國立交通大學

多媒體工程研究所

碩 士 論 文

籃 球 影 片 中 的 球 員 追 蹤 與 戰 術 分 析

Player Tracking and Tactic Analysis in Basketball Video

研 究 生：伏宗勝

指導教授：李素瑛　教授

中 華 民 國 一 百 年 三 月

籃 球 影 片 中 的 球 員 追 蹤 與 戰 術 分 析
Player Tracking and Tactic Analysis in Basketball Video

研 究 生：伏宗勝　　Student：Tsung-Sheng Fu

指導教授：李素瑛　　Advisor：Suh-Yin Lee

國 立 交 通 大 學
多 媒 體 工 程 研 究 所
碩 士 論 文

A Thesis

Submitted to Institute of Multimedia Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

January 2011

Hsinchu, Taiwan, Republic of China

中華民國一百年三月

籃球影片中的球員追蹤與戰術分析

研究生：伏宗勝　　　　　　　　　　指導老師：李素瑛 教授

國立交通大學多媒體工程研究所

摘　要

隨著電視轉播技術的發展，越來越多人觀賞籃球比賽，但是大多數的人對於籃球知識並不是非常了解。我們也許會因為球員投進壓哨三分球而尖叫，或為一個強力灌籃而興奮，但是不見得知道球員是如何擺脫防守者進行投籃。目前已經有一些籃球影片內容的研究，例如精采畫面擷取和記分板辨識，但是這些仍然無法幫助觀眾對於籃球有更深入的了解。所以我們希望能設計一個系統來提供觀眾一些比較深入的籃球知識，而不只是表面上的資訊。籃球比賽中，觀眾最有興趣的就是得分。但是得分背後的戰術是一門很深奧的學問，因為籃球是一項五個人的運動，不可能只靠一個球員去對抗另外一隊，也就是說單一球員很難靠他自己擊潰對方的防守並且進行得分。大部分的得分都是經由執行戰術而來的。所以我們的目標就是自動辨認出籃球比賽中執行的戰術，並且把這些收集來的資訊帶給觀眾，讓他們能更了解籃球這項運動；甚至可以提供教練和球員，作為他們訓練及了解敵隊的攻防策略之用。

籃球戰術種類眾多，很難用單一演算法一以概之，因此我們著重於大多數戰術中都會使用的「掩護」，加以偵測並且分類，藉此分析出戰術執行的模式。我們開發的系統執行步驟如下。在比賽一開始先收集整場比賽都不會變的資訊，包含球場地板顏色以及兩隊球衣顏色。首先我們計算攝影機的參數，並且產生一張

表示球場範圍的遮罩。第二步我們在球場範圍內計算出現次數最多的顏色，也代表著地板顏色。接著利用背景相減法，我們可以從球場範圍減去地板得到前景物體。最後我們利用顏色資訊將前景分成兩群，分別代表著兩支球隊的球衣顏色。因為這些資訊在整場比賽中都不會改變，所以我們可以利用它們降低往後的計算量，並且提升系統效能。在比賽中，針對每一次球權先分辨哪一隊是進攻方，了解雙方球員的行為模式才能判斷執行的戰術。利用先前得到的資訊並追蹤雙方球員的軌跡。在一波進攻結束的時候，根據追蹤到的雙方球員軌跡來判斷執行的掩護。經由實驗結果，我們開發的系統對於掩護的偵測和分類準確度相當令人滿意，因此在戰術分析上也有著顯著的幫助。這些被辨識出來的戰術會存入資料庫，於是觀眾就可以查詢他們有興趣的戰術並且學習。

關鍵字：籃球影片、球員追蹤、戰術分析、運動影片分析、影像處理

Player Tracking and Tactic Analysis in Basketball Video

Student: Tsung-Sheng Fu Advisor: Prof. Suh-Yin Lee

Institute of Multimedia Engineering

National Chiao Tung University

ABSTRACT

Thanks to the development of TV broadcasting technology, there are more and more people watching basketball games. Most of us, however, do not know the basketball sport very well. We may scream for a buzzer beater three-point shot or get excited about a slam dunk, but we do not exactly realize how a player gets rid of defenders and makes shots. There have been some researches on basketball video, such as highlight extraction and scoreboard recognition, but they still cannot help people further understand this sport. Therefore, we intend to design a system which provides audience with further knowledge of basketball instead of superficial information. In basketball games, people are most interested in scoring events. Nevertheless, scoring is not that simple as it looks. It can be an abstruse subject since basketball is a five-person sport and one player is not able to fight against the opponent team. That is, it is difficult for an individual player to break the defense and score by himself. Most shots are made through execution of tactics. Consequently, our goal is to automatically identify tactics executed in basketball games and bring audience the collected information so that they can learn more about the basketball sport.

There is plenty of basketball tactics, and it is hard to model them by a single algorithm. Hence, we focus on “screen,” which is widely used in most basketball tactics. We detect and classify screens, and regard their patterns as certain tactics. Our proposed system performs with the following steps. First of all, we gather some consistent information at the beginning of the game, including the floor color and the jersey colors of the two teams. We first compute the camera calibration and generate a court mask indicating the court region. Second, we calculate the dominant color within the court region, which represents the floor color. Next, we obtain the foreground objects by subtracting the floor from the court region. This procedure is similar to a background subtraction mechanism. Finally, we divide the foreground region into two clusters with color information. Thus, the two clusters denote the jersey colors of the two teams respectively. Since this information is consistent through the entire game, we can utilize it to reduce computational cost and accelerate the computation in the following frames. During the game, we first distinguish which team is on offense in each possession since we have to learn the behaviors of offensive and defensive players respectively in order to identify tactics. Next, we extract players of the two teams with the previously obtained information and track them. At the end of a possession, we identify what screens are set by the trajectories of the players. Through our experiment, the accuracy of screen detection and classification is satisfactory, which significantly helps analysis of basketball tactics. The identified tactics are then inserted into a database from which audience can query tactics they are interested in.

Keyword: basketball video, player tracking, tactic analysis, sports video analysis, image processing

Acknowledgement

First of all, I greatly appreciate my advisor, Prof. Suh-Yin Lee. Not only for her kind guidance, but also for her sincere help whenever I am troubled or upset. Next, I would like to thank my seniors Hua-Tsung Chen, Hui-Zhen Gu and Min-Chun Hu for their graceful ideas, precious experience and technical assists. Besides, I am grateful to my colleagues for their inspiration. Also, I have to thank my brother Kuang-Yu Fu, who is an expert in basketball and teaches me a lot. Last but not least, I appreciate my parents Hai-Ju Fu and Hsiao-Li Tung. Without their support and encouragement, I am not able to complete this achievement. I devoutly dedicate this thesis to them.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

There have been many researches on sports video analysis in the past decade. However, not much research is focused on broadcast basketball video analysis. Doing researches on basketball video, one may face some difficulties and challenges. Most of all, basketball players occlude each other very often. As a result, it is difficult to segment and track players correctly. Unfortunately, segmentation and tracking are the soul of video analysis. In other words, unless we overcome the occlusion problem, we are not able to analyze much content in basketball videos. Chang et al. proposed a method [1, 50] that can accurately separate players of different teams. This tremendously improved the possibility of basketball video analysis because in basketball games, in order to make wide-open shots, players of the same team seldom stay together. On the other hand, the defensive players usually stand next by his target to defend. That is, once we distinguish players of the two teams, we can avoid most occlusions. Second, in order for the audience to see the ball clearly, the camera usually follows the ball. This may lead to violent camera motions since the ball moves fast. Consequently, the camera calibration is another challenge. Farin et al. introduced a robust and efficient court model tracking algorithm [2], which helps us use the frame coherence to obtain the camera calibration with slight computational cost.

Besides, there is another question: what can we analyze in basketball videos? Some researches focus on event detection and highlight extraction [3-6]; others are interested in trajectory reconstruction [7]; still others concentrate on frame information, including shot classification [8] and scoreboard recognition [9]. Nevertheless, recent researches consider more information in video clips than in basketball sport itself. Our goal is to bring the audience further knowledge about basketball, or even to provide professional players and coaches with technical information. To achieve this goal, we put most effort in verifying the tactics executed in basketball games. Having surveyed hundreds of basketball tactics, we discovered that there is one fundamental essence – *screen*. A screen is a blocking move performed by an offensive player, by standing beside or behind a defender, in order to free a teammate to shoot, to receive a pass, or to drive in to score. Basketball tactics can be categorized by strategies which they are following and players whom the tactics are set for. Strategies include *isolation*, *low-post*, *high-post*, *mid-range*, *three-point*, *pick-and-roll*, and *pick-and-fade*. *Isolation* means that the team on offense tries to isolate a player and make a one-on-one attack. *Low-post* and *high-post* indicate the location where players start attacks. *Mid-range* and *three-point* are similar to low post and high post, describing the attack locations, but they focus on the finish of attacks instead of the beginning. *Pick-and-roll* and *pick-and-fade* strategies intend players to make open shots through screens. A tactic sometimes does not follow a specific strategy, and we categorize it as *general*. Furthermore, once the strategy is decided, a player is expected to shoot the ball. That is, tactic categories are then distinguished by the positions of players, namely, *point guard*, *shooting guard*, *small forward*, *power forward*, and *center*. In general, *point guards* (PG) organize the offense of a team; *shooting guards* (SG) are good shooters from long range; *small forwards* (SF) have high speed so that they usually drive in and break the defense of the opponent team; *power forwards* (PF) and *centers* (C) are the tallest players of a team and they behave most near the basket. Through our observation, most tactics consist of screens. Table 1.1 shows total number of surveyed tactics and number of tactics using screens. According to Table 1.1, over 80% of tactics contain screens. In other words, most basketball tactics are composed of different types of screens. Once we want to study a basketball tactic, we have to learn what types of screens are used in it first. In this thesis, therefore, we are focused on detecting screens and classifying their types.

Table 1.1: Tactic categories and number of tactics using screens.

Strategy \ Position	PG	SG	SF	PF	C	Overall
General	12/16	13/16	8/10	7/10	9/10	49/62
Isolation	8/16	10/16	7/16	6/16	1/4	32/68
Low Post	5/7	11/16	12/16	10/16	14/16	52/71
High Post	4/5	5/8	4/6	7/8	7/11	27/38
Three Point	11/12	15/16	14/16	6/8	6/7	52/59
Mid Range	13/16	14/16	14/16	13/16	15/16	69/80
Pick and Roll	16/16	16/16	16/16	16/16	16/16	80/80
Pick and Fade	16/16	16/16	16/16	14/14	2/2	64/64
Overall	85/104	100/120	91/112	79/104	70/82	425/522

In Chapter 2, we review previous works on object tracking and some applications in basketball video. In Chapter 3, we present our proposed system, including player tracking and tactic analysis. Chapter 4 shows our experimental results. At last, we will discuss the conclusion and future work in Chapter 5.

Chapter 2. Related Work

In this chapter, we will briefly introduce the methods for object tracking, and then show some recent researches on basketball video analysis.

2.1 Object Tracking

Object tracking is an important field in computer vision. When watching videos, we can easily distinguish objects and tell their behavior through our background knowledge. In computer vision, people want computers to recognize what objects are in videos and how the objects behave. Nevertheless, it is simple for people but difficult for computers to realize the video contents. Thus, many methods for object tracking have been proposed, and are introduced in the following sections.

2.1.1 Object Detection

Before tracking objects, we have to extract objects either in every frame or when they first appear in the video. That is, we will present the object detection methods before we start to discuss the object tracking algorithms. The object detection methods can be classified into four categories: point detectors, segmentation, background subtraction, and supervised learning [13]. Table 2.1 shows the four categories and their representative work, respectively.

Table 2.1: Object detection categories [13].

Categories	Representative Work
Point detectors	Moravec's detector [14], Harris detector [15], Scale Invariant Feature Transform [16]
Segmentation	Mean-shift [18], Graph-cut [19]
Background modeling	Mixture of Gaussians [21], Eigenbackground [22], Wall flower [23], Dynamic texture background [24]
Supervised classifiers	Support Vector Machine [25], Neural Networks [26], Adaptive boosting [27]

Point detectors are used to find points of interest in images which have an expressive texture in their respective region. To find points of interest, Moravec's operator [14] computes the variation of the image intensities within a 4-by-4 window in the horizontal, vertical, diagonal, and anti-diagonal directions, and then chooses the minimum of the four variations as representative values for the window. A point is declared interesting if the intensity variation is a local maximum in a 12-by-12 window. The Harris detector [15] computes the first order image derivatives in horizontal and vertical directions to emphasize the directional intensity variations, and then construct a structure matrix $\mathbf{S}_m$ over a small window around each pixel. The points of interest are identified by thresholding $R = det(\mathbf{S}_m) - k \cdot tr(\mathbf{S}_m)^2$, where $det(\mathbf{S}_m)$ represents the *determinant* of $\mathbf{S}_m$ and $tr(\mathbf{S}_m)$ denotes the *trace* of $\mathbf{S}_m$, after applying non-maxima suppression. Theoretically, the $\mathbf{S}_m$ matrix is invariant to both rotation and translation. However, it is not invariant to affine or projective transformations. In order to provide robust detection of interest points under different transformations, Lowe introduced the SIFT (Scale Invariant Feature

Transform) method [16], which is confirmed outperforming most point detectors and more tolerable to image deformations according to the survey by Mikolajczyk and Schmid [17].

The objects we are interested in are usually moving objects in videos. Frame difference is a typical method and is well studied since Jain and Nagel's work [28]. However, differencing temporally adjacent frames cannot achieve robust results under some circumstances. Thus, background subtraction became popular which builds a representation of the scene called the background model and regards any significant change in an image region from the background model as moving object. Stauffer and Grimson [21] use a mixture of Gaussians to model the pixel color. Each pixel is classified based on whether the matched distribution represents the background process. Instead of modeling the variation of individual pixels, Oliver et al. introduce an integral approach using the eigenspace decomposition [22]. It first forms a background matrix $\mathbf{B}$ of dimension $k \times l$ from k input frames of dimension $n \times m$, where $l = nm$. The background is then determined by the most descriptive eigenvectors.

Segmentation algorithms partition an image into regions of reasonable homogeneity. The mean-shift [18] method is proposed to find clusters in the spatial-color space, which is scalable to various other applications such as edge detection, image regularization [30], and tracking [31]. Shi and Malik [19] formulate image segmentation as a graph partitioning problem, where the vertices (pixels) are partitioned into disjoint subgraphs (regions), and overcome the difficulty of oversegmentation by the proposed normalized cut.

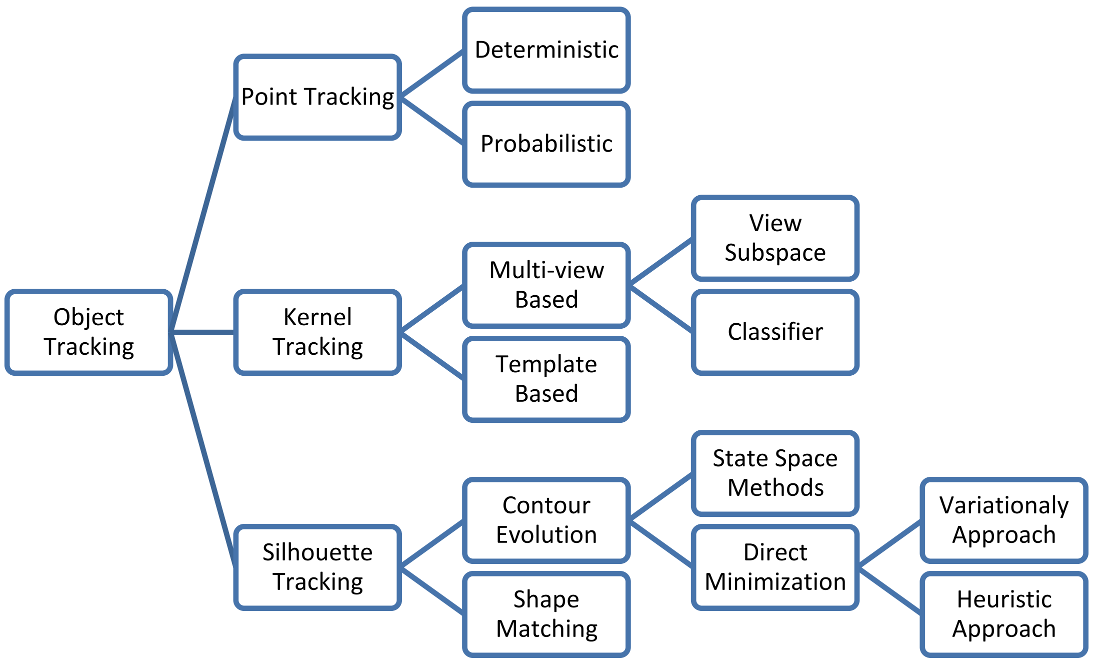


Figure 2.1: Taxonomy of tracking methods [13].

2.1.2 Object Tracking

The goal of object tracking is to gather the trajectory of a specific object. Take our system for example, since we intend to identify what tactics are executed, we have to analyze how the players move. That is, we must track players during the game in order to obtain their trajectories. Tracking algorithms can be classified into three main categories: point tracking, kernel tracking, and silhouette tracking. Figure 2.1 illustrates the taxonomy of tracking methods and Table 2.2 demonstrates their most notable works.

Detected objects over a video clip can be represented by points, and the point tracking finds the point correspondence across frames. Point tracking methods can be divided into two categories: deterministic and statistical methods. Deterministic methods define a cost of associating each object to a single object in two adjacent frames using a set of motion constraints, which is usually a combination of the constraints illustrated in Figure 2.2. *Proximity* assumes the location of the object would not change notably from one frame to other. *Maximum velocity* defines an upper bound on the object velocity and limits the possible correspondences to the circular neighborhood around the object. *Small velocity change* assumes the direction and speed of the object does not change drastically. *Common motion* constrains the velocity of objects in a small neighborhood to be similar. *Rigidity* assumes that objects in the 3D world are rigid, so the distance between any two points on the actual object will remain unchanged.

Table 2.2: Tracking categories [13].

Categories	Representative Work
Point Tracking	
Deterministic methods	MGE tracker [32], GOA tracker [33]
Statistical methods	Kalman filter [34], JPDAF [35], PMHT [36]
Kernel Tracking	
Template and density based appearance models	Mean-shift [31], KLT [37], Layering [38]
Multi-view appearance models	Eigentracking [39], SVM tracker [40]
Silhouette Tracking	
Contour evolution	State space models [41], Variational methods [42], Heuristic methods [43]
Matching shapes	Hausdorff [44], Hough transform [45], Histogram [46]

Statistical methods consider the measurement and the model uncertainties during object state estimation. State space approach is used to model the object properties such as position, velocity, and acceleration. Measurements usually consist of the object position in the image, which is obtained by a detection algorithm. The Kalman filter [34] computes the covariance for state estimation while the particle filter [47] uses the conditional state density to estimate the next state, which can be regarded as the generalized Kalman filter since the Kalman filter concentrates on estimating the state of a linear system where the state variables are assumed to be normally distributed (Gaussian) and the particle filter deals with the non-Gaussian state.

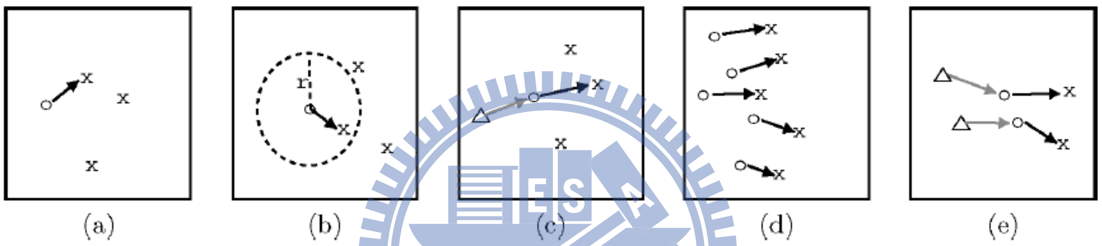


Figure 2.2: Motion constraints [13]. (a) Proximity. (b) Maximum velocity. (c) Small velocity-change. (d) Common motion. (e) Rigidity constraint.

Kernel refers to the object shape and appearance, and kernel tracking is typically performed by computing the motion of the object, which is represented by a primitive object region and generally in the form of parametric motion or the dense flow field computed in subsequent frames. The major differences among kernel tracking methods are the appearance representation used, the number of objects tracked, and the method used to estimate the object motion. For instance, the mean-shift tracking method [31] uses templates and density-based appearance models, while the SVM tracker [40] tracks objects with multiview appearance models.

Objects may have complex shapes. Humans, for example, have head, arms, and legs, and cannot be well described by simple geometric shapes. The aim of silhouette-based methods is to provide an accurate shape description, and to find the object region in each frame through an object model generated according to the previous frames. One category of the silhouette-based methods is shape matching [44-46], which can be performed similar to tracking based on template matching where an object silhouette and its corresponding model is searched in the current frame. The search is invoked by computing the similarity between the object and the model generated from the hypothesized object silhouette according to the previous frame. The other category of the silhouette-based methods is contour tracking [41-43], which iteratively evolve an initial contour in the previous frame to its new position in the current frame. Tracking by evolving a contour can be performed with either state space models which model the contour shape and motion or direct evolution through minimizing the contour energy using direct minimization techniques such as gradient descent.

