2nd ed.). Florida: CRC Press.

Shah, M. and Kumar. R. (Ed.). (2003). *Video Registration.* USA: Kluwer Academic Publishers.

SportsCAD GOLD [Computer software manual]. (2002). Berlin: Seaside Software. Retrieved (June 15, 2003) from SportsCAD Motion Analysis on the World Wide Web: http://www.sportscad.com/goldinfo.htm

StadeXpert [Computer software manual]. (2003). Moncton: Advanced Fitness Designs Inc. Retrieved (August 8, 2003) from Advanced Fitness Designs Inc on the World Wide Web: http://www.af-d.com/pages/gb/pres_stadexpert_gb.htm

Uchiyama, T. and Arbib, M. A. (1994). Color Image Segmentation Using Completive Learning. *IEEE Transactions On Pattern Analysis And Machine Intelligence, 16(12).* Retrieved (October 1, 2003) from IEEE on the World Wide Web: http://www.ieee.org

Appendix A

The test results are shown in Figure A-1 to Figure A-5.

Video No	No of Frames	Comments on settings	Comments on thresholds	No of frames processed Total(Manual/Success(4RefLines/3RefLines)/Fail)	Quality?	Bad Thresholds	Lens Distortion	Ref Lines too Close	Shadow	Max Mean Err	Avg Mean Err	Comments
1	858	Use default	Need to change thresholds; good thresholds can be found for the beginning of the video; but for frames close to the end of the video, it's hard to find good thresholds, reason: 1. Lines are not so clear in the field; 2. One team is wearing white uniform cloth, which produces lots of noise in the output of color-filter	830(but about 50 failed at the end of the sequence)	Good when 4 lines identified as references	0	0	6	0	1.95	1.32	
2	912	Increase minValTrk to 30, to reduce the effect of noise	Horizontal lines too far away from the camera can not be filtered out, manually marking a line possibile	785	The distortion of the lens affect the quality seriously when using 3 reference lines	4	7	0	0	7.86	2.78	
3	2869	Increase minValTrk to 25, decrease diffAng and diffDist to 1 and 2, to stop when the distortion is serious.	Good detection for vertical lines and closer horizontal lines	645	The distortion of the lens affect the quality seriously when using 3 reference lines; sometimes the camera moves too fast, making the tracking very difficult	2	2	0	0	2.85	1.76	
4	1030	Increase minValTrk to 25, decrease diffAng and diffDist to 1 and 2, to stop when the distortion is serious.	Good detection for vertical lines and closer horizontal lines	427	Some mis-detection on far-away horizontal lines	4	0	0	0	7.70	1.65	
5	1078	Increase minValTrk to 20, decrease diffAng and diffDist to 1 and 2, to stop when the distortion is serious.	Good thresholds could be found for most of the frames	251(8/243(155/88)/0)	Good, due to the lines are very clear so that color-filter produces good outputs	3	0	0	0	2.20	1.12	
6	885	Increase minValTrk to 20	Not able to find a set of thresholds, due to the quality of the video recording	0	N/A							