2.2 Applications in Basketball Video

As discussed in Chapter 1, basketball video analysis is not a common field due to several difficulties and limitations. Fortunately, there are more and more new methods proposed that help us overcome those obstacles and make basketball video analysis much more practicable. We are going to introduce some recent researches on basketball video analysis related to our work.

At first, we would like to introduce the work of Chen et el. [7]. Their research has several notable contributions. First of all, they modify the shot classification algorithm to basketball videos. Basketball shots can be classified into three types: court shots, medium shot, and close-up shots or out-of-court shots. A *court shot* displays a global view of the court. A *medium shot* focuses on an individual player, who is usually the ball handler. A *close-up shot* shows the above-waist view of players, and an *out-of-court shot* presents spectators, coaches, or other places out of the court. Figure 2.3 shows examples of different shot types in a basketball game. Obviously, court shot is the type that contains most information on the court and should be retrieved.

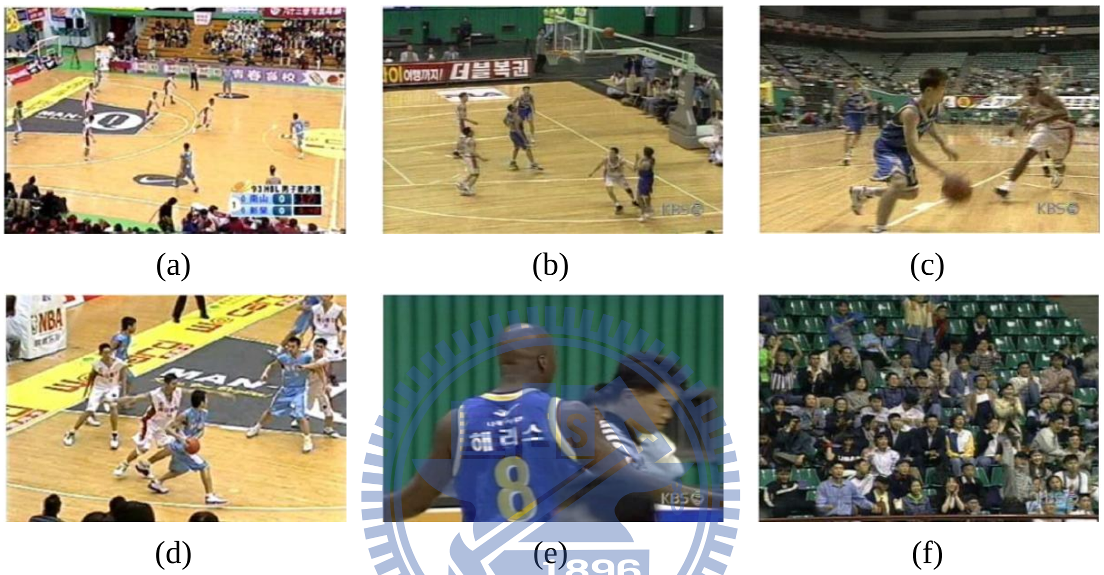


Figure 2.3: Examples of shot types in a basketball game [7]. (a) Court shot. (b) Court shot. (c) Medium shot. (d) Medium shot. (e) Close-up shot. (f) Out-of-court shot.

They divide frames into nine regions by employing Golden Section spatial composition rule as Figure 2.4 shows, and count the number of pixels of the floor color in each region to distinguish shot types. Second, they propose a new method to obtain vertical information in order to form a nonsingular 3D-to-2D transformation. In addition to the typical court lines (2D), they extract the top-border of the backboard (3D) by scanning the baseline from the vanishing point. Figure 2.5 demonstrates the method and Figure 2.6 illustrates the result. Last but not least, they reconstruct 3D information from single view 2D video sequences. With the reconstructed 3D information, they provide a trajectory-based high-level basketball video analysis as well. The 3D ball trajectories facilitate automatic collection of game statistics about shooting locations, from which people can learn the shooting tendency of an individual player, or even a whole team. Figure 2.7 shows some experimental results. In each image in Figure 2.7, blue circles are the ball positions over frames, green circle represents the estimated shooting location, and the red squares show the movements of corresponding points due to the camera motion.

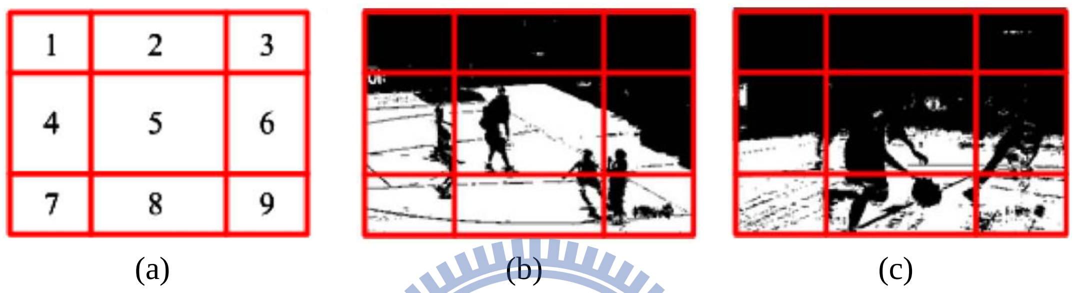


Figure 2.4: Example of Golden Section spatial composition [7]. (a) Frame regions. (b) Court view. (c) Medium view.

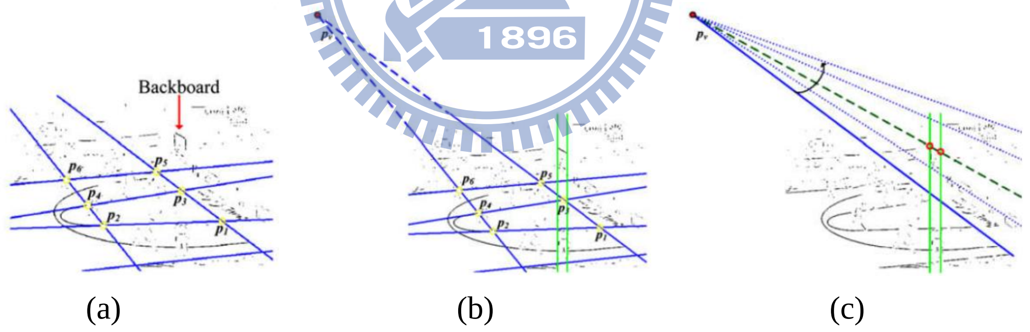


Figure 2.5: Detection of backboard top-border [7]. (a) Detected court lines. (b) Computing vanishing point. (c) Searching backboard top-border.

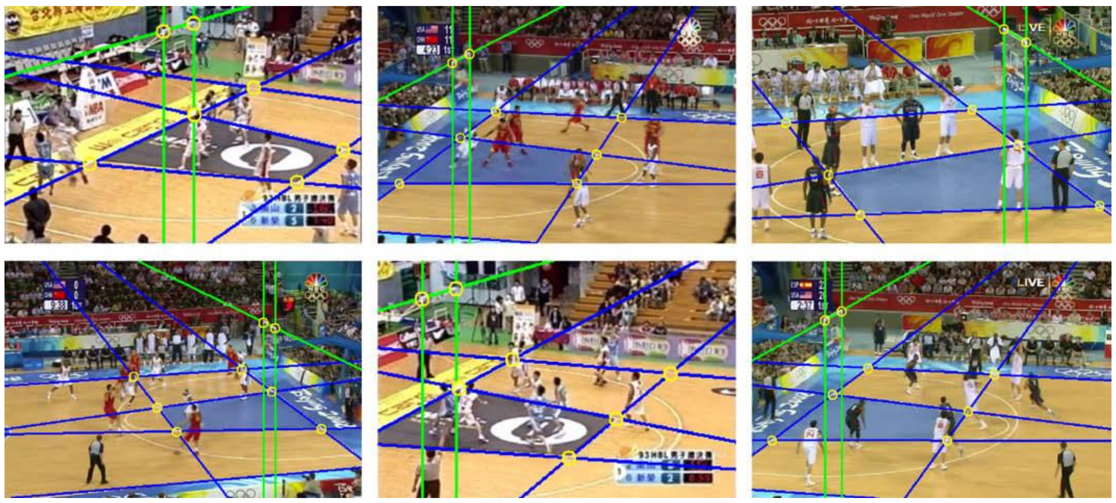


Figure 2.6: Detection of court lines and corresponding points [7].

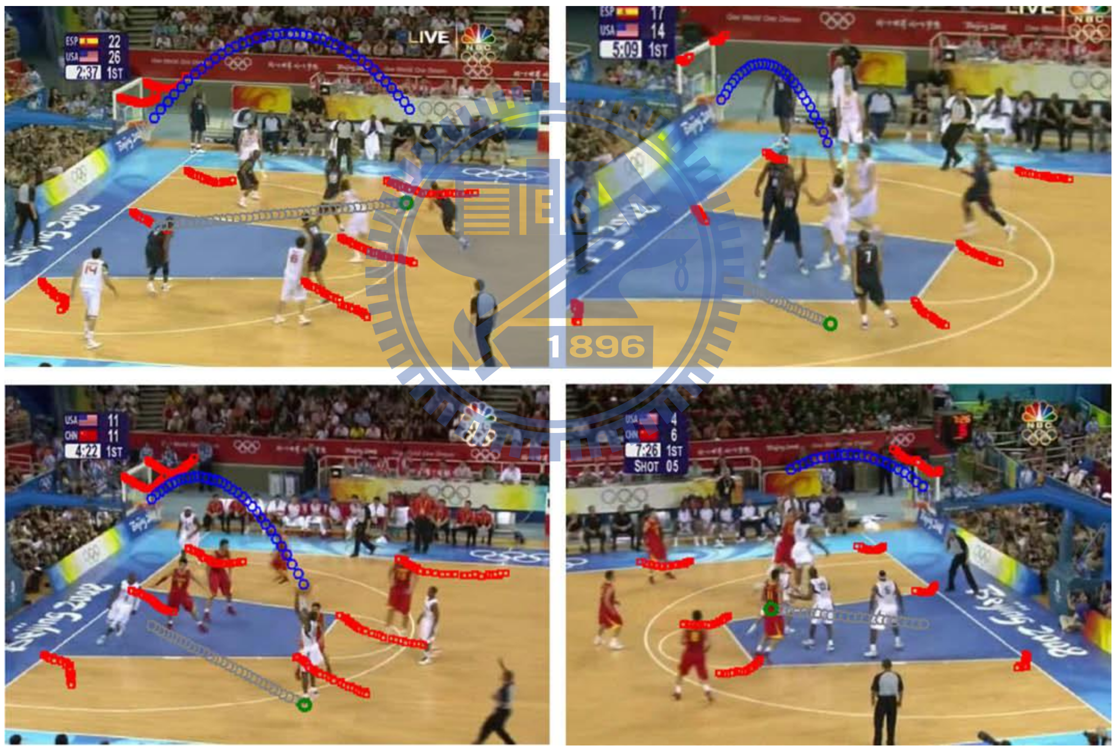


Figure 2.7: Demonstration of shooting location estimation [7].

1896

Besides, we highly praise the work of Chang et al. [1, 50] not only for their contribution to basketball video analysis but also for their novel research on basketball tactics. They propose a method that can gracefully extract players on the court, which vastly improves the performance of object tracking in basketball videos.

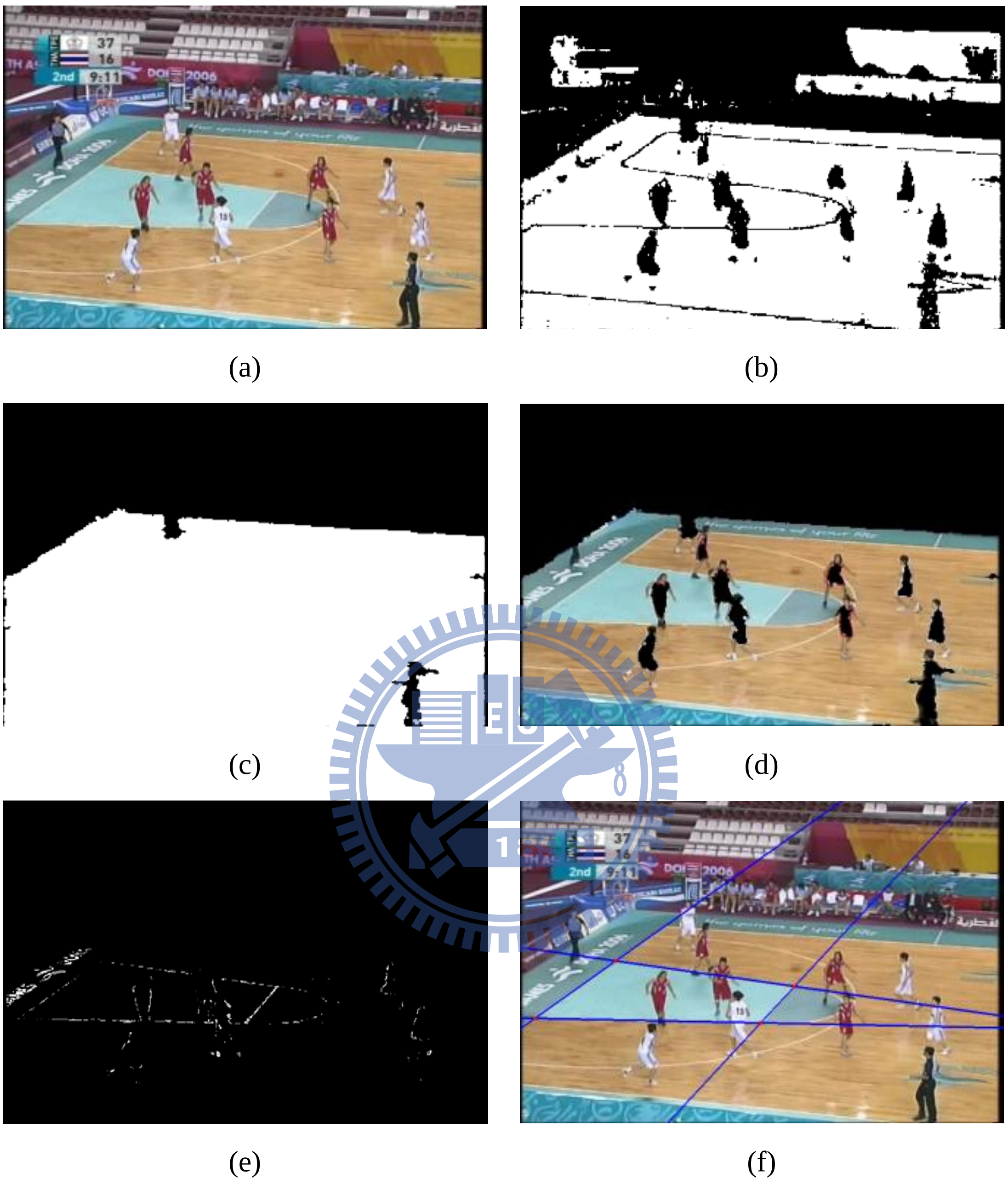


Figure 2.8: Example of the procedure [1]. (a) Original Frame. (b) Dominant color map. (c) Court mask. (d) Removing foreground objects. (e) White pixel detection. (f) Camera calibration.

At first, dominant (floor) color is obtained and a dominant color map is generated. The court region can then be shown through largest connected component analysis of the dominant color map. By utilizing this, foreground objects (player candidates) are extracted. This can also be an additional constraint to white line pixels for the sake of camera calibration since court lines are only located within the court region. Figure 2.8 illustrates the procedure and the result of camera calibration. Next, using color information and any clustering algorithm, foreground region is separated into two clusters representing the jersey colors of the two teams. That is, players of the two teams are recognized. Most important of all, they step into a further field of tactic analysis. Their system informs the user when the distribution of players satisfies the preset rules of the wide-open event. Although their system does not explicitly imply what tactic has been executed, the user can infer the tactic from how the wide-open event occurs. This inspires us to design a system that identifies tactics executed in basketball games and keeps the patterns in order for users to learn basketball tactics. Figure 2.9 demonstrates results of the wide-open warning system.

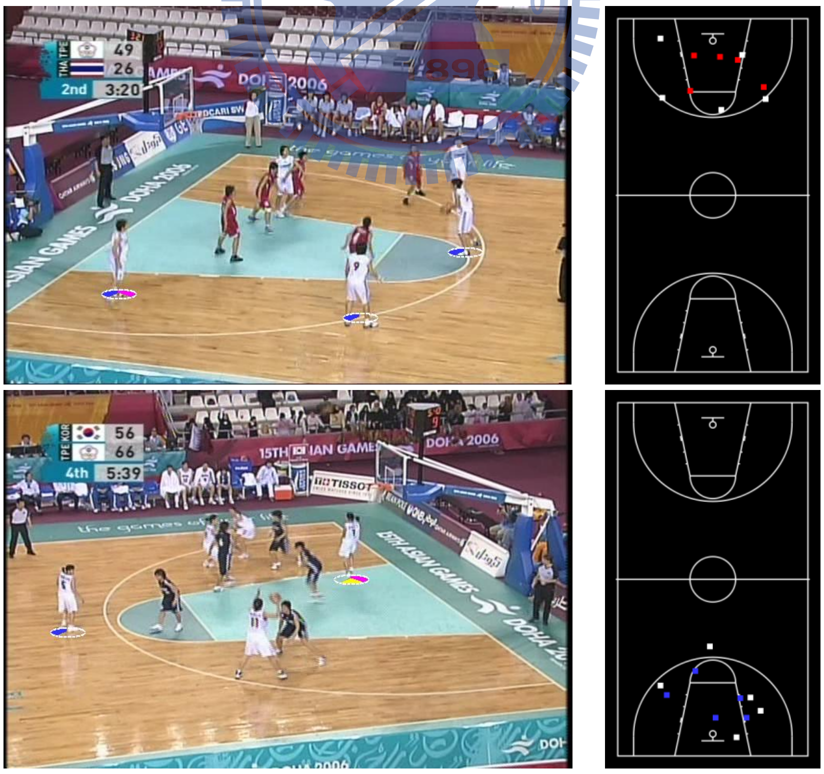


Figure 2.9: Sample results of wide-open warning [1].

Chapter 3. Proposed System Architecture

This chapter describes the details of our proposed system. First of all, we will give an overview in Section 3.1. In Section 3.2, pre-processing is described. Next, we explain our proposed scheme of player tracking during the game in Section 3.3. At last, we will introduce our algorithm for tactic detection and classification in Section 3.4. Note that the video clips we are using are manually segmented by possessions instead of a whole game because our main purpose is to analyze the tactics executed in possessions and automatic possession distinction is not our focus here. Possession means control of the ball. When one team is on offense, we say the team has the possession. One team loses possession if it makes a shot or the opponent team gets the ball. That is, the period we are interested in is from one team first gets the ball until the team shoots the ball.

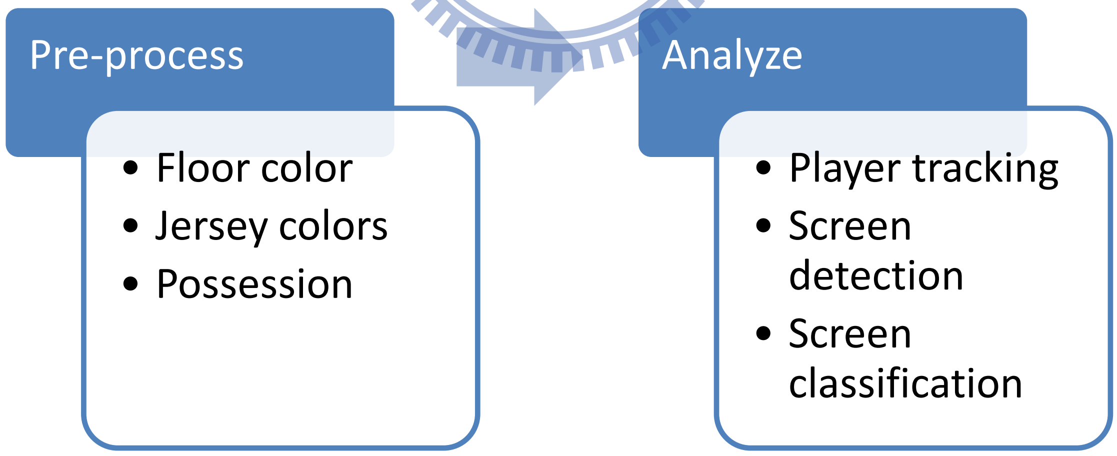


Figure 3.1: System overview.

3.1 Overview

The goal we are going to achieve is to analyze the tactics executed in basketball games. However, there are some obstacles blocking our way to this goal since basketball tactics are complex. The position which a player plays, for instance, is usually considered when setting tactics but difficult for computer to distinguish. Fortunately, we have figured out that screen is a key to all basketball tactics as mentioned in Chapter 1. Hence, screen verification is the core of our system.

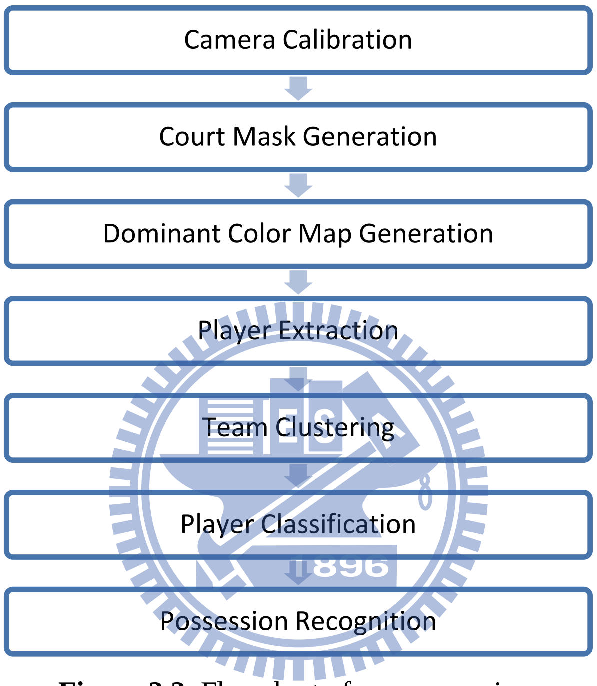


Figure 3.2: Flowchart of pre-processing.

Our system can be divided into two parts: *pre-processing* and *analysis* as shown in Figure 3.1. Pre-processing is performed at the beginning of a video clip in order to gather consistent information in this possession, such as floor color and jersey colors. Since they are invariant during a possession, or even the whole game, we only have to compute them once and for all. With these information gathered in pre-processing, we can avoid computing them each frame and accelerate the computation. As Figure 3.2 illustrates, we first compute the camera calibration and generate a court mask which indicates the court region. Second, we can obtain the floor color by calculating the color histogram and finding the dominant color within the court region. With the floor color, we can perform a background subtraction and extract the foreground objects, that is, the players. Next, we cluster the players into two teams according to their jersey colors. At last, we can realize which team is on offense through the distance between the players and the basket.

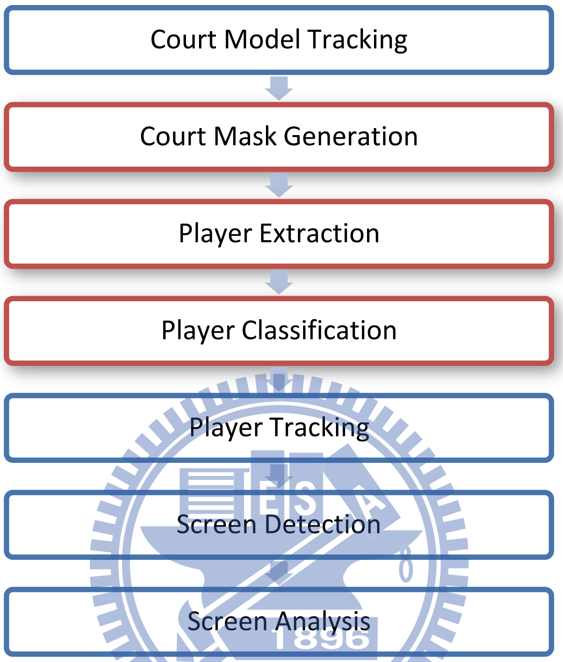


Figure 3.3: Flowchart of content analysis. The modules with shadows have the same functionality as those in the pre-processing phase.

In the following frames, we track the players and also confirm if a screen is set. We have to calculate the camera calibration at first, and then generate a new court mask. Unlike the pre-processing phase, we can obtain current camera calibration from previous frame. Next, we extract the players by the floor color and the jersey colors obtained from the pre-processing. Now we can track the players and detect screens with the positions of players. Once a screen is detected, we retain the state at the moment for the sake of screen type classification. At the end of the possession, we classify the type of the screen set in the possession according to the trajectories of the players. Figure 3.3 shows the flowchart of the analysis phase.

3.2 Pre-Processing

The reason why we perform the pre-processing is that there is some information which will not change during a game, including the floor color and the jersey colors of the two teams. If we repeatedly calculate the information in each frame and just acquire the same result, it is nothing more than an impediment to efficiency. Therefore, in order to reduce the computational cost, we prefer gather the information once and for all. The pre-processing is summarized in Figure 3.2.

3.2.1 Camera Calibration

Camera calibration describes how objects in the world coordinates are projected onto the image coordinates. Since sport courts can be assumed to be planar, camera calibration defines a plane-to-plane mapping (a homography) $\mathbf{H}$ from a position $\mathbf{p}$ in the world coordinates to the image coordinates $\mathbf{p}'$. Writing positions as homogeneous coordinates $\mathbf{p} = (x, y, 1)^{\mathrm{T}}$ and $\mathbf{p}' = (u, v, 1)^{\mathrm{T}}$, the transformation $\mathbf{Hp} = \mathbf{p}'$ is defined in equation (1).

$$\begin{pmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{1}$$

Camera calibration plays an important role in our system since we do most works under the real-world coordinates. The way we obtain the camera parameters is based on the court lines in the frame. Hence, we first have to detect all white pixels in the frame, which belong to the court lines. Second, we find the possible line candidates passing through those white pixels using Hough transform. Next, we filter some unreasonable line candidates out and fit the remaining for the real court lines. Finally, we can obtain the camera parameters through the mapping between the intersection points of the line candidates and those of the court lines.

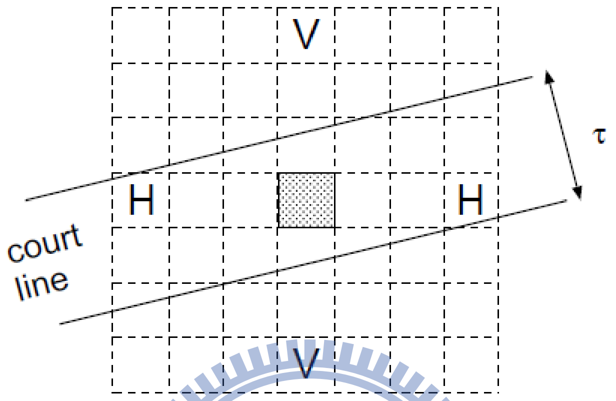


Figure 3.4: Schematic, magnified view of part of an input image containing a court line [2].

1896

3.2.1.1 White Pixel Detection

The court lines are generally painted with white color. Accordingly, the first filter is to confirm if the value of the R, G, B channels of a pixel are above a threshold σ_l to guarantee the pixel is white since the (R, G, B) value of a white pixel is (255, 255, 255). Unfortunately, court lines are usually not the only white objects in a frame and they will influence the line extraction seriously. Hence, other constraints should be applied to the white pixels. Assuming that court lines are not wider than τ pixels in the frame, we verify if the brightness at a distance of τ pixels from four neighbors of the candidate pixel is considerably darker than the candidate pixel as shown in Figure 3.4. Only if they are, the candidate pixel is classified as a white pixel. We can formulate it as equation (2) [2].

$$l(x,y) = \begin{cases} 1, g(x,y) - g(x-\tau,y) > \sigma_d \wedge g(x,y) - g(x+\tau,y) > \sigma_d \\ 1, g(x,y) - g(x,y-\tau) > \sigma_d \wedge g(x,y) - g(x,y+\tau) > \sigma_d \\ 0, \text{else} \end{cases} \tag{2}$$

where $l(x,y)$ indicates if a pixel at position (x,y) is a white pixel ($l(x,y) = 1$) or not ($l(x,y) = 0$), $g(x,y)$ is the luminance of a pixel at position (x,y), and σ_d is the luminance difference threshold. In equation (2), the first line corresponds to the test if darker pixels can be found at some horizontal distance, assuming that the court line is mostly vertical. The second line performs the analogous test in the vertical direction, assuming that the court line is almost horizontal.

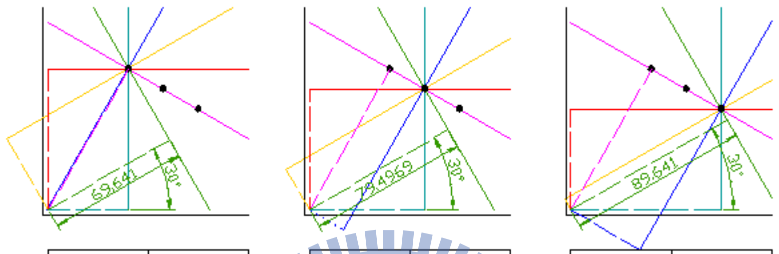
Sometimes the white pixels in textured areas may pass the above white line test, such as small white letters in advertisement logos, spectators dressed in white clothes, or white areas in the stadium. Therefore, we apply an additional line-structure constraint to eliminate those white pixels in the textured areas by observing the two eigenvalues of the structure matrix **S** which is computed over a small window of size $(2b+1)$ around each candidate pixel (p_x, p_y) and defined by equation (3) [10].

$$\mathbf{S} = \sum_{x=p_x-b}^{p_x+b} \sum_{y=p_y-b}^{p_y+b} \nabla g(x,y) \cdot \left(\nabla g(x,y)\right)^{\mathrm{T}} \tag{3}$$

Depending on the two eigenvalues of the matrix **S**, called λ_1 and λ_2 ($\lambda_1 \geq \lambda_2$), the area can be classified into *textured* (both λ_1 and λ_2 are large), *linear* ($\lambda_1 \gg \lambda_2$), and *flat* (both λ_1 and λ_2 are small). On the straight court lines, the *linear* case will apply to retain the white pixels only if $\lambda_1 > \alpha\lambda_2$. We find that when $\alpha = 4$, most linear cases can be recognized.

3.2.1.2 Hough Line Extraction

In order to extract the court lines, we perform the standard Hough transform on the detected white pixels. The parameter space (θ, d) is used to represent a line, where θ is the angle between the line normal and the horizontal axis, and d is the distance between the line and the origin.



Angle	Dist.
0	40
30	69.6
60	81.2
90	70
120	40.6
150	0.4

Angle	Dist.
0	57.1
30	79.5
60	80.5
90	60
120	23.4
150	-19.5

Angle	Dist.
0	74.6
30	89.6
60	80.6
90	50
120	6.0
150	-39.6

Figure 3.5: Hough transform diagram.

Figure 3.5 demonstrates how the Hough transform searches lines. Given three points and we want to find a line passing through them. For each point, a number of lines at different angles are plotted through it. In this example, we plot lines at an interval of 30 degrees. For each plotted line, we compute its distance to the origin and obtain an angle-distance pair representing this line. The results are shown in the tables in Figure 3.5, and the corresponding accumulator matrix is shown in Table 3.1. We can figure out that the parameter set (Angle, Distance) = (60, 81) appears most frequently (three times). Thus, it is the line that we are looking for. Now come back to our problem that we want to extract court line candidates from those detected white pixels. Similarly, we construct an accumulator matrix for all (θ, d) and sample the accumulator matrix at a resolution of one degree for θ and one pixel for

d. By extracting the local maxima in the accumulator matrix, we can determine the line candidates.

Table 3.1: Corresponding accumulator matrix to Figure 3.5.

Angle\Dist.	-40	-20	0	6	23	40	41	50	57	60	70	75	80	81	90
0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0
30	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1
60	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0
90	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0
120	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0
150	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0

In addition, to obtain more precise line parameters, we refine them by minimizing the distance between line pixel candidates and their nearest hough lines. First, we re-parameterize a line obtained from Hough transform by its normal $\mathbf{n} = (n_x, n_y)^{\mathrm{T}}$ with $\|\mathbf{n}\| = 1$ and the distance to the origin d. With the parameters, the distance between a point with homogeneous coordinates in image space $\mathbf{p} = (x, y, 1)^{\mathrm{T}}$ and a line can be calculated by the dot product $(n_x, n_y, -d) \cdot \mathbf{p}$. Next, we define a set L of court line pixels that are close to the line as equation (4) [2].

$$L = \{\mathbf{p} = (x, y, 1)^{\mathrm{T}} \mid l(x, y) = 1 \wedge \|(n_x, n_y, -d) \cdot \mathbf{p}\| < \sigma_r\} \qquad (4)$$

where σ_r is the largest distance constraint in order to discard line pixel candidates far away from any hough line. Since the pixels in this set are supposed to be on the same court line and we assume the refined line equation to be $x \cdot m_x + y \cdot m_y = 1$, we form an equation system and then solve it in the least squares sense as shown in equation (5).

$$\begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_{|L|} & y_{|L|} \end{pmatrix} \begin{pmatrix} m_x \\ m_y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \tag{5}$$

Finally, the refined parameters are computed by $d = 1 \,/ \sqrt{{m_x}^2 + {m_y}^2}$, $n_x = m_x d$, $n_y = m_y d$ since the slope of the line is $-m_x \,/\, m_y$, and the slope of the line normal is $m_y \,/\, m_x$.

3.2.1.3 Court Model Fitting

A court model consists of the lines that are drawn onto the ground to define the playfield geometry. Basketball court model is illustrated in Figure 3.6 and the dimensions are shown in Table 3.2.

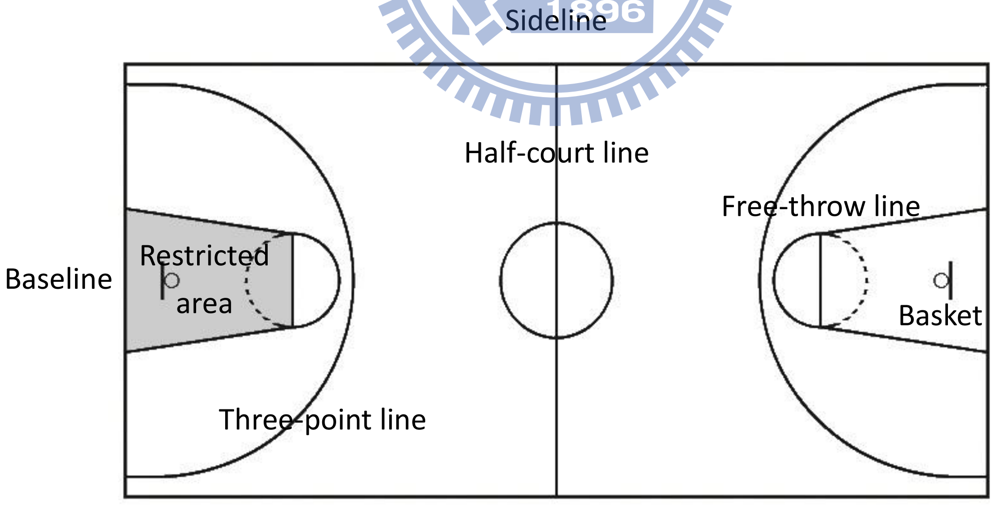


Figure 3.6: Basketball court model.

Table 3.2: Basketball court dimensions.

Area		Dimension (m)
Court length (sideline length)		28
Court width (baseline length)		15
3-point line distance from the basket		6.25
Free-throw line distance from the baseline		5.8
Basket distance from the baseline		1.2
Restricted area width	Free-throw line side	3.6
	Baseline side	5

The camera calibration describes how those lines are projected from the world coordinates onto the image coordinates. Therefore, in order to define the mapping, the correspondence between a previously extracted hough line and the court line in court model must be found. An algorithm has been proposed to find the line correspondence [2] and performs well in several kinds of sport videos such as tennis, volleyball and soccer. They regard the lines determined by extracting the local maxima in the accumulator matrix (mentioned in Section 3.2.1.2) that are above a threshold σ_h as court line candidates. The line candidates are then classified as two sets: one contains the horizontal lines and the other consists of the vertical lines. Next, they sort the line candidates according to their distances to the image boundary, and can search for the correspondence between the candidate lines and the model lines. Nevertheless, when applying to basketball videos, we find that the performance is not good as we expected. The major problem is: how to determine the value of σ_h? The right column of Figure 3.7 shows some results of typical line extraction method with different σ_h values. When the σ_h value is small, there are many unreasonable lines passing the test and viewed as court line candidates which will disturb the line correspondence. On the other hand, when the σ_h value is large, we are not able to obtain sufficient lines to solve camera parameters. Most important of all, whatever threshold we set, the free-throw line is always filtered out because it is short. However, the free-throw line is not negligible since all the corresponding points may locate on the baseline and it will lead to a singular solution to the camera calibration without the free-throw line.

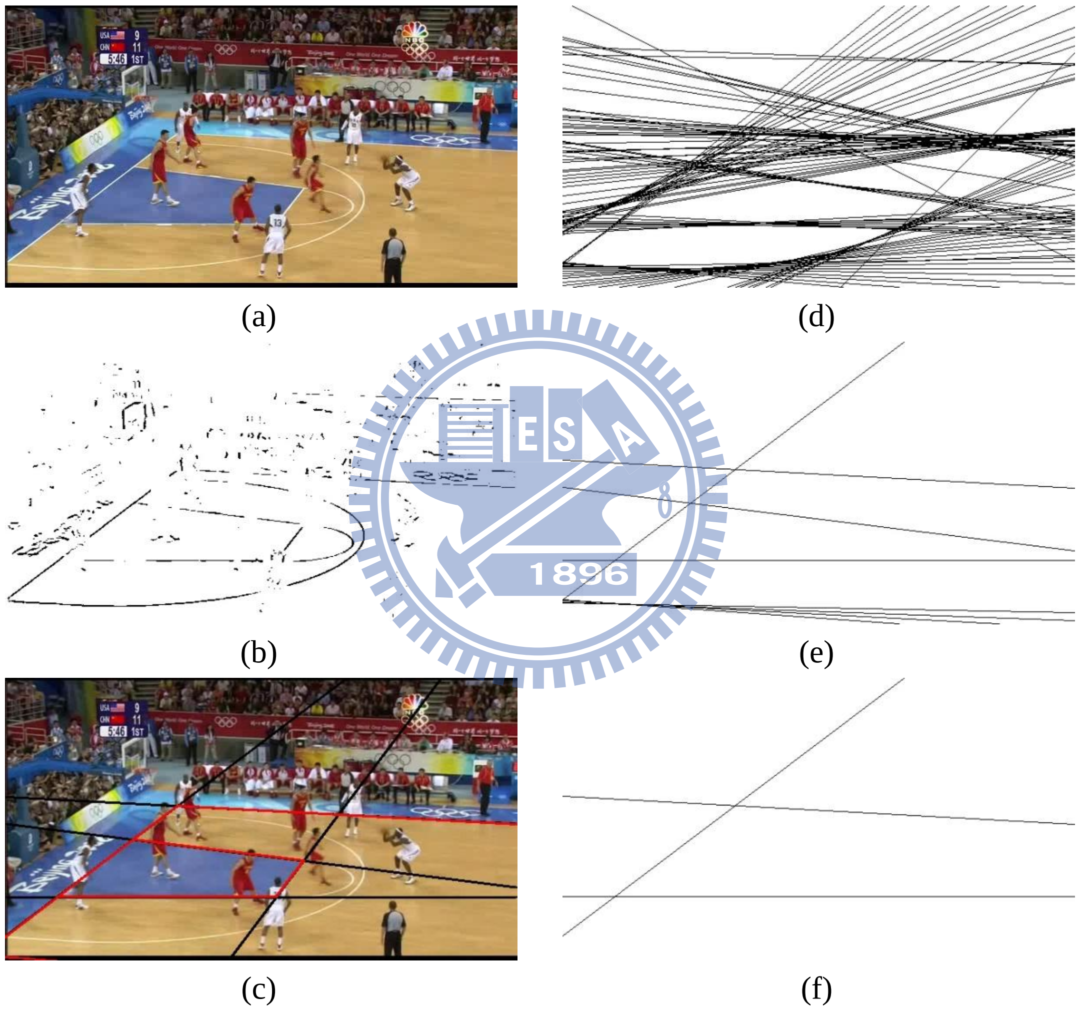
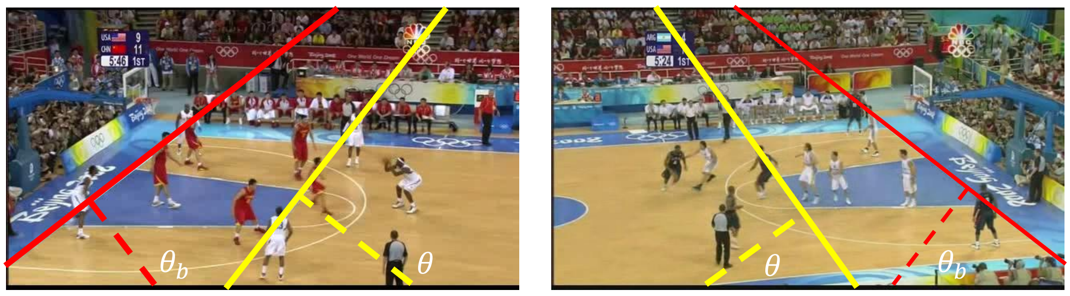


Figure 3.7: Sample results of line extraction. (a) Original frame. (b) Detected white pixels. (c) Result using our method. The right column shows some results using typical method with different thresholds σ_h of (d) 50 (e) 100 (f) 150.

To overcome such a difficulty, we propose a new method to find the line correspondence in basketball video. We do not sample the entire accumulator matrix; instead, we search each line within a specific range in order to gather all the necessary lines. It is an experiential method, and the searching ranges are determined through our observation and knowledge of basketball video.