Figure A-1 Test Results

Video No	No of Frames	comments on settings	comments on thresholds	No of frames processed Total(Manual/Success(4RefLines/3RefLines)/Fail)	Quality?	Bad Thresholds	Lens Distortion	Ref Lines too Close	Shadow	Max Mean Err	Avg Mean Err	Comments
7	862	Increase minValTrk to 20	Color-filter could find lines, but lots of noise due to one of the teams wearing white-color cloth	117(2/115(94/21)/0)	Sometimes the noise makes the detected lines differ from where it should be	4	0	0	0	2.38	0.75	From frame 117, not able to find good thresholds
8	1730	Increase minValTrk to 30	Color-filter could find lines, but lots of noise due to one of the teams wearing white-color cloth	82(2/76(73/3)/4)	It's hard to find thresholds to tell lines from players; and sometimes the noise makes the detected lines differ from where they should be							
9	1338	Increase minValTrk to 30	Horizontal lines not detectable	6(2/3(3/0)/1)	Only vertical lines detectable, so not able to trace the lines							
10	1216	Decrease diffAng and diffDist to 1 and 1, to reduce the effect of noise	Horizontal lines not detectable	101(4/97(84/12)/1)	The quality of tracing of horizontal lines is very poor. So the overall quality is bad	8	0	0	0	5.53	4.56	From frame 101, not able to find good thresholds
11	1401	Increase minValTrk to 20, decrease diffAng and diffDist to 1 and 2, to stop when the distortion is serious.	Lines could be filtered out, some noise	191(1/188(94/94)/2)	Good	1	0	0	0	1.61	1.01	
12	982	Increase minValTrk to 20	Lines could be filtered out, some noise	168(4/163(125/38)/1)	Some noise, sometimes cause mis-identification	2	6	0	0	3.09	1.87	
13	1226	ID: 16, using default	Lines filtering OK	311(1/310(146/164)/0)	Good, with some errors, caused by lens distortion	0	9	6	0	3.79	2.89	
14	1898	Using default settings	Far-away horizontal lines could not be detected, others OK	203(4/199(195/4)/0)	Good, with some errors, caused by lens distortion	0	1	0	0	2.50	1.07	
15	958	Increase minValTrk to 20, decrease diffAng and diffDist to 2 and 2, to stop when the distortion is serious.	Good	110(3/107(0/107)/0)	Good, with some errors, caused by lens distortion	0	9	0	0	4.11	2.48	
16	882	Using default settings	Good	305(12/293(188/1050/0)	Lens distortion causes serious errors	0	5	2	0	3.37	1.75	

Figure A-2 Test Results (cont.)

Video No	No of Frames	Comments on settings	Comments on thresholds	No of frames processed Total(Manual/Success(4RefLines/3RefLines)/Fail)	Quality?	Bad Thresholds	Lens Distortion	Ref Lines too Close	Shadow	Max Mean Err	Avg Mean Err	Comments
17	924	Using default settings	Because some of the field is covered by the shadow of the stadium roof i.e. the field is devided into two areas-- one darker, one lighter; so there are two wave crest in the histogram; only one could be chosen at a time;	73(3/70(28/42)/0)	Only some segments of the video are trackable, for example, for the beginning of the video, none of the two areas contain enough lines to calculate the transform matrix							
18	1742	Similar to video clip 17										
19	1341	Using default settings	Lots of noise	207(1/206(206/0)/0)	Not too bad	7	0	0	0	3.50	1.62	
20	909	Increase minValTrk to 15, to reduce the effect of noise	Good detection for vertical lines and closer horizontal lines	52(9/43(42/1)/0)	Sometimes lens distortion causes serious errors							
21	2745	Similar to video clip 17										
22	2037	Increase minValTrk to 15, to reduce the effect of noise	Good	150(2/148(78/70)/0)	Good, but sometimes lens distortion causes serious errors	3	8	2	0	8.58	4.08	
23	725	Increase minValTrk to 30, to reduce the effect of noise	Good	355(7/348(144/204)/0)	Good	0	9	6	0	3.78	2.55	
24	1057	Using default settings	Good detection for vertical lines and closer horizontal lines	152(12/140(0/140)/0)	Because two horizontal reference-lines are close to each other, small errors in reference lines will cause big errors in areas far away from the reference lines	0	8	7	0	4.57	3.36	
25	1675	Increase minValTrk to 15, to reduce the effect of noise	Very good	236(2/234(85/149)/0)	Very good	0	7	0	0	2.91	1.77	
26	1045	Using defaultsettings	Good	121(1/120(0/120)/0)	Because two horizontal reference-lines are close to each other, small errors in reference lines will cause big errors in areas far away from the reference lines	0	8	9	0	4.65	3.61	
27	1318	Increase minValTrk to 20, to reduce the effect of noise	Good detection on vertical and closer horizontal lines	318(1/317(172/145)/0)	Good, some lens distortion problem	1	6	0	0	4.27	1.92	

Figure A-3 Test Results (cont.)