(a) (b)

Figure 3.8: Examples of basketball video frames. Solid red lines are baselines and solid yellow lines are free-throw lines, and dotted lines are their normals respectively. (a) Left court. (b) Right court.

ES A 1896

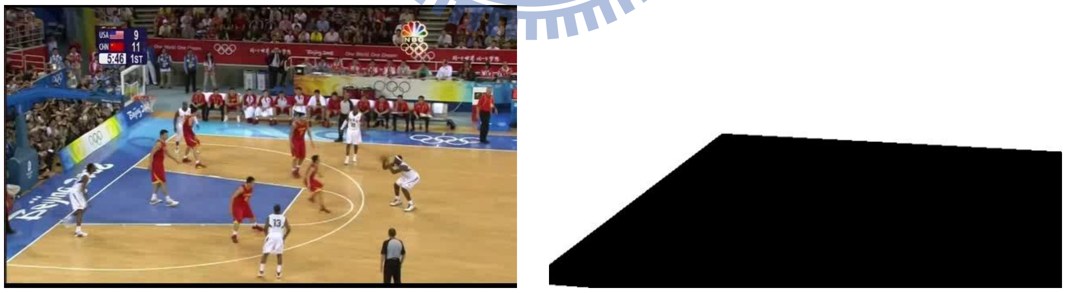
Our main purpose is to discard noise white pixels outside the court region and extract correct court lines. The court region is determined by sideline and baseline. Through Figure 3.8, we can realize that sideline and baseline are the longest horizontal and vertical lines in the frame respectively. Hence, our first step is to find the longest horizontal and vertical lines. For the longest vertical line, we extract the local maximum in the accumulator matrix within the range of [0, 80] and [100, 180] degrees. Remember that the parameters in Hough space are the distance between a line and the origin, and the angle between the line normal and the horizontal axis. That is, this ignores lines whose angle between the horizontal axis is within the range of [-10, 10] degrees, namely, those almost horizontal lines. We obtain the longest vertical line by eliminating horizontal lines instead of directly finding vertical lines since it may not look that perpendicular on screen. Furthermore, the angle of baseline also helps us distinguish whether it is the left court or right (see solid red lines Figure 3.8). On the other hand, when extracting the longest horizontal line, we just set the searching range to [80, 100] degrees since horizontal lines do not change significantly on screen. With the longest vertical and horizontal line, that is, baseline and sideline respectively, we filter those white pixels out which are outside the region bounded by the two lines, and reconstruct the accumulator matrix from the remaining white pixels. Next, we extract the longest two horizontal lines as edges of the restricted area. Top edge and bottom edge are then distinguished by angles of the two lines. Through Figure 3.8 we can find that bottom edge is always more horizontal than top edge. At last, we have to find free-throw line. Please view Figure 3.8 again. We mark the baseline with the solid red line and the free-through line with the solid yellow line, and the dotted lines are their normals respectively. We can clearly figure out that although they are both vertical lines in court model, free-throw line always looks more perpendicular than baseline whichever side of court is on screen because the camera is usually set at the center of the court. Thus, we set the searching range as [0, θ_b] degrees for right court and [θ_b, 180] degrees for left court in order to extract free-throw line. Here, θ_b is the angle between baseline normal and the horizontal axis. Since the remaining white pixels are guaranteed to be within the court region, we can recognize those extracted lines as correct court lines. In this way, we extract lines and find the correspondence at the same time since we know exactly which line we are looking for. Finally, we compute the intersection points and solve the equation system defined as equation (6) which is rewritten from equation (1).

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x'_2x_2 & -x'_2y_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -y'_2x_2 & -y'_2y_2 \\ & & & & \vdots & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_nx_n & -x'_ny_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_nx_n & -y'_ny_n \end{pmatrix} \begin{pmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \end{pmatrix} = \begin{pmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ \vdots \\ x'_n \\ y'_n \end{pmatrix} \tag{6}$$

Note that this makes use of the normalization $h_{22} = 1$. There are eight variables $h_{00}, h_{01}, \dots, h_{21}$ so we need at least four points ($n \geq 4$) in order to form more than eight equations. Here we use baseline, free-throw line and two edges of restricted area to solve the equation system. Figure 3.7 (c) illustrates the result using our method.

3.2.2 Court Mask Generation

In basketball video, most of important information is inside the court region. In other words, court is our region of interest. In order to filter out noise and keep significant information, we need a mask to indicate the court region, that is, the court mask. With the previously computed camera calibration, we can project pixels from image coordinates back to world coordinates and confirm whether they are located in the court. Figure 3.9 shows a sample result of the court mask.



(a) (b)

Figure 3.9: Court mask. (a) Original frame. (b) Corresponding court mask.

3.2.3 Dominant Color Map Generation

In order to extract players, we have tried several methods [18, 21, 25]. Nevertheless, none of them performs well in basketball videos. The most serious obstacle is the camera motion. For example, redundant moving pixels resulting from the camera motion generate huge amount of noise when performing the frame difference. For another example, the camera motion prevents us from obtaining a consistent background image and extracting real moving objects. Therefore, a new method is proposed to extract the players on the court by detecting objects with different colors from the floor [1, 50].

The way we obtain the floor color is to find the dominant color within the court region using the previously generated court mask. First of all, we calculate the color histogram. Since it has been proved in [11] that the performance in the YCbCr space is better than that in the HSI space, we choose the YCbCr space and use the Cb and Cr components to calculate the color histogram. With the color histogram, we next find peaks by the following steps

Step 1: Determine the main peak bin $Peak_1$, that is, the bin with the largest value.

Step 2: Find the connected region around the main peak bin. Only bins with value larger than $\alpha * value(Peak_1)$ are considered.

Step 3: Compute the sum of the connected bins Sum_1 and subtract the connected region from the histogram. That is, we set the values of the bins of the connected region to zero in order not to be considered again in the following iterations.

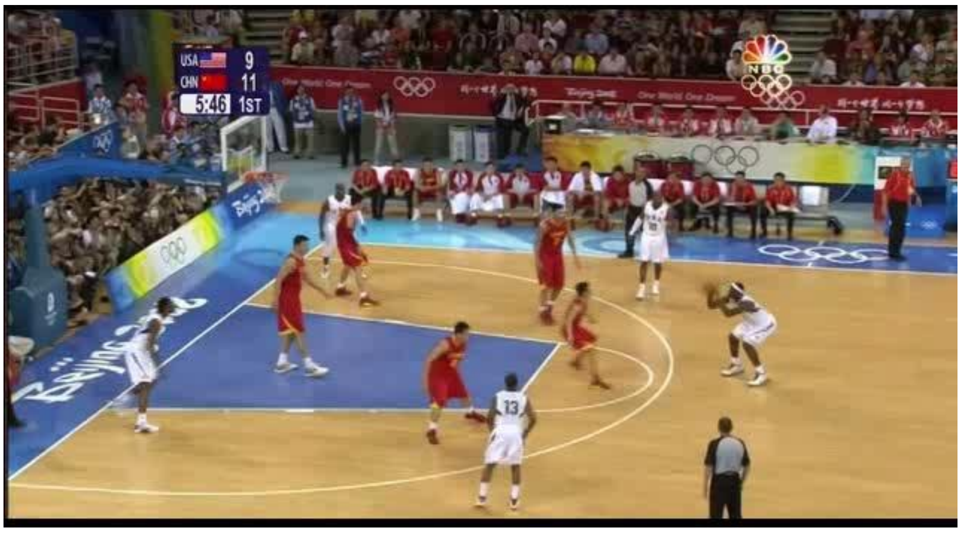
Step 4: Repeat the above steps until there are no bins remaining.

After completing the procedure, we will have several peaks and their sums. Finally, by sorting these peaks according to their sums, we can realize the dominant color. It deserves to be mentioned that in a basketball court, there is a restricted area (see

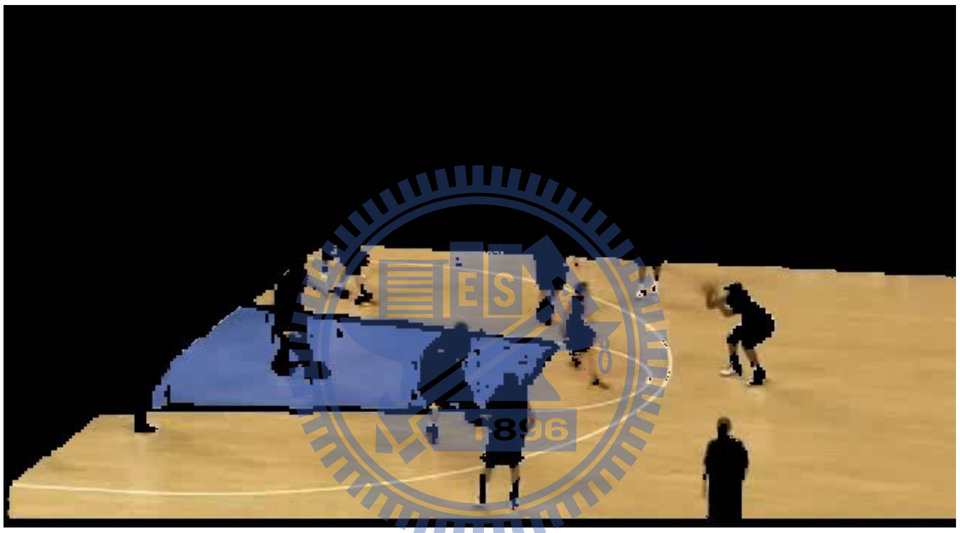
Figure 3.6), which is also called the painted area since it is usually painted with different color from other parts of the court. That is, if we just recognize the largest peak as the floor color, we will miss the restricted area. We propose two ways to solve this problem. One is to regard the largest two peaks as the floor color, and the other is to run the procedure again with another mask indicating the restricted area. Both methods have their pros and cons. The first one takes advantage of the previous result but it fails when there are many players stay in the restricted area. The second one can distinguish the players from the restricted area since it compares the two series of sorted peaks and verifies which peak represents the restricted area. Through our experiment, we prefer the first one because it has good performance and does not require extra computation. Figure 3.10 (b) illustrates a sample result of the dominant color map.

3.2.4 Player Extraction

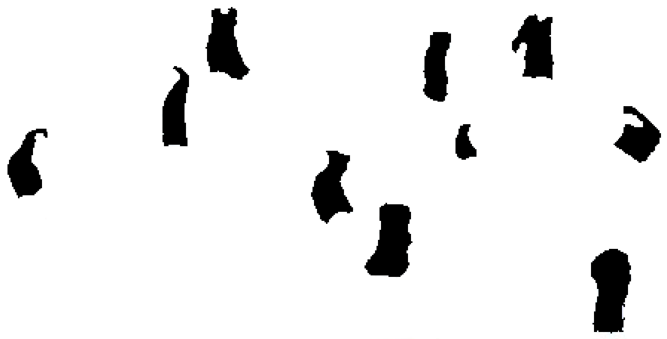
With the court mask and the dominant color map, we can perform a background-subtraction-like method to extract the foreground objects in the court region. If the color of a pixel can be found in the dominant color map, the pixel should be labeled as background; otherwise, it is a foreground pixel. After all pixels are confirmed, we apply morphological operators in order to remove small objects and gaps. Figure 3.10 (c) demonstrates a sample result of extracted foreground objects.



(a)



(b)

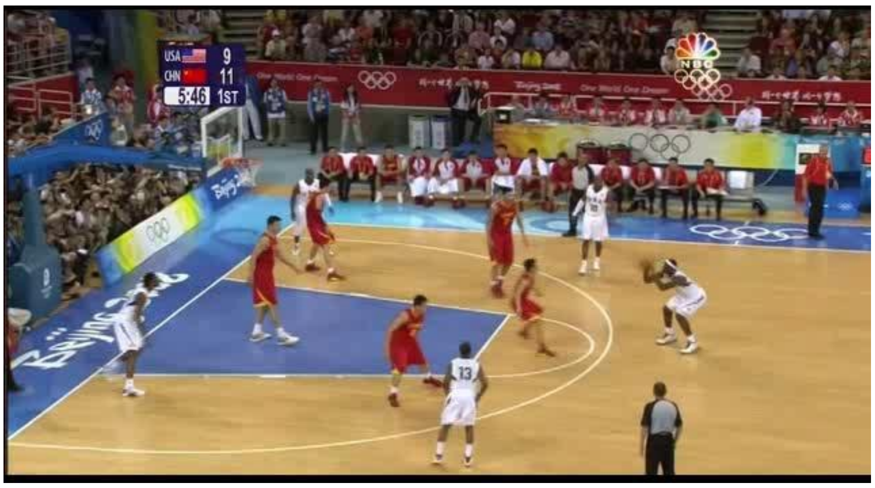


(c)

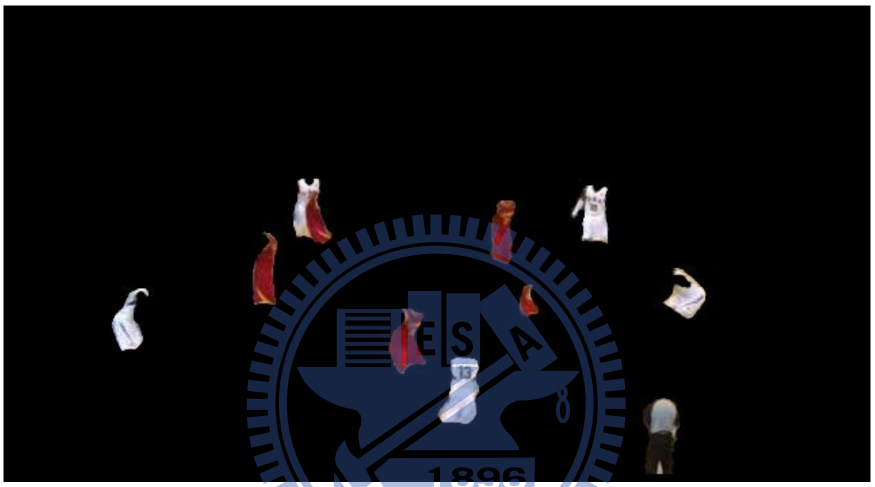
Figure 3.10: Object extraction. (a) Original frame. (b) Dominant color map. (c) Foreground objects.

3.2.5 Team Clustering

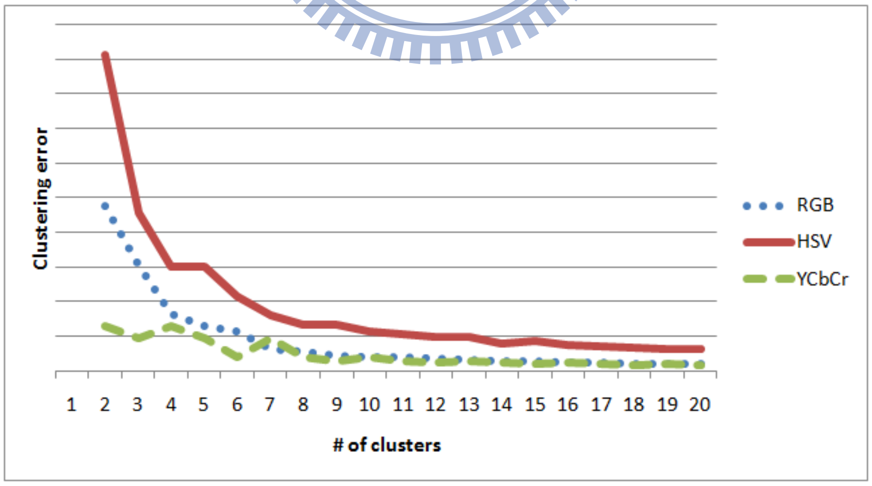
Despite the fact that we have the foreground objects within the court region extracted, we need more information to analyze the content. First of all, we have to distinguish the jersey colors in order to separate the players of the two teams. We use color information and k-means clustering to divide the foreground region into two clusters representing the jersey colors of the two teams. In fact, we cannot have just two clusters since there is some noise in the foreground region, the referees for example, which enormously interferes with the cluster centroids and leads to a miserable result of player classification. Figure 3.11 shows experimental data about the number of clusters and the performances. Generally, the more the clusters, the smaller the total distance between all data points and their corresponding cluster centroids, which can also be regarded as the clustering error. However, the computing time of the k-means clustering is proportional to the number of clusters. We discovered that the clustering error decreases most rapidly when there are six clusters. The clustering errors almost converge when there are more than six clusters. This fact is also adaptive to other video clips through our experiment. Thus, we separate the foreground region into six clusters and view the largest two clusters as the jersey colors of the two teams, and choose the YCbCr space since it performs better than the RGB and HSI space through our experiment.



(a)



(b)



(c)

Figure 3.11: K-means clustering. (a) Original frame. (b) Foreground objects. (c) Experimental data with different color spaces and number of clusters. The horizontal axis means the number of clusters and the vertical axis indicates the clustering error, and different lines represent different color spaces.

3.2.6 Player Classification

Having gathered the jersey colors of the two teams, we are going to classify players of the two teams in this step. At first, we verify the pixels in the foreground region which clusters they belong to by their colors by equation (7) where $Centroid_A$ and $Centroid_B$ are the centroids of the largest two clusters from the Team Clustering step, that is, the jersey colors of the two teams.

$$cluster(x,y) = \begin{cases} Cluster_A, \|color(x,y) - Centroid_A\| < \delta_c \\ Cluster_B, \|color(x,y) - Centroid_B\| < \delta_c \\ None, \text{else} \end{cases} \tag{7}$$

After clustering all foreground pixels, we can generate two maps indicating the players of the two teams as Figure 3.12 illustrates. Since we set a constraint to the minimal distance between a color of a pixel and the cluster centroid which it belongs to, we can remove those non-player objects such as the referees during clustering. Also, we perform morphological operators to remove noise and gaps. At last, we apply object segmentation and obtain all players.

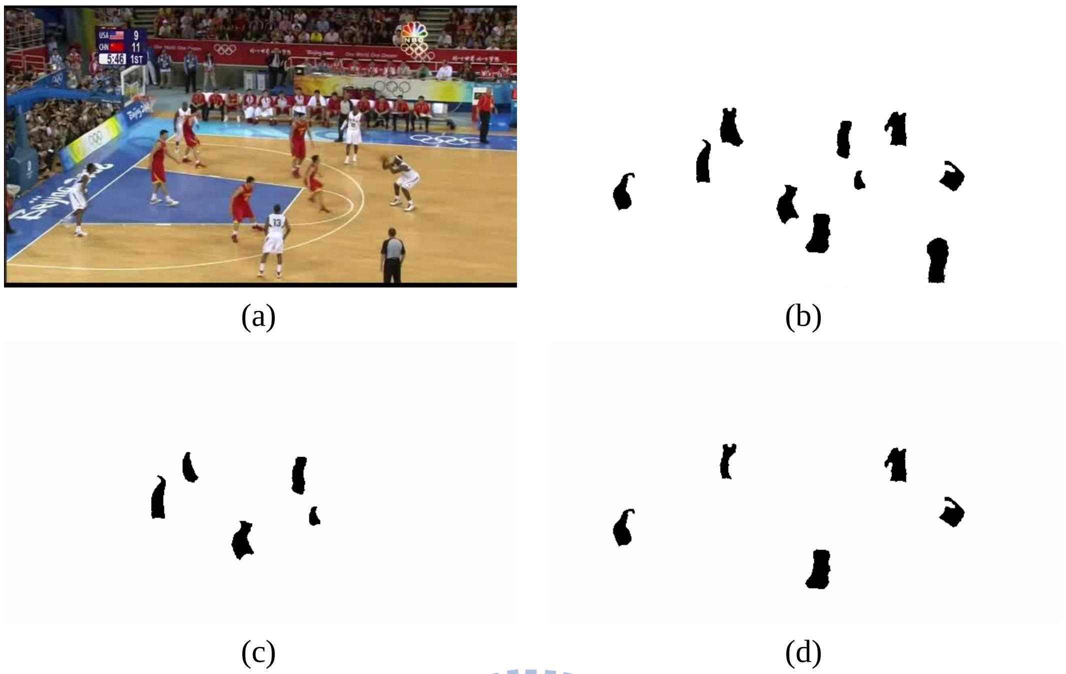


Figure 3.12: Player classification. (a) Original frame. (b) Foreground objects. (c) Players of one team (red jerseys). (d) Players of the other team (white jerseys).

ESA 1896

3.2.7 Possession Recognition

It is important to realize which team has the possession of the ball, or which team is on offense, before tactic analysis. Typically, defenders are expected to stand closer to the basket than the offensive player who he is guarding in basketball games since the purpose of the team on defense is to prevent the opponent team from putting the ball into the basket. Hence, we can make use of this feature to judge which team is on offense. We first project all players back to the real-world court model with the camera calibration. For each team, we compute the average distance between its players and the basket. The team with shorter distance to the basket is recognized as on defense. On the other hand, the team on offense is averagely farther away from the basket.

Algorithm 1: Possession Recognition

Input: positions of players of the two teams, represented by $players_{team_1}$ and $players_{team_2}$, and position of the basket

Output: the team on offense, $team_1$ or $team_2$

```
local $dist[2]$
for $i := 1$ to $2$ do
  $dist[i] := 0$
  for each $player$ in $players_{team_i}$ do
    $dist[i] := dist[i] + dist(player, basket)$
  end for
  $dist[i] := dist[i] \, / \, |players_{team_i}|$
end for
if $dist[1] > dist[2]$ then
  return $team_1$
else
  return $team_2$
end if
```

3.3 Content Analysis

In this section, we are going to explain how we gather information from each frame during the game. With the consistent data from the pre-processing, we can obtain the information we want simply and fast. Figure 3.3 gives a brief view about our analysis mechanism. Remark that the modules in Figure 3.3 with shadows have the same functionality as those in pre-processing. That is, we will perform Court Mask Generation, Player Extraction, and Player Classification in analysis part as well.

3.3.1 Court Model Tracking

In basketball video, camera motion is usually wild in order to focus on the ball.

Camera motion is made up of translation, rotation and zooming. Figure 3.13 illustrates an example of camera motion. Figure 3.13 (a) and (b) are from the same video clip and taken by the same camera. In Figure 3.13 (a), the ball is around the three-point line. On the other hand, a player shoots the ball so that the ball is in the basket in Figure 3.13 (b). It is obvious that camera motion is tremendous between the two frames. Therefore, we have to update camera parameters every frame in order to keep correct camera calibration.

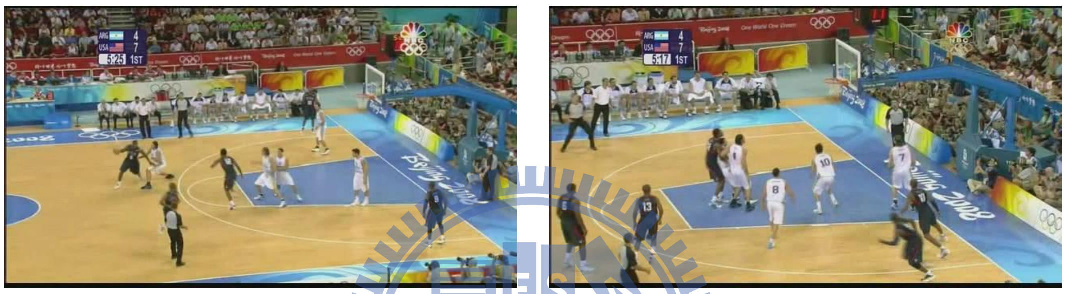

(a) (b)

Figure 3.13: Camera motion in basketball video, rotate and zoom in.

In fact, computing camera calibration is quite a heavy load and it takes too much time to do it every frame. Thus, a court model tracking algorithm has been introduced in [2] in order to save the computing time. The algorithm starts from a prediction to the camera parameters for the next frame. Next, white pixel detection is performed to find line pixels. Those line pixels are then projected to real court model using predicted camera calibration. Finally, camera calibration is refined by minimizing distances between projected line pixels and their closest court lines. Figure 3.14 demonstrates the idea of camera parameter prediction.

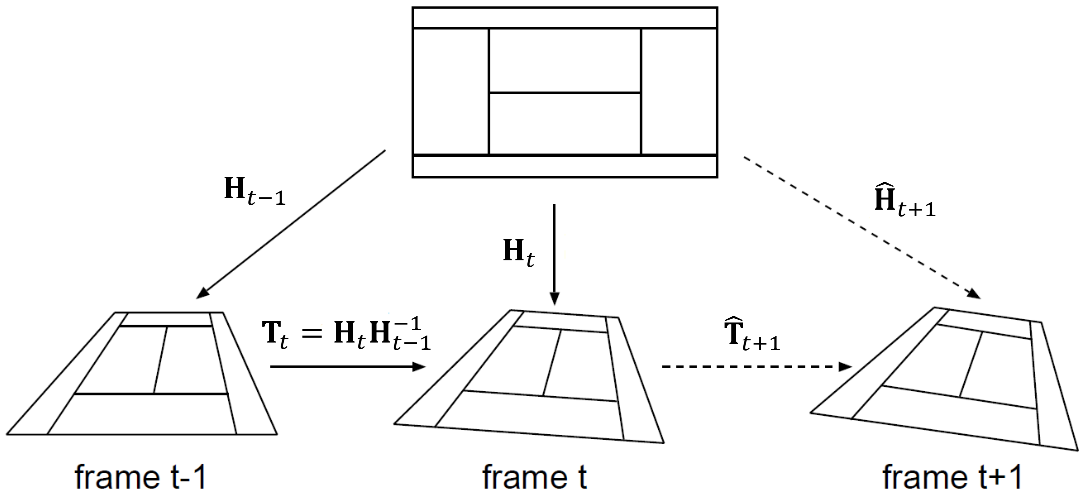


Figure 3.14: Predicting the camera parameters for frame $t+1$ based on the previously computed parameters for frames t and $t-1$ [2].

As Figure 3.14 shows, the camera parameters for frame t is denoted by $\mathbf{H}_t$. First of all, we want to compute the transform matrix $\mathbf{T}_t$ such that $\mathbf{H}_t = \mathbf{T}_t\mathbf{H}_{t-1}$. According to Figure 3.14, $\mathbf{T}_t$ can be easily obtained by $\mathbf{T}_t = \mathbf{H}_t\mathbf{H}_{t-1}^{-1}$. Similarly, once we know the transform matrix $\mathbf{T}_{t+1}$, we can obtain the camera parameters $\mathbf{H}_{t+1}$ for frame $t+1$ from the previous camera parameters $\mathbf{H}_t$ by $\mathbf{H}_{t+1} = \mathbf{T}_{t+1}\mathbf{H}_t$. We suppose that the camera motion is stationary, and the transformation from frame t to $t+1$ is equivalent to that from frame $t-1$ to t. That is, the predicted transform matrix $\hat{\mathbf{T}}_{t+1}$ is assumed to be $\mathbf{T}_t$. Consequently, the predicted camera parameters for frame $t+1$, namely $\hat{\mathbf{H}}_{t+1}$, are computed by $\hat{\mathbf{H}}_{t+1} = \hat{\mathbf{T}}_{t+1}\mathbf{H}_t$, which can be rewritten as equation (8).

$$\hat{\mathbf{H}}_{t+1} = \mathbf{H}_t\mathbf{H}_{t-1}^{-1}\mathbf{H}_t \tag{8}$$

Since the camera motion is usually slight between two adjacent frames, the prediction provides a good initial estimation of the new camera parameters.

Nevertheless, the predicted camera parameters still have to be adapted to the next frame. First, we perform the white pixel detection again to extract the white line pixel candidates as described in Section 3.2.1.1. Note that we have already obtained a reliable initial estimation of the camera parameters and only a narrow neighbor is considered. Therefore, the accuracy of the white pixel detection can be decreased by disabling the line structure constraint for the sake of efficiency. Second, we project all the white pixel candidates back to the real-world court model with the inverse of the predicted camera parameter matrix determined by $\mathbf{M} = \widehat{\mathbf{H}}_{t+1}^{-1}$. Pixels projected to positions too far away from any court line are ignored, and the remaining pixels are grouped with the closest court model line. Our target is to find a matrix minimizing the distance between white pixels $\mathbf{p}_i = (x_i, y_i, 1)^{\mathrm{T}}$ and their corresponding court model lines $\mathbf{l}_i = \left(n_{x;i}, n_{y;i}, -d_i\right)^{\mathrm{T}}$. Here we define an additional operator $h(\cdot)$ which normalizes the homogeneous coordinates such that $h: (x, y, w) \rightarrow (x/w, y/w, 1)$. The projection error D can be then formulated as equation (9) [2].

$$D = \sum_i \left[\mathbf{l}_i{}^{\mathrm{T}} h(\mathbf{M}\mathbf{p}_i)\right]^2 \tag{9}$$

The Levenberg-Marquardt algorithm [48, 49] is used to find the optimized $\mathbf{M}$, denoted by $\mathbf{M}^*$, which minimizes D, and we can obtain the refined camera calibration by $\mathbf{M}^{*-1}$.

Algorithm 2: Court Model Tracking

```
Input: current frame $f_{t+1}$, court model, and previous camera parameters $\mathbf{H}_t$, $\mathbf{H}_{t-1}$
Output: current camera parameters $\mathbf{H}_{t+1}$

// predict
local $\widehat{\mathbf{H}}_{t+1} := \mathbf{H}_t \mathbf{H}_{t-1}^{-1} \mathbf{H}_t$

// refine
procedure $projection\_error(white\_pixels, \mathbf{M})$
  local $D := 0$
  for each $\mathbf{p}$ in $white\_pixels$ do
    local $\mathbf{p}' := h(\mathbf{Mp})$
    local $\mathbf{l} := find\_closest\_court\_line(\mathbf{p}')$
    if $\mathbf{l} = nil$ then
      continue
    end if
    $D := D + (\mathbf{l}^{\mathrm{T}}\mathbf{p}')^2$
  end for
  return $D$
end

local $\mathbf{M} := \widehat{\mathbf{H}}_{t+1}^{-1}$,
local $white\_pixels := white\_pixel\_detection(f_{t+1})$
local $\mathbf{M}^* := levenberg\_marquardt(projection\_error, white\_pixels, \mathbf{M})$
return $\mathbf{M}^{*-1}$
```

3.3.2 Player Tracking

We use the Kalman filter [12] to track players. The Kalman filter is composed of two steps: predict and correct. Figure 3.15 shows the complete operation of Kalman filter. In Figure 3.15, $\hat{a}_t$, z_t, c_t are the system state, measurement, and external control at the moment t respectively, where A, Z, C are the transition matrices of them; Q and R are the noise covariance matrices of the process and the measurement respectively; E_t is the error covariance and K_t is the Kalman gain.

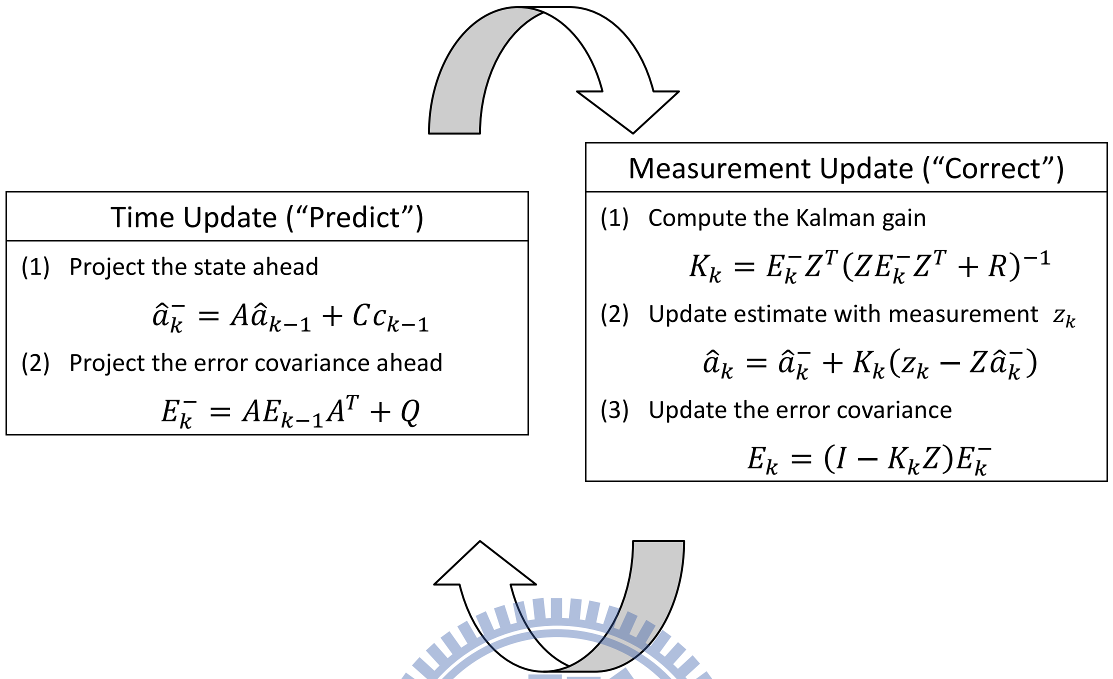

Time Update ("Predict")
(1) Project the state ahead
$\hat{a}_k^- = A\hat{a}_{k-1} + Cc_{k-1}$
(2) Project the error covariance ahead
$E_k^- = AE_{k-1}A^T + Q$
Measurement Update ("Correct")
(1) Compute the Kalman gain
$K_k = E_k^- Z^T (ZE_k^- Z^T + R)^{-1}$
(2) Update estimate with measurement z_k
$\hat{a}_k = \hat{a}_k^- + K_k(z_k - Z\hat{a}_k^-)$
(3) Update the error covariance
$E_k = (I - K_k Z)E_k^-$

Figure 3.15: Complete diagram of Kalman filter [12].

With the prediction of the trackers computed by the Kalman filter, we can choose the nearest candidate as the measurement of a tracker. We also set a limitation that the distance between the predicted state and the chosen measurement should never be larger than a threshold δ_t. If there is no candidate that is close enough to the predicted state, we regard the predicted state as measurement directly. Besides, since all players are expected to stay in the court, once a tracker state is out of the court, we mark it as *missing*. Every time a tracker misses, its search range is increased by a multiplicand since the object may be occluded and so that the tracker misses temporarily, and we hope that the tracker can keep tracking on the object when it shows again. If a tracker consecutively misses for ε_f frames, that is, the tracker is outside the court for too many frames, it is terminated and no longer tracked. After updating all trackers, there are some candidates tracked, and we add new trackers for the untracked candidates.

Algorithm 3: Player Tracking

Input: tracker list $trackers$ and candidate list $candidates$

Output: none

```
// Step 1: update trackers
for each $tracker$ in $trackers$ do
   local $prediction := predict(tracker)$
   local $measurement$
   if $\exists candidate \in candidates: dist(prediction, candidate) < \delta_t$ then
      $measurement := candidate$
      $set\_tracked(candidate)$
   else
      $measurement := prediction$
   end if
   $correct(tracker, measurement)$
   if $is\_out\_of\_court\_bound(measurement)$ then
      $increase\_missing\_count(tracker)$
   else
      $reset\_missing\_count(tracker)$
   end if
   if $missing\_count(tracker) > \varepsilon_f$ then
      $terminate(tracker)$
   end if
end for
// Step 2: create new trackers
for each $candidate$ in $candidates$ do
   if $not\_tracked(candidate)$ then
      $add\_tracker(trackers, candidate)$
   end if
end for
```

3.4 Tactic Analysis Algorithm

Before explaining our tactic analysis algorithm, we introduce the basis of basketball tactics first. As mentioned in Chapter 1, most basketball tactics consist of screens, and we identify tactics by what and where screens are set. That is, if we want to learn what tactics are executed, we have to realize what types of screens are set.

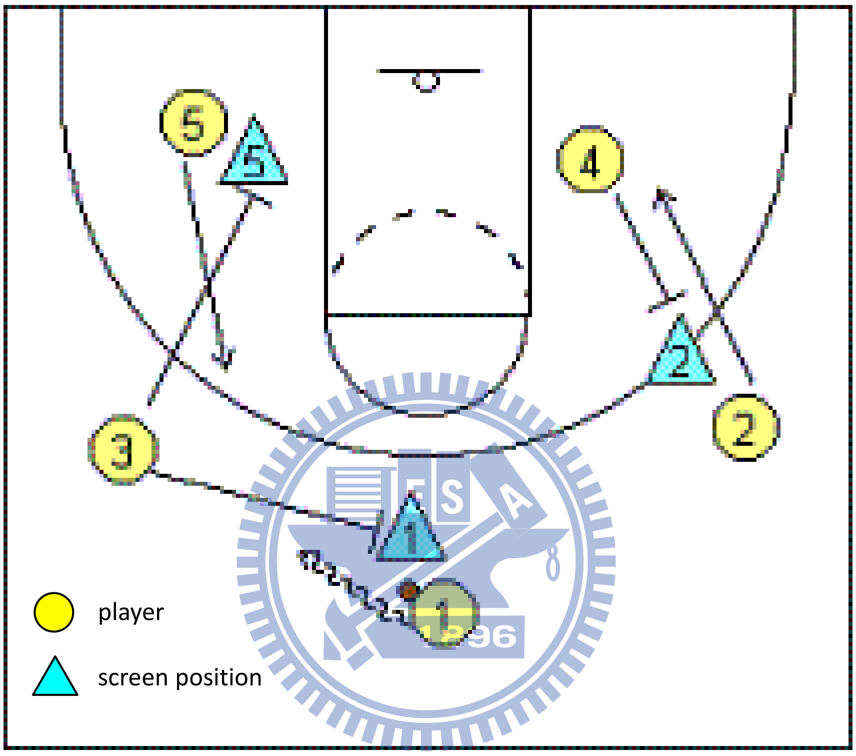


Figure 3.16: A sample basketball tactic.

Generally, screens can be classified as three major types: *front-screen*, *back-screen* and *down-screen*. Figure 3.17, 3.18, and 3.19 illustrate the three screen types respectively by displaying different moments of a screen: the beginnings of the screens are shown in (b), the moments when the screens are being set are illustrated in (c), the ends of the screens are demonstrated in (d), and (a) presents the overall trajectories of the two players involved in the screen. In the three figures, players with yellow circles on their heads are screeners, while players with cyan triangles are their offensive teammates whom the screens are set for.

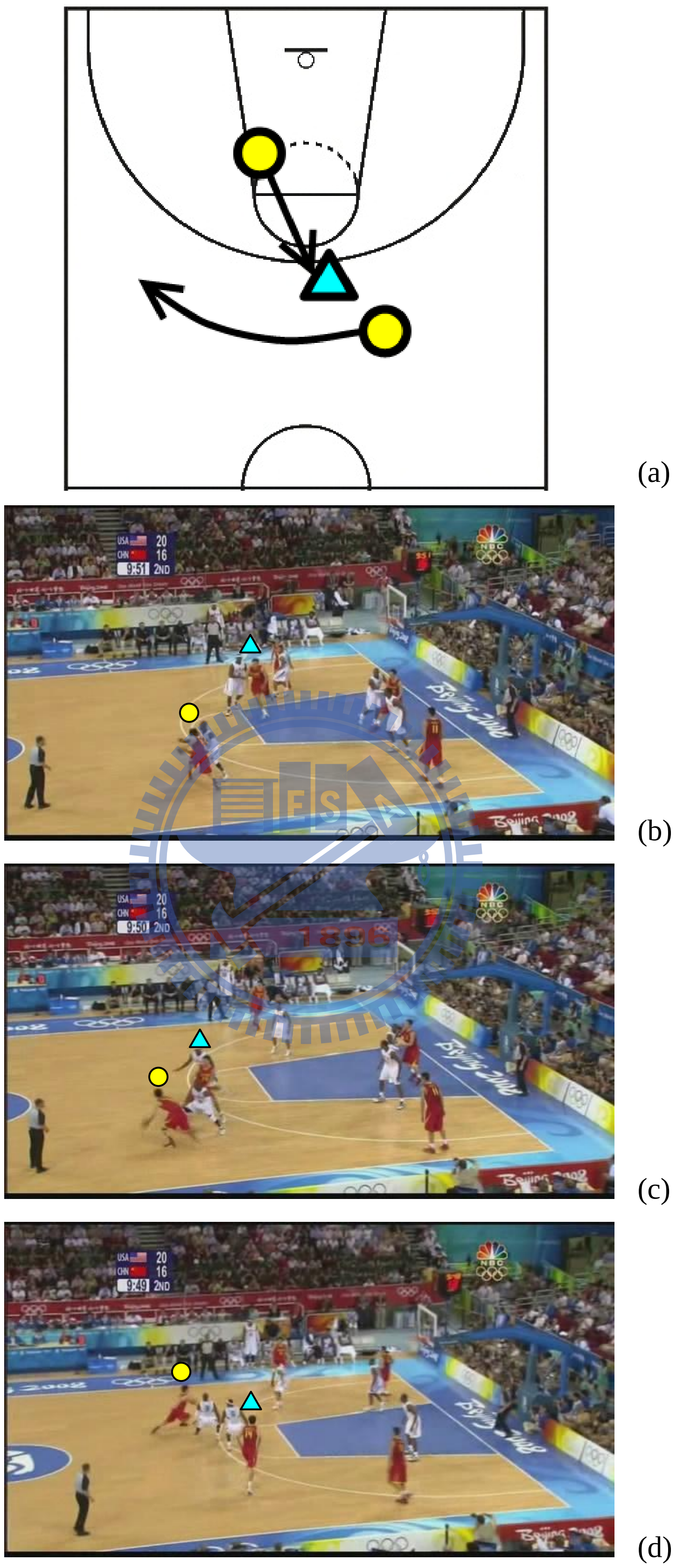


Figure 3.17: Example of front-screen. (a) Trajectories. (b) Before screen. (c) Setting screen. (d) After screen.