Video No	No of Frames	Comments on settings	Comments on thresholds	No of frames processed Total(Manual/Success(4RefLines/3RefLines)/Fail)	Quality?	Bad Thresholds	Lens Distortion	Ref Lines too Close	Shadow	Max Mean Err	Avg Mean Err	Comments
28	1134	Increase minValTrk to 15, to reduce the effect of noise	Good	246(2/244(98/146)/0)	Good, some lens distortion problem	0	9	0	0	2.05	1.81	
29	1112	Increase minValTrk to 15, to reduce the effect of noise	Very good	158(1/157(0/157)/0)	Very good	0	0	0	0	2.16	1.57	
30	1710	ID:43, increase minValTrk to 15, to reduce the effect of noise	Good, with some noise	76(2/74(0/74)/0)	Good, some lens distortion problem	0	9	0	0	2.45	1.68	
31	1197	Decrease diffAng and diffDist to 1 and 2, to reduce the effect of noise	Some lines could not be detected	139(6/133(27/106)/0)	Good, some lens distortion problem	0	7	0	0	1.85	1.22	
32	1037	ID:45, increase minValTrk to 20, decrease diffAng and diffDist to 1 and 2; decrease maxDist to 7, to detect two very close lines	Good	186(5/181(43/138)/0)	Serious lens distortion problem; and because two horizontal reference-lines are close to each other, small errors in reference lines will cause big errors in areas far away from the reference lines	0	9	9	0	10.23	4.07	Stopped at 185, ask for human identification of lines, but only 3 lines in the frame 186
33	837	Similar to video clip 32				0	8	9	0	2.33	1.70	
34	1262	Using default settings	Good	152(1/151(57, 94)/0)	Serious lens distortion problem	0	9	0	0	3.60	2.76	
35	1076	Increase minValTrk to 20, to reduce the effect of noise	Good	276(1/275(44/231)/0)	Good, Some lens distortion problem	0	9	0	0	2.03	1.64	
36	1542	Using default settings	Good	380(2/378(5/373)/0)	Good	0	0	9	0	1.76	1.40	
37	1006	Using default settings	Good, lots of noise caused by the text added to the video	218(1/217(7/210)/0)	Good, but sometimes noise causes serious errors	0	7	0	0	2.87	1.69	
38	1052	Increase minValTrk to 20, to reduce the effect of noise	Lines could be detected but with lots of noise	217(2/215(182/33)/0)	Good, Some lens distortion problem	0	6	0	0	3.04	1.96	

Figure A-4 Test Results (cont.)

video No	No of Frames	comments on settings	comments on thresholds	No of frames processed Total(Manual/Success(4RefLines/3RefLines)/Fail)	Quality?	Bad Thresholds	Lens Distortion	Ref Lines too Close	Shadow	Max Mean Err	Avg Mean Err	Comments
39	2481	Using default settings	Lots of noise	111(5/106(0/106)/0)	Good, but sometimes noise makes the result not so accurate	6	0	9	0	1.79	1.17	From frame 118, not able to find good thresholds
40	2389	Decrease maxDist to 6, to separate two close lines	Good	134(16/118(1/117)/0)	Not bad, because two horizontal reference-lines are close to each other, small errors in reference lines will cause big errors in areas far away from the reference lines	0	0	9	0	6.65	3.08	
41	2254	Using default settings	Only vertical lines could be detected correctly	239(3/236(234/2)/0)	The quality is poor due to the horizontal lines not correctly detected	9	0	0	0	2.91	2.11	
42	852	Increase minValTrk to 20, to reduce the effect of noise	Far-away horizontal lines could not be detected, others OK	37(2/35(13/22)/0)	The quality is poor due to the horizontal lines not correctly tracked							
43	1088	Increase minValTrk to 20, to reduce the effect of noise	Far-away horizontal lines could not be detected, others OK	116(3/81(79/2)/32)	Tracking failed due to camera zooming in and horizontal lines not detectable							
44	1305	Using default settings	Not able to clearly detect horizontal lines	0	The quality of video makes it impossible to find horizontal lines							
45	1116	Using default settings	Not able to find any horizontal line	0	The quality of video makes it impossible to find horizontal lines							
46	1706	Decrease maxDist 6, diffAng 1, diffDist 2	Good	12(3/9(0/9)/0)	Because two horizontal reference-lines are close to each other, small errors in reference lines will cause big errors	0	0	9	0	2.89	1.76	
47	2681	Decrease maxDist to 7, to separate two close lines	Good	117(5/112(0/112)/0)	Good	1	0	9	0	2.12	1.58	
48	1044	Similar to video clip 17				0	9	9	9	7.47	5.55	
49	961	Similar to video clip 17				0	9	9	9	10.19	6.43	
50	1557	Similar to video clip 17										

Figure A-5 Test Results (cont.)