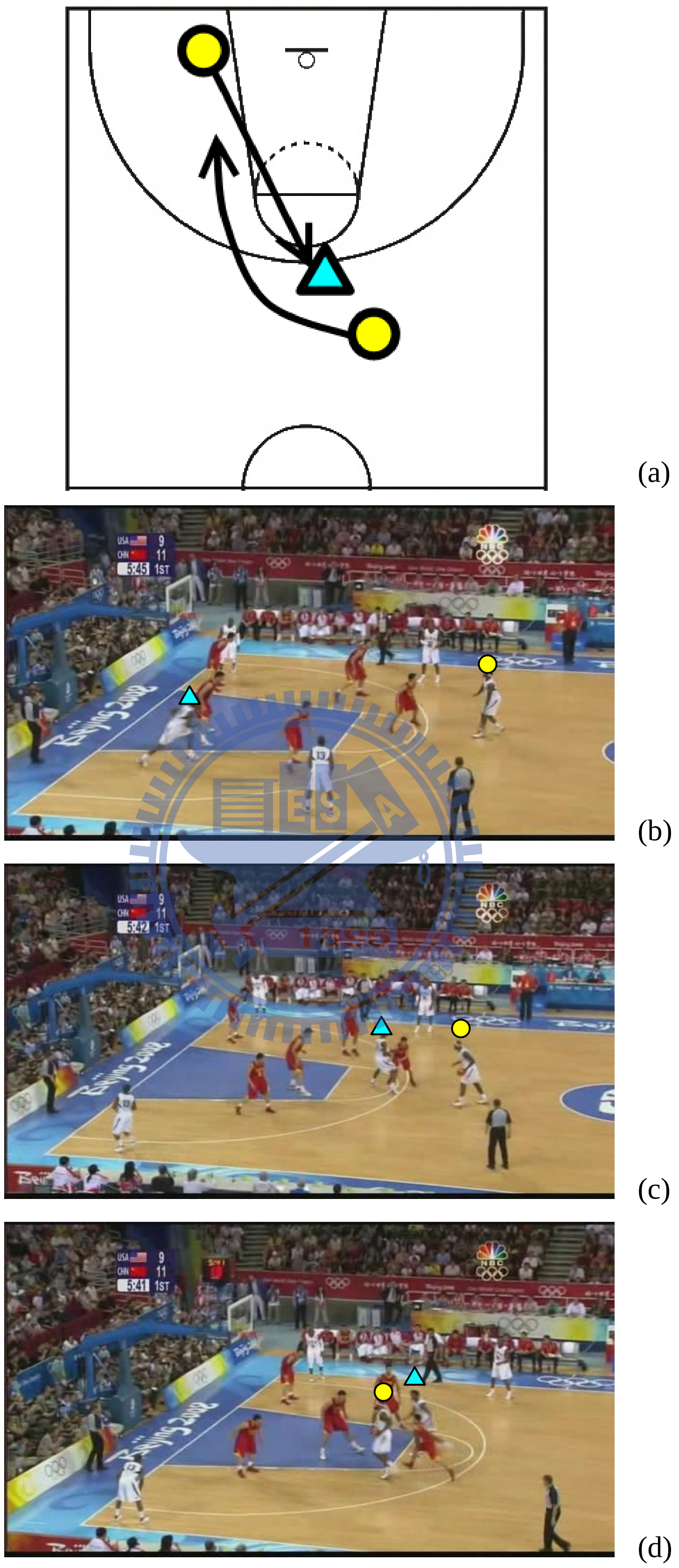


Figure 3.18: Example of back-screen. (a) Trajectories. (b) Before screen. (c) Setting screen. (d) After screen.

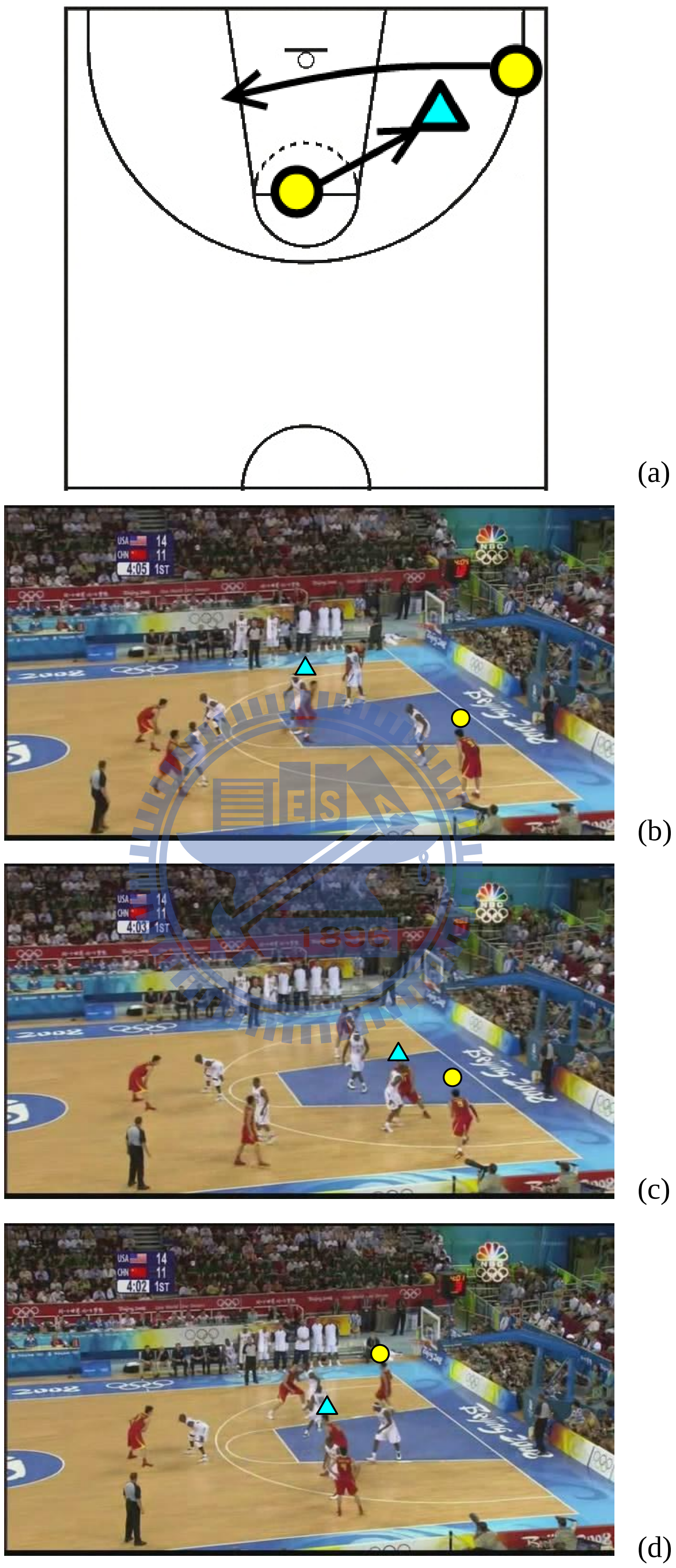


Figure 3.19: Example of down screen. (a) Trajectories. (b) Before screen. (c) Setting screen. (d) After screen.

In the *front-screen*, the screener is facing the defender that he is setting the screen on (the two marked red players in Figure 3.17). In the *back-screen*, the screener sets the screen on the back side of the defender, and the screener is usually facing away from the basket (the two marked white players in Figure 3.18). In the *down-screen*, the screener sets the screen usually down low for a player near the block, and is usually facing the basket with his back to the ball (the two marked red players in Figure 3.19). Figure 3.16 gives a sample tactic combined with the three types of screens. In Figure 3.16, circles present players that are denoted by O_i in the following paragraph, and triangles indicate positions at which screens are set. At first, the player O_3 sets a front-screen for O_1, who is the ball handler, and makes O_1 free to pass or shoot. Then O_3 moves to the low post and sets a down-screen for O_5, who cuts outside for the pass from O_1. Concurrently, O_4 moves to the high post and sets a back-screen for O_2, who goes back-door to the basket and makes a huge threat to the opponent team. Another example is the "pick and roll", which is perhaps the most famous and widely used basketball tactic. It usually starts with a back-screen for the ball handler. If the defender tries to guard the ball handler, the screener can move toward the basket, sometimes by a foot pivot, and now is open for a pass. Oppositely, if the defender tries to guard the screener, the ball handler has an open shot. A screen is also called a "pick" in basketball, and a foot pivot is a "roll" move. That is, the pick and roll means that the screener slips behind the defender (roll) to accept a pass after a screen (pick). There are some variations of the pick and roll, like the "pick and pop", where the screener moves for an open shot instead of rolling to the basket, or the "pick and slip", where the screener fakes setting a screen before slipping behind the defender to accept the pass.

With the background knowledge of screens, we are going to describe our proposed algorithm for screen analysis, including the screen detection during a possession and the screen classification at the end of a possession.

3.4.1 Screen Detection

Typically, in order to prevent the defensive players from helping their teammates, the offensive players tend to make the space wider and seldom stay close to each other. The only chance for an offensive player to stand next to his teammates is when a tactic is to be executed. Therefore, we can initially determine whether there is any screen based on the distance between two offensive players. Besides, since the screens are set to block defenders, there must be at least one defensive player between the screener and his teammate. That is, it can be an additional requirement to obtain a more accurate result of the screen detection. Furthermore, we can recognize who is the screener through this fact. Generally, the defender stays close to his target but not side by side since the defender has to prevent his target from driving to the basket. On the other hand, the screener must make contact with the defender in order to block the defender certainly. Accordingly, if we find that there are two offensive players close to each other, and there is at least one defensive player between them, we can certify that there is a screen and the screener is the offensive player who is closer to the defensive player. Once a screen is detected, we retain the current states of the screener and his teammate since we classify what type of screen they set via their trajectories before and after the screen.

Algorithm 4: Screen Detection

Input: offensive and defensive players, denoted by $players_{off}$ and $players_{def}$

Output: $screener$ and $screenee$, which means the offensive player whom the screener sets screen for

```
$screener := nil$
$screenee := nil$
**if** $\exists player_i \in players_{off}, player_j \in players_{off}: \delta_s < dist(player_i, player_j) < \Delta_s$ **and** $i \neq j$ **then**
    **if** $\exists defender \in players_{def}: dist(defender, player_i) < \delta_s$ **or** $dist(defender, player_j) < \delta_s$ **then**
        **if** $dist(defender, player_i) < dist(defender, player_j)$ **then**
            $screener := player_i$
            $screenee := player_j$
        **else**
            $screener := player_j$
            $screenee := player_i$
        **end if**
    **end if**
**end if**
**return** $screener$, $screenee$
```

3.4.2 Screen Classification

We have introduced the essence of screens in the previous section. Thus, we can observe that each screen type follows a specific pattern. A screener sets a back-screen by moving from the low post to the high post, and his teammate drive to the basket after the screen. The down-screen is set by a screener moving from the high post to the low post. The front-screen is usually set around the three-point line and the offensive player, who the screener is trying to free, moves to an open space instead of driving to the basket. We therefore have to utilize the trajectories of the screener and his teammate to judge the screen type. Recall that we retain the state at the moment that a screen is detected so that we can realize how the screener move before the screen and what his teammate does after the screen by the initial and the last state of the trajectory and the state when the screen is set. We denote the initial position of the screener as $\mathbf{p}_{init}$, the last position of the screener as $\mathbf{p}_{last}$, the position of the screener when the screen is set as $\mathbf{p}_{screen}$, and those of his teammate as $\mathbf{p'}_{init}$, $\mathbf{p'}_{last}$ and $\mathbf{p'}_{screen}$ respectively. Also, we denote the position of the basket as $\mathbf{p}_{basket}$. Accordingly, the only screen type that is set in the low post is the down-screen. Hence, we can first confirm the down-screen if $\mathbf{p}_{init}$ is farther away from the basket than $\mathbf{p}_{screen}$. The other two types are hard to distinguish through the move of the screener since the screener acts similarly in the back-screen and the front-screen, but we can still discriminate them by the trajectory of the offensive player whom the screener sets the screen for after the screen. We compute the angle θ_s between two vectors. The first one is the direction $\mathbf{d}_{player} = \mathbf{p'}_{last} - \mathbf{p'}_{screen}$ which the offensive player moves in, and the other indicates the direction $\mathbf{d}_{basket} = \mathbf{p}_{basket} - \mathbf{p'}_{screen}$. This angle θ_s tells us whether the offensive player tends to move to the basket or not. If the offensive player attempts to drive to the basket, it is a back-screen; otherwise, a front-screen is set. We formulate it as equation (10) and demonstrate it in Figure 3.20.

$$ScreenType = \begin{cases} Down, \text{if } |\mathbf{p}_{basket} - \mathbf{p}_{init}| > |\mathbf{p}_{basket} - \mathbf{p}_{screen}| \\ Back, \text{if } cos^{-1}\left(\dfrac{\mathbf{d}_{player} \cdot \mathbf{d}_{basket}}{|\mathbf{d}_{player}||\mathbf{d}_{basket}|}\right) < \theta_s \\ Front, \text{if } cos^{-1}\left(\dfrac{\mathbf{d}_{player} \cdot \mathbf{d}_{basket}}{|\mathbf{d}_{player}||\mathbf{d}_{basket}|}\right) \geq \theta_s \\ Undefined, \text{else} \end{cases} \tag{10}$$

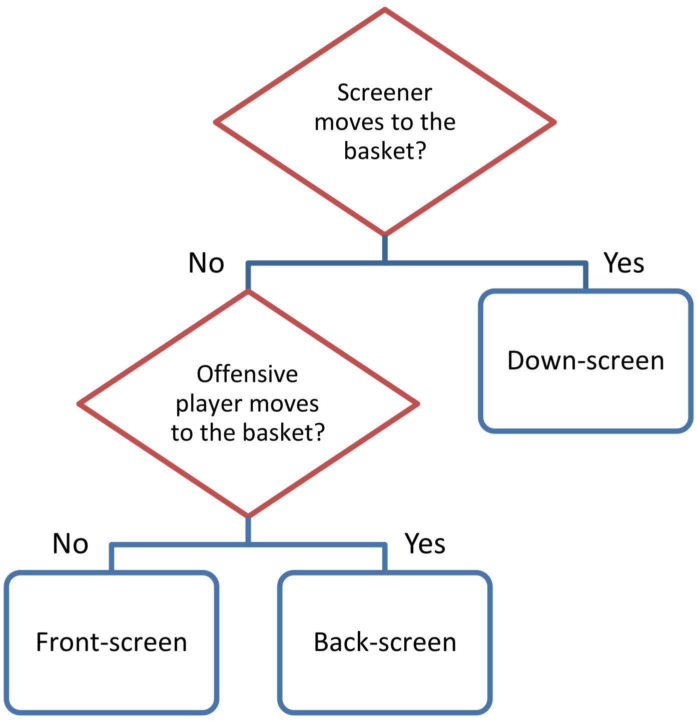

Screener moves to the basket?
No
Yes
Offensive player moves to the basket?
Down-screen
No
Yes
Front-screen
Back-screen

Figure 3.20: Diagram of screen classification.

Chapter 4. Experimental Results

In this chapter, we are going to show our experimental results of White Pixel Detection, Camera Calibration, Player Extraction, Player Classification, Possession Recognition, Player Tracking, and Tactic Analysis in the following sections. Furthermore, the phenomena observation and discussion are also included. We use three different games and manually extract ten possession clips with obvious tactic execution from each game. That is, we have total of thirty video clips for test. All of the three games are from 2008 Olympic Games. Game 1 is USA vs. AUS with dimensions 720x416. Game 2 is ARG vs. USA with dimensions 640x352. Game 3 is USA vs. CHN with dimensions 640x352. Table 4.1 describes our video sources. We randomly choose ten clips as training data to train our parameters, and the well-trained parameters are adaptive to all of our testing data.

Table 4.1: Video sources.

Game	Source	Width	Height
Game1	2008 Olympic Games USA vs. AUS	720	416
Game2	2008 Olympic Games ARG vs. USA	640	352
Game3	2008 Olympic Games USA vs. CHN	640	352

4.1 White Pixel Detection

In this section, we present and discuss the results of white pixel detection with/without the line structure constraint. The value of color filtering threshold σ_l is 128, brightness difference σ_d is 20, court line width w is 3 pixels, and line structure constraint window size b is 3 pixels. The parameters are mentioned in Section 3.2.1.1. Table 4.2 gives our experiment configuration. Table 4.3 presents the comparison of the number of white pixel candidates with/without the line structure constraint. Through Table 4.3, it is obvious that after applying the line structure constraint, most of the non-line white pixel candidates, up to 61%, are filtered out. Figure 4.1 shows some sample results of white line pixel detection. Figure 4.1 (a) shows original frames, (b) illustrates sample results without the line structure constraint, and (c) demonstrates sample results with the line structure constraint. Those discarded pixels mostly come from spectators, the score board overlay, the channel mark, and advertisement logos.

Table 4.2: Configuration for white pixel detection.

Parameter	Symbol	Value
Color filtering threshold	σ_l	128
Brightness difference	σ_d	20
Court line width	w	3 (pixels)
Line structure constraint	b	3 (pixels)

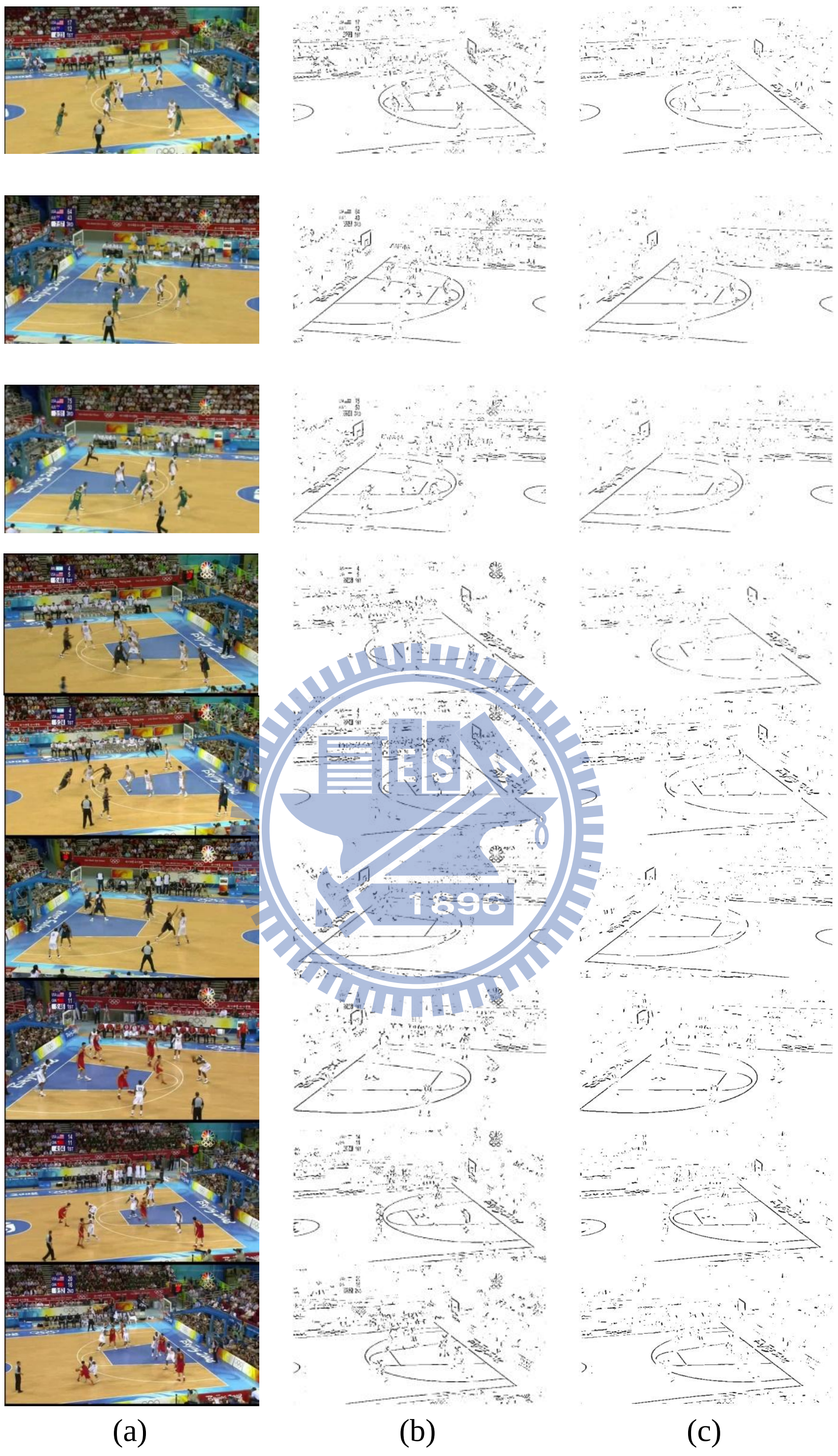


Figure 4.1: Results of white pixel detection. (a) Original frame. (b) Without line structure constraint. (c) With line structure constraint.

Table 4.3: Statistics of white pixel detection.

Clip	# of white pixel candidates without line structure constraint	# of white pixel candidates with line structure constraint	Ratio of discarded candidates (%)
Game1-1	11704	7391	36.851
Game1-2	10985	7109	35.284
Game1-3	10241	7019	31.462
Game1-4	11996	7494	37.529
Game1-5	10848	7135	34.228
Game1-6	10394	7129	31.412
Game1-7	11272	7352	34.776
Game1-8	11622	7156	38.427
Game1-9	10023	6916	30.999
Game1-10	10244	7300	28.739
Game2-1	7067	4089	42.140
Game2-2	9973	5719	42.655
Game2-3	8031	5277	34.292
Game2-4	9944	3862	61.163
Game2-5	7712	5553	27.995
Game2-6	9839	3842	60.951
Game2-7	8177	4567	44.148
Game2-8	8377	4092	51.152
Game2-9	8404	5297	36.970
Game1-10	7636	4212	44.840
Game3-1	8069	5140	36.299
Game3-2	9200	5685	30.201
Game3-3	9784	5540	43.377
Game3-4	7901	5050	36.084
Game3-5	9717	5664	41.710
Game3-6	9734	5141	47.185
Game3-7	8077	5766	28.612
Game3-8	9788	5180	47.078
Game3-9	8970	5205	41.973
Game3-10	9372	5419	42.179

4.2 Camera Calibration

In this section, we show some results of camera calibration, and some demonstrations are shown in Figure 4.2, where black lines are court lines extracted by our proposed method, and red lines represent real court model that is projected onto image coordinate. In Figure 4.2 (3), (6), (7), bottom sideline and bottom edge of the restricted area are used to calculate camera calibration and the results are correct. Thus, we can figure out that our proposed method has the flexibility in using different court lines to calculate camera calibration. Nevertheless, in Figure 4.2 (8), the half-court line is projected to a wrong position due to some computational error. The average projection error in Table 4.4 is computed by the distance between the position which a manually pointed intersection of court lines is projected to and the corresponding corner of court model. Averagely the error is lesser than 0.06 meters. Remark that court lines have width so that when selecting ground truth, we may not point the real intersection points of the extracted court lines. As a result, they are not projected to the same positions as court model. Therefore, despite the fact that the projection error is not zero, this result is satisfactory and convincible.

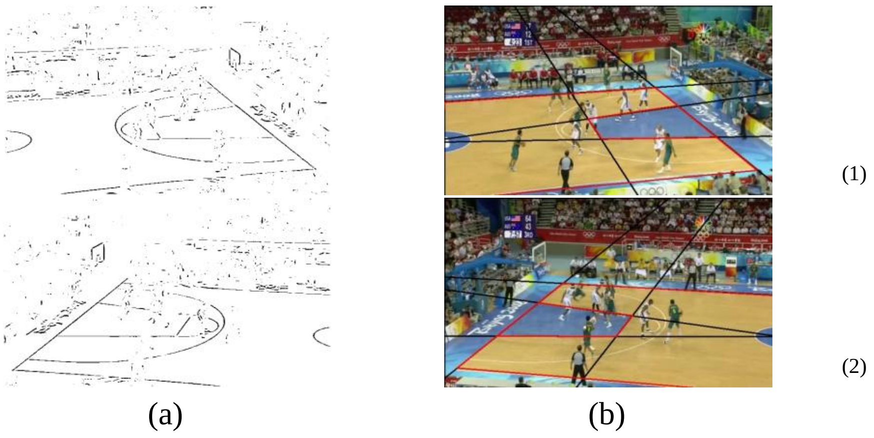


Figure 4.2: Results of camera calibration. (a) White line pixels. (b) Extracted court lines and camera calibration.

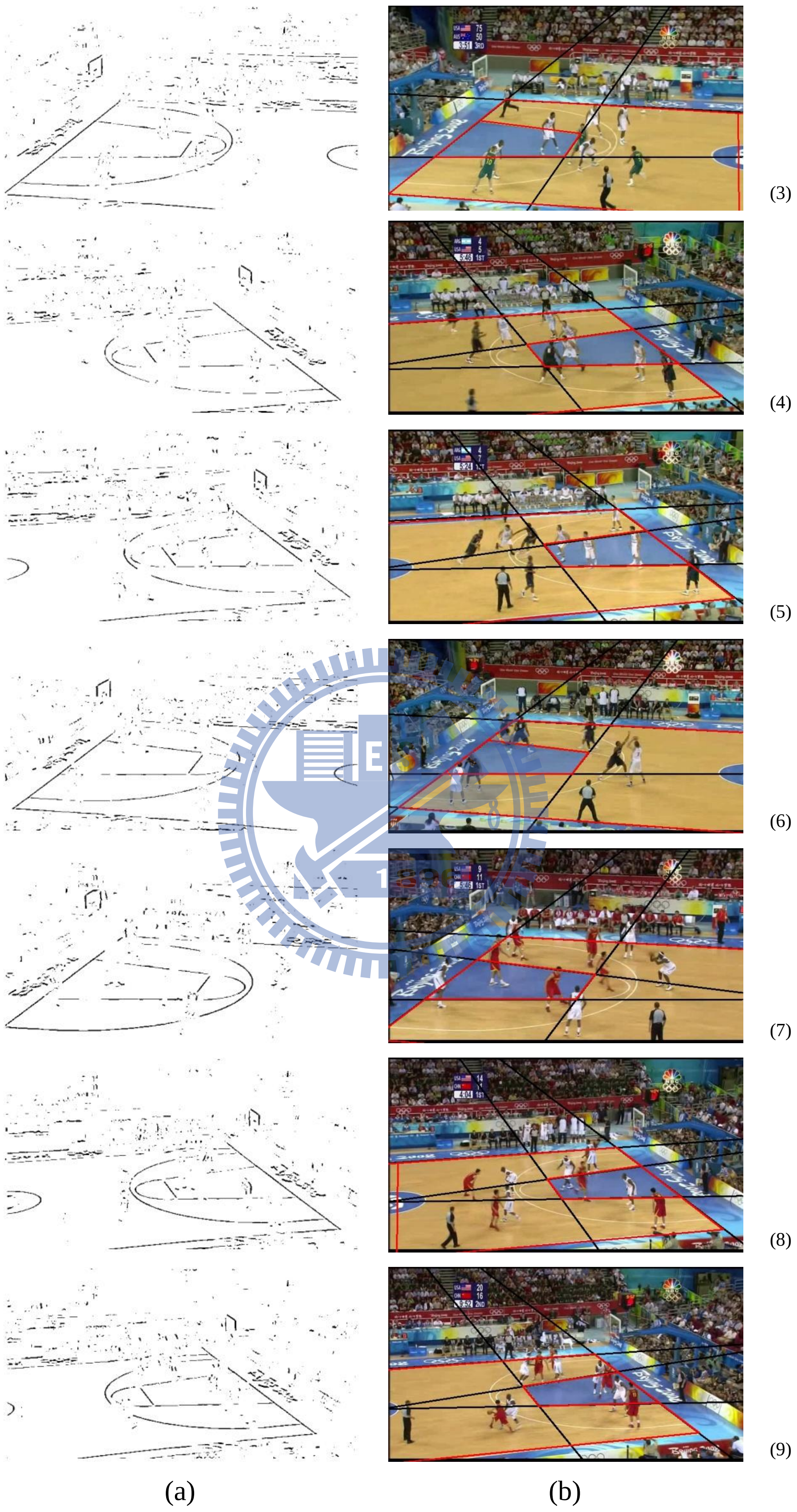

(3)
(4)
(5)
(6)
(7)
(8)
(9)
(a)
(b)

Figure 4.2: Results of camera calibration. (a) White line pixels. (b) Extracted court lines and camera calibration. (cont’d)

Table 4.4: Average projection error of camera calibration.

Clip	Average projection error (meters)
Game1-1	0.04
Game1-2	0.019
Game1-3	0.063
Game1-4	0.02
Game1-5	0.041
Game1-6	0.021
Game1-7	0.044
Game1-8	0.146
Game1-9	0.045
Game1-10	0.048
Game2-1	0.03
Game2-2	0.5
Game2-3	0.084
Game2-4	0.075
Game2-5	0.018
Game2-6	0.077
Game2-7	0.449
Game2-8	0.086
Game2-9	0.131
Game2-10	0.104
Game3-1	0.037
Game3-2	0.064
Game3-3	0.132
Game3-4	0.052
Game3-5	0.396
Game3-6	0.088
Game3-7	0.028
Game3-8	0.079
Game3-9	0.125
Game3-10	0.024

4.3 Player Extraction

Some sample results of foreground object extraction and the corresponding dominant color map are shown in Figure 4.3. Figure 4.3 (a) shows original frames, (b) demonstrates background region, that is, pixels with color that can be found in dominant color map, and (c) illustrates extracted foreground objects. In order to keep the jersey colors and exclude the other colors such as skins and shoes, we tend to make the foreground object mask "smaller". Hence, we perform a morphological operation to erode the initial foreground object mask without dilating it. The major problem of our player extraction method is that we only consider pixels within the region of interest, that is, the court region. Thus, when there are players standing near court boundaries, their bodies may be cut. This may affect the result of player tracking, but due to our knowledge of basketball, those players standing near the court boundaries usually do nothing but just stay there and wait for passes. Also, when standing near the court boundaries, they do not have space to be set screens or tactics for. That is, their trajectories usually have nothing to do with our screen classification algorithm and we can simply ignore them until they move closer to the center of the court.

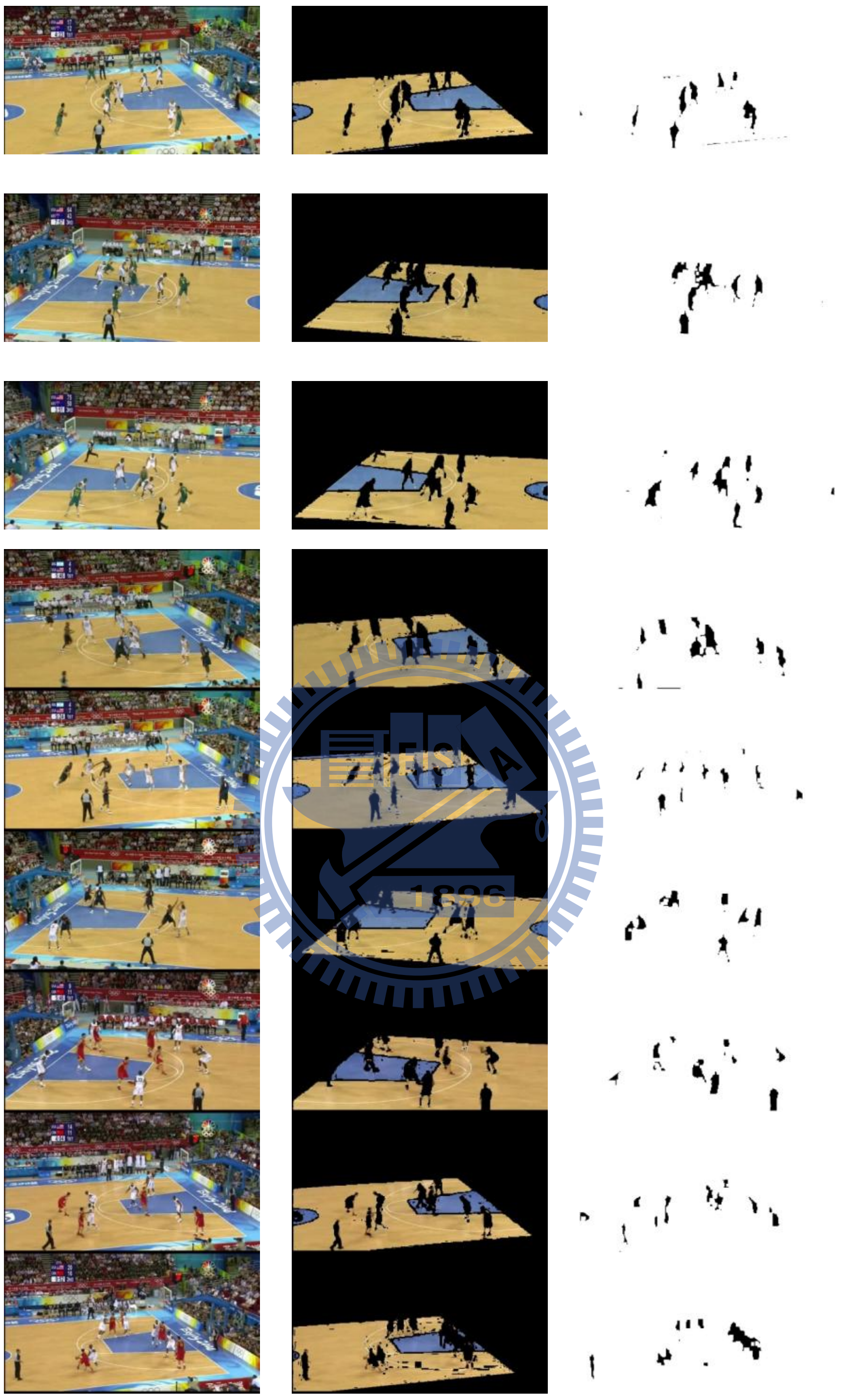

(a) (b) (c)

Figure 4.3: Results of player extraction. (a) Original frame. (b) Dominant color map. (c) Foreground objects.

4.4 Player Classification

We demonstrate some results of player classification and discuss the phenomena in this section. Through player classification, some noise with different colors from the jersey colors of the two teams is removed, such as referees and logos on the floor (see red circles in Figure 4.4 (1), (3), (7), (8), (9)). However, sometimes player bodies are divided into parts by some small objects such as arms and numbers on jerseys (see blue squares in Figure 4.4 (1), (3)). Even though we perform the dilation operation and try to remove gaps, we are not able to completely cover all situations. On the other hand, once we discover that there are two objects close to each other, it is hard to judge whether they are two parts of a player or two different players (see green triangles in Figure 4.4 (4), (6), (8)). Our way to solve this problem is to regard them as different objects, and let the tracking procedure clarify if they are the same object through their trajectories.

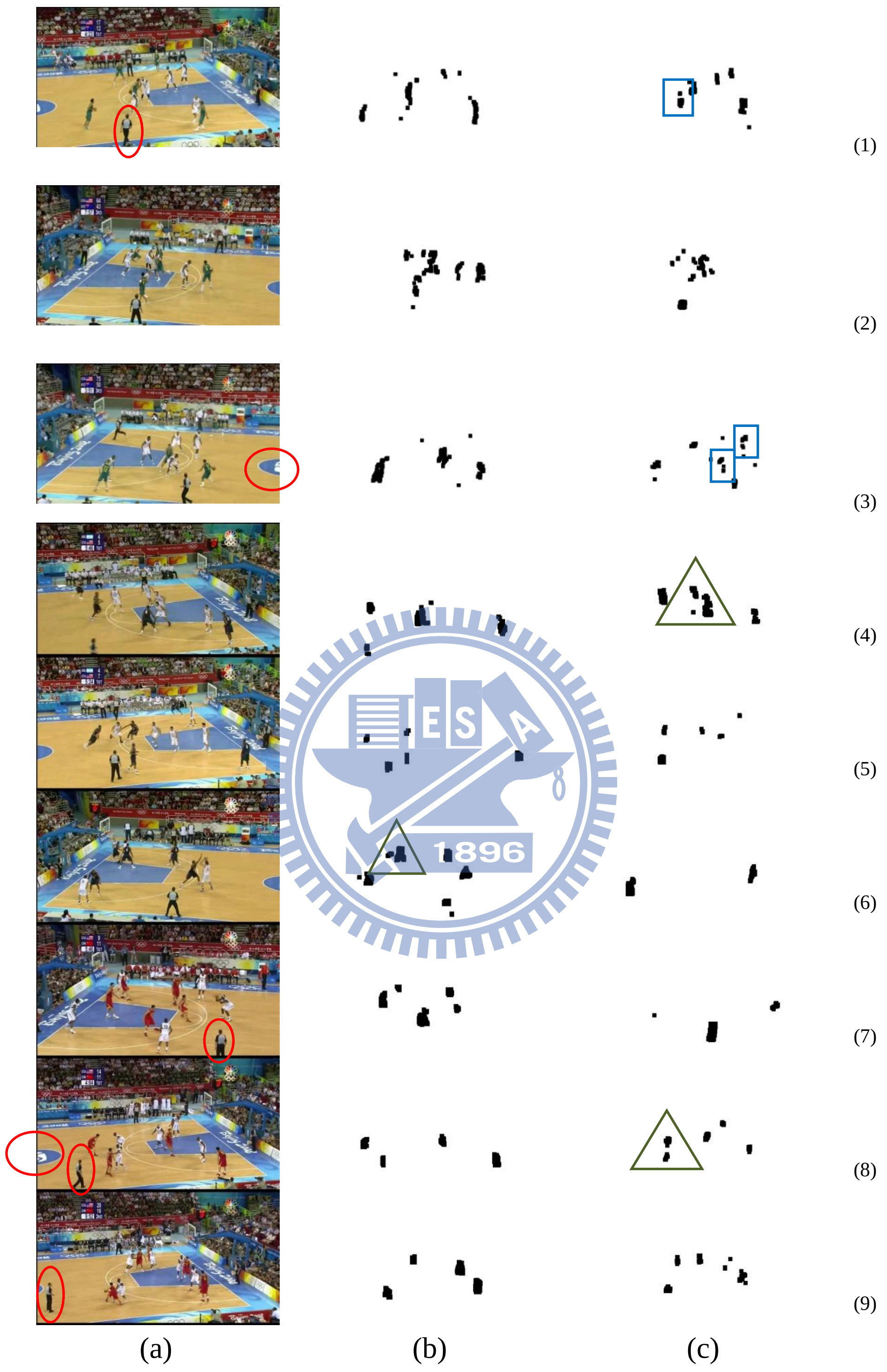


Figure 4.4: Results of player classification. (a) Original frame. (b) Player mask of team 1. (c) Player mask of team 2.

4.5 Possession Recognition

In this section, we take Figure 4.4 for example and show the correspondence between which team has the possession and the average distance between players of that team and the basket in Table 4.5. Table 4.5 demonstrates the average distances of the two teams in each row for Figure 4.4. For the team number and its jersey color, see Figure 4.4. Figure 4.4 (a) also provides the ground truth of possession, and we can verify that all of the results are correct.

Table 4.5: Statistical results of possession recognition.

Figure 4.4	**Average distance of Team 1 (meters)**	**Average distance of Team 2 (meters)**	**Possession**
(1)	7.207	4.795	Team 1 (green)
(2)	4.964	4.799	Team 1 (green)
(3)	7.485	6.447	Team 1 (green)
(4)	7.832	4.559	Team 1 (black)
(5)	7.748	6.099	Team 1 (black)
(6)	5.499	7.031	Team 2 (white)
(7)	4.453	7.555	Team 2 (white)
(8)	7.483	5.680	Team 1 (red)
(9)	5.577	5.062	Team 1 (red)

4.6 Player Tracking

Since the Kalman filter which we use to track players is a point tracking mechanism, we can evaluate the performance with *precision* and *recall* measures defined as [13]

$$\text{precision} = \frac{\text{\# of correct correspondences}}{\text{\# of established correspondences}}, \quad \text{recall} = \frac{\text{\# of correct correspondences}}{\text{\# of actual correspondences}}$$

Established correspondences represent total number of created trackers, and actual correspondences denote number of object at the moment. In this experiment, the tracker termination criterion ε_f is 10 frames and the maximum distance for a valid measurement δ_t is 0.6 meters. Remark that the world record of 100 meters race is about 10 seconds, that is, the average velocity is 10 meters per second. Our video sources have 30 frames per second, namely, the time period between two adjacent frames is about 0.03 seconds. Accordingly, the fastest runner can move about 0.3 meters between two adjacent frames. Thus, it is our basis of δ_t and we set it for a double value since players might move to the opposite direction suddenly. Table 4.6 shows our configuration and Table 4.7 demonstrates the evaluation performance. We have total of 89.5% of precision and 89.994% of recall. Most failures result from occlusion and merging problem. Occlusion is always the most notorious enemy of object tracking. Unfortunately, occlusions occur frequently in basketball video. Furthermore, it is difficult to predict trajectories of basketball players since they usually change directions rapidly. As a consequence, occlusion cannot be perfectly solved in basketball video. Besides, as mentioned in Section 4.4, player bodies sometimes are divided into parts due to arms or numbers in jerseys. Although we can tell duplicate objects by their trajectories, they already decrease the performance of player tracking at the moment.

Table 4.6: Configuration of player tracking.

Parameter	Symbol	Value
Tracker termination criterion	ε_f	10 (frames)
Maximum distance threshold	δ_t	0.6 (m)

Table 4.7: Performance of player tracking.

Clip	# of frames	Precision (%)	Recall (%)
Game1-1	175	89.266	88.784
Game1-2	175	76.386	79.940
Game1-3	100	80.455	82.969
Game1-4	119	81.529	82.077
Game1-5	207	88.160	89.514
Game1-6	101	84.832	85.362
Game1-7	238	82.911	84.286
Game1-8	186	86.023	85.179
Game1-9	201	84.735	85.208
Game1-10	232	89.177	90.340
Game2-1	132	89.675	89.303
Game2-2	241	87.262	86.940
Game2-3	182	92.830	93.595
Game2-4	133	86.801	88.795
Game2-5	157	88.641	88.180
Game2-6	198	88.150	86.647
Game2-7	145	92.697	94.036
Game2-8	151	90.361	91.489
Game2-9	181	86.882	87.531
Game2-10	194	93.484	92.273
Game3-1	150	94.262	94.456
Game3-2	150	93.284	94.429
Game3-3	100	91.398	90.563
Game3-4	241	92.009	92.842
Game3-5	245	99.194	97.270
Game3-6	151	92.557	94.018
Game3-7	138	92.497	93.975
Game3-8	219	93.242	94.167
Game3-9	205	92.336	92.790
Game3-10	210	97.020	96.324
Total	5257	89.500	89.994

4.7 Tactic Analysis

We illustrate the results of our screen detection and classification algorithm, and show the accuracy in this section. In this experiment, the minimum screen distance δ_s is 2 meters, the maximum screen distance Δ_s is 3 meters, and the angle between moving direction and basket direction is 45 degrees. The screen distance can also be regarded as the distance between a defender and his target, which is usually 2 to 3 meters, since the screener is almost at the same position as the defender. The determination of θ_s depends on user's sensitivity to back-screens, and we find that 45 is a moderate value which can detect most back-screens without too many false positives. Table 4.8 shows our configuration and Figure 4.5 illustrates some sample results. In Figure 4.5, cyan triangles indicate screeners, and yellow circles represent their offensive teammates. Figure 4.5 (a) demonstrates the moment when a screen is detected, and Figure 4.5 (b) shows the trajectories of the screener and his offensive teammate with which we classify the screen type. In Figure 4.5 (1), the screener moves from near the top sideline to the top of the three-point line and sets the screen, and his offensive teammate moves from near the bottom sideline to the top sideline. It should be a front-screen but the screener is closer to the basket when he sets the screen than he starts to move. As a result, our algorithm regards it as a down-screen. In Figure 4.5 (2), the screener moves from the free-throw line to the low-post, and his offensive teammate tries to move to the three-point line. It is a special case since the screener is setting the screen on the moving path of his offensive teammate instead of standing next by the defender so that the screener is farther away to the defender than his offensive teammate when the screen is detected, and our algorithm fails to recognize the screener correctly. In Figure 4.5 (3), the screener sets the screen around the free-throw line, and his offensive teammate drives to the basket. It is a typical back-screen and our algorithm identifies it correctly. In Figure 4.5 (4), the screener moves from the restricted area to the free-throw line and sets the screen, and his offensive teammate moves around the three-point line and reaches the free-throw line at last. Since the offensive player stops at the free-throw line without driving to the basket, it is a front-screen and successfully classified by our algorithm. In Figure 4.5 (5), the screen is set near the top edge of the restricted area, and the offensive player moves from outside the three-point line to the free-throw line. Although the offensive player does not drive to the basket and it should be a front-screen, the angle between his moving direction and the basket direction is small and our algorithm mistakes the screen type. In Figure 4.5 (6), the screener moves along the baseline, from under the basket to the bottom edge of the restricted area, and his offensive teammate moves from the baseline to the free-throw line. It is correctly classified as a front-screen. In Figure 4.5 (7), the screener moves from the free-throw line to the top of the three-point line, and his offensive teammate drives to the basket after the screen. It is also a standard back-screen. In Figure 4.5 (8), the screener moves down from the free-throw line, and his offensive teammate tries to pass through the restricted area from the bottom sideline. This down-screen is correctly classified by our algorithm. In Figure 4.5 (9), the screener moves up from near the free-throw line, and his offensive teammate moves to the upper region. This is no doubt a front-screen. Table 4.9 demonstrates the results of our screen classification algorithm using Figure 4.5. Table 4.10 shows the accuracy of screen detection. Our screen detection algorithm detects over 96% of screen frames. Most missing frames result from beginnings and ends of screens since the two offensive players move and the distance between them is hard to control.

Table 4.8: Configuration of screen detection and classification.

Parameter	Symbol	Value
Minimum screen distance	δ_s	2 (m)
Maximum screen distance	Δ_s	3 (m)
Angle between moving direction and basket direction	θ_s	45 (degree)

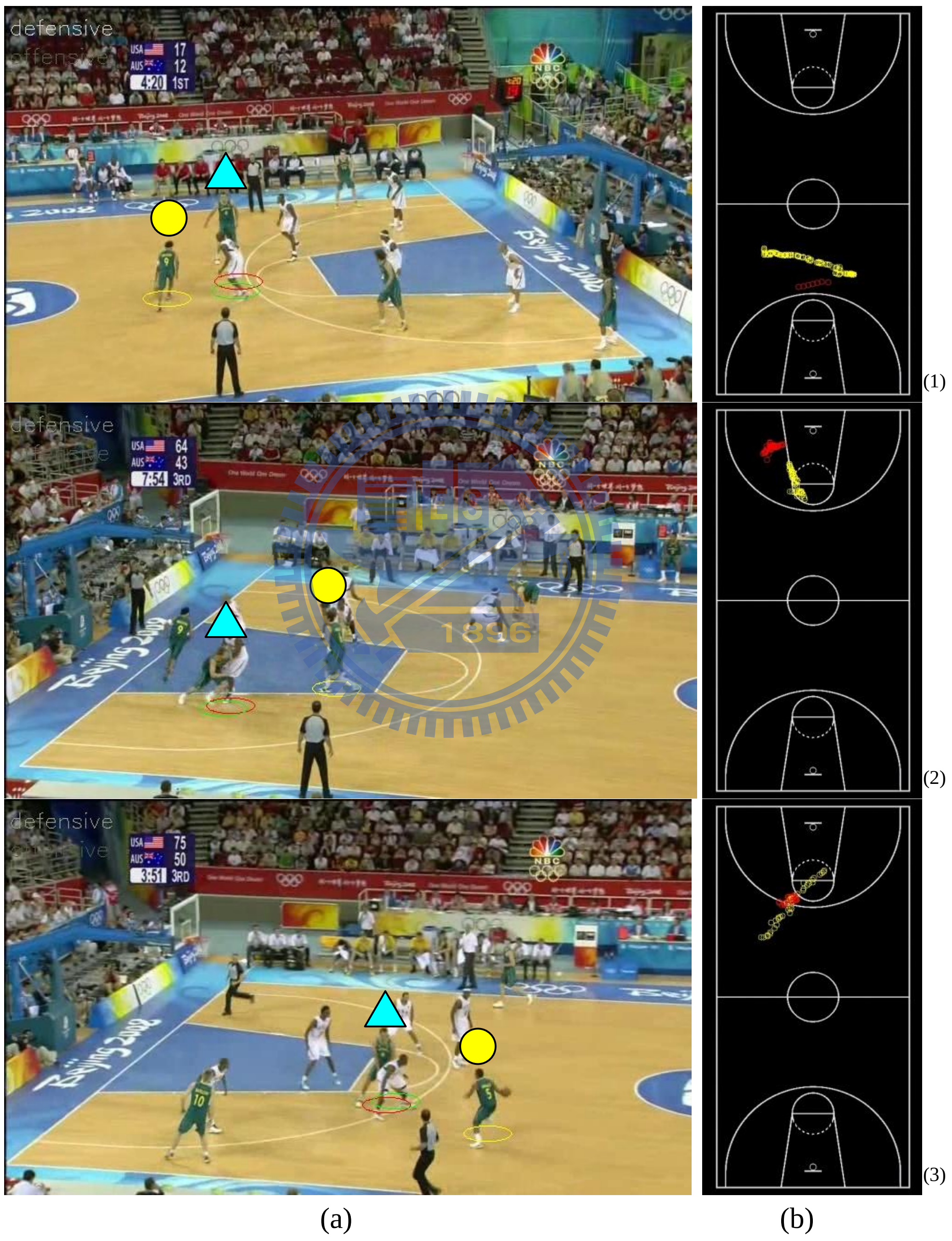


Figure 4.5: Results of tactic analysis. (a) Screen detection (b) Screen classification.

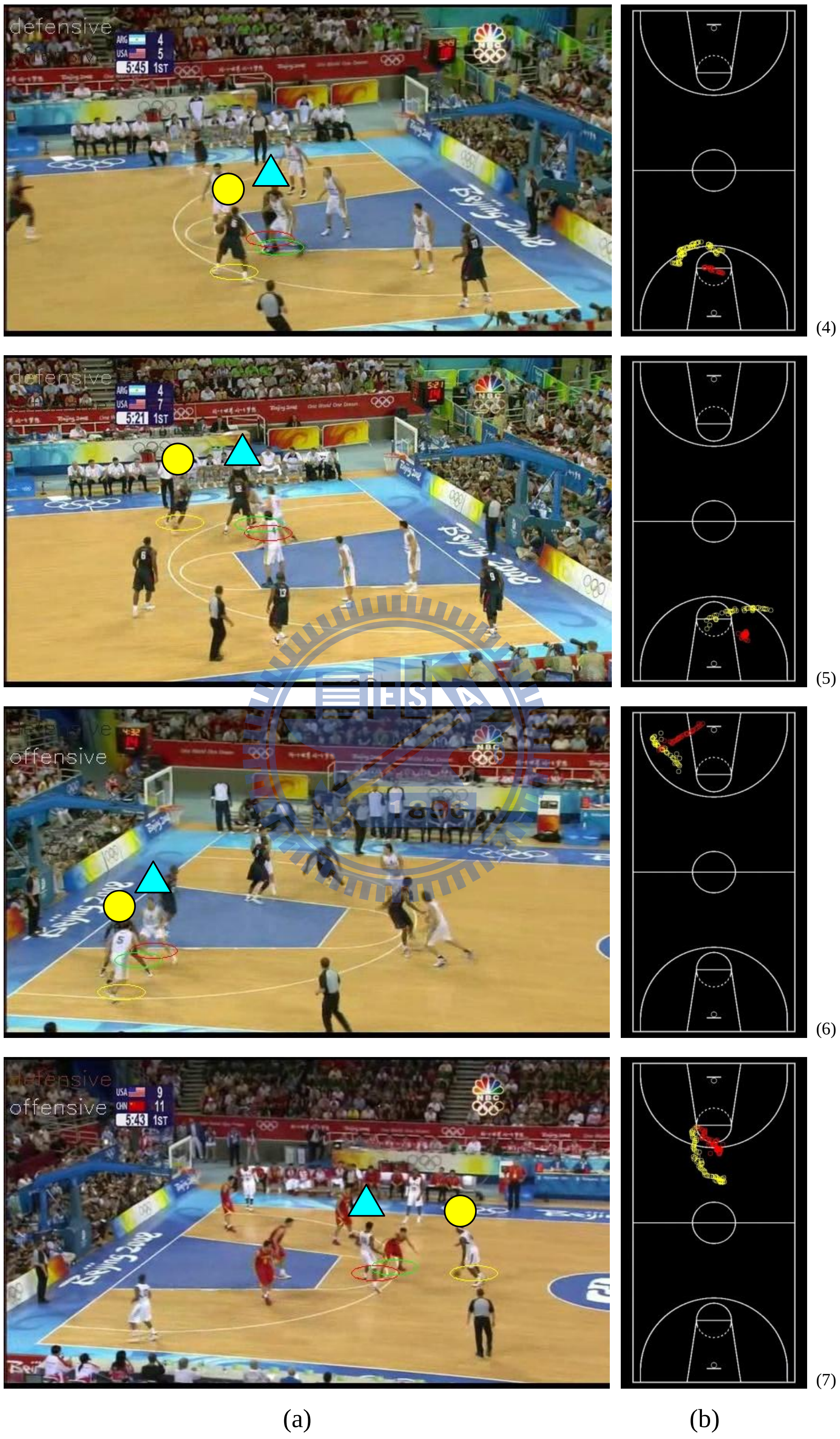
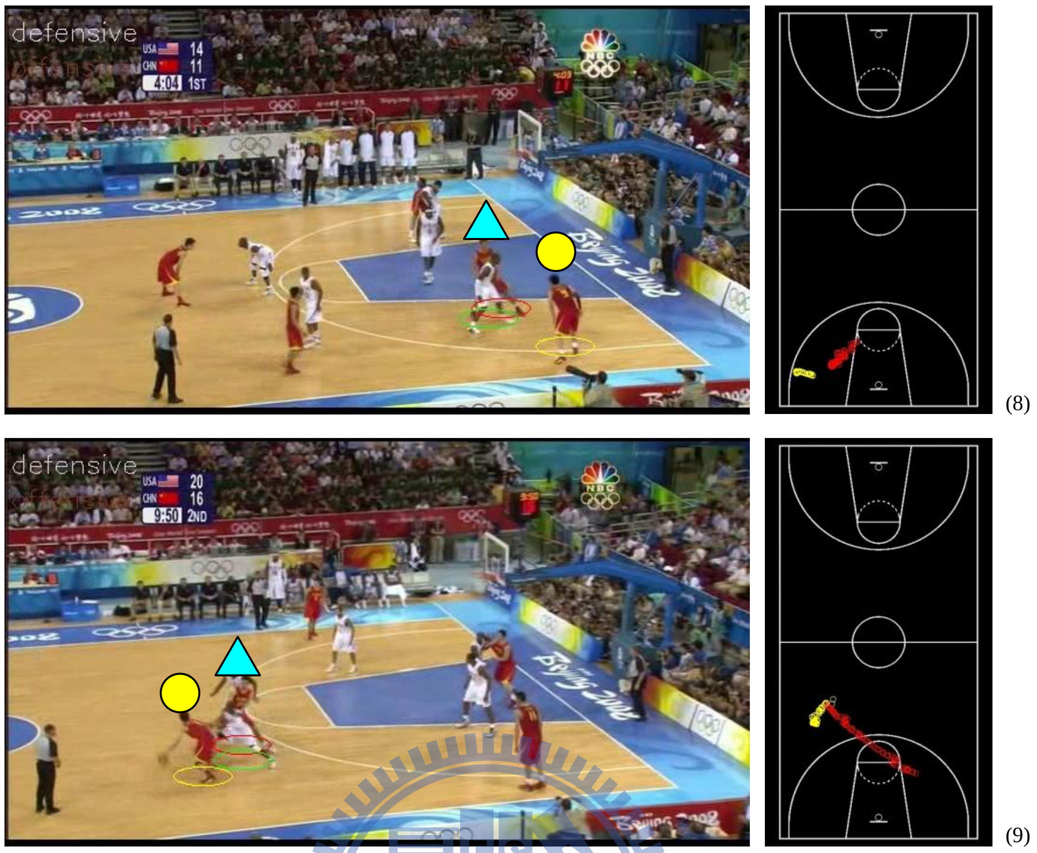


Figure 4.5: Results of tactic analysis. (a) Screen detection (b) Screen classification. (cont'd)



(a) (b)

Figure 4.5: Results of tactic analysis. (a) Screen detection. (b) Screen classification. (cont'd)

Table 4.9: Corresponding results of screen classification to Figure 4.5.

Figure 4.5	**Real screen type**	**Identified screen type**
(1)	Front-screen	Down-screen
(2)	Down-screen	Back-screen
(3)	Back-screen	Back-screen
(4)	Front-screen	Front-screen
(5)	Front-screen	Back-screen
(6)	Front-screen	Front-screen
(7)	Back-screen	Back-screen
(8)	Down-screen	Down-screen
(9)	Front-screen	Front-screen

Table 4.10: Accuracy of screen detection.

Clip	# of real screen frames	# of detected screen frames	Accuracy (%)
Game1-1	49	48	97.959
Game1-2	39	37	94.872
Game1-3	38	38	100
Game1-4	42	39	92.857
Game1-5	45	40	88.889
Game1-6	36	35	97.222
Game1-7	45	42	93.333
Game1-8	44	42	95.455
Game1-9	36	35	97.222
Game1-10	33	30	90.909
Game2-1	37	35	94.595
Game2-2	43	41	95.349
Game2-3	35	34	97.143
Game2-4	38	37	97.368
Game2-5	44	44	100
Game2-6	38	36	94.734
Game2-7	31	31	100
Game2-8	48	47	97.917
Game2-9	42	40	95.238
Game2-10	38	34	89.474
Game3-1	47	45	95.745
Game3-2	36	36	100
Game3-3	35	33	94.286
Game3-4	48	48	100
Game3-5	33	30	90.909
Game3-6	33	32	96.970
Game3-7	43	40	93.023
Game3-8	39	39	100
Game3-9	41	40	97.561
Game3-10	32	31	96.875
Total	1179	1139	96.607

Chapter 5. Conclusions

We have proposed a scheme that can detect and classify screens in basketball games, which is the fundamental essence of basketball tactics. Through combining screens and trajectories in each possession, our system is able to recognize what tactics are executed. The collected tactics are then indexed so that users can query the tactic that they are interested in. Our system can also work with other researches and lead to more applications. For instance, when combining the shooting location estimation [7], we can speculate on the movement of the ball and obtain further information about basketball tactics. It is difficult to tell who the ball handler is at the beginning of a possession, and it prohibits us from tracking the ball as well. Nevertheless, once we know the shooting location, we can trace back how the ball is passed to the shooter. For another instance, with the wide-open detection [1, 50], we can verify whether the execution of a tactic is successful in making open shot since tactics are set in order to make open shots, with which the players score easily.

(a) (b)

Figure 5.1: Real game example. (a) Coach setting tactic. (b) Tactic execution.

Our system currently identifies tactics with the patterns of screens and trajectories from video clips. That is, we do not know if the verified tactics are equivalent to those set by the coach. In fact, there are some factors affecting the execution of tactics, such as the interference from the defensive players. Although the team on offense sets tactics to block the defensive players, the opponent team has its way to counter. Imagine that the ball handler is trying to pass the ball to a teammate as the tactic indicates, but that teammate is being double teamed and is not free to catch the pass. The ball handler therefore has no choice but to pass the ball to another teammate. As a result, the behaviors of offensive players we see on the screen may not follow the instructions from the coach, and the tactics identified by our system may differ from those set by the coach. Figure 5.1 is a real example that the tactic execution is different from the coach's instruction. Hence, we want to add coach's instructions to our system in the future. With the real instruction, we not only can verify the identified tactics but also figure out why the offensive players are not able to implement the tactics. First, we query the database with the real tactic and obtain its pattern of screens and trajectories. After analyzing the behavior of the players from the video clip, we compare the result with the pattern of the real tactic set by the coach. By searching the difference between the performance of the players and the instruction of the coach, we can realize what keeps the team on offense from executing the tactic successfully. Furthermore, we can speculate on the strategy used by the team on defense, which is also an important issue that both professional coaches and players are interested in. We hope to have the chance to cooperate with basketball teams so that we can improve the proposed system with the real information they provide.

Bibliography

[1] M.-H. Chang, M.-C Tien, J.-L. Wu, "WOW: Wild-Open Warning for Broadcast Basketball Video Based on Player Trajectory," in *Proceedings of the ACM International Conference of Multimedia*, pp. 821-824, 2009.

[2] D. Farin, S. Krabbe, P. H. N. d. With, W. Effelsberg, "Robust Camera Calibration for Sport Videos Using Court Models," in *Proceedings of Storage and Retrieval Methods and Applications for Multimedia*, pp. 80-91, 2004.

[3] G.-G. Lee, H.-K. Kim, W.-Y. Kim, "Highlight Generation for Basketball Video Using Probabilistic Excitement," in *Proceedings of the IEEE International Conference on Multimedia and Expo*, pp. 318-321, 2009.

[4] L. Li, Y. Chen, W. Hu, W. Li, X. Zhang, "Recognition of Semantic Basketball Events Based on Optical Flow Patterns," in *Proceedings of the International Symposium on Visual Computing*, pp. 480-488, 2009.

[5] Y. Zhang, C. Xu, Y. Rui, J. Wang, H. Lu, "Semantic Event Extraction from Basketball Games Using Multi-Modal Analysis," in *Proceedings of the IEEE International Conference on Multimedia and Expo*, pp. 2190-2193, 2007.

[6] W. Kim, H. Kong, J. Choi, K. Kim, P. Kim, "Event Detection from Basketball Video Using Audio Information," in *Proceedings of the International Conference on Artificial Intelligence*, pp. 716-721, 2002.

[7] H.-T. Chen, M.-C. Tien, Y.-W. Chen, W.-J. Tsai, S.-Y. Lee, "Physics-Based Ball Tracking and 3D Trajectory Reconstruction with Applications to Shooting Location Estimation in Basketball Video," *Journal of Visual Communication and Image Representation*, vol. 20, no. 3, pp. 204-216, 2009.

[8] M.-C. Tien, H.-T. Chen, Y.-W. Chen, M.-H. Hsiao, S.-Y. Lee, "Shot Classification of Basketball Videos and Its Application in Shooting Position Extraction," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pp. 1085-1088, 2007.

[9] G. Miao, G. Zhu, S. Jiang, Q. Huang, C. Xu, W. Gao, "A Real-Time Score Detection and Recognition Approach for Broadcast Basketball Video," in *Proceedings of the IEEE International Conference on Multimedia and Expo*, pp. 1691-1694, 2007.

[10] B. Jahne, "Digital Image Processing," *Springer Verlag*, 2002.

[11] Y. Liu, S. Jiang, Q. Ye, W. Gao, Q. Huang, "Playfield Detection Using Adaptive GMM and Its Application," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, pp. 421-424, 2005.

[12] G. Welch, G. Bishop, “An Introduction to the Kalman Filter,” *Technical Report* TR95-041, University of North Carolina at Chapel Hill, 1995.

[13] A. Yilmaz, O. Javed, M. Shah, “Object Tracking: A Survey,” *ACM Computing Surveys*, vol. 38, no. 4, 2006.

[14] H. Moravec, “Visual Mapping by a Robot Rover,” in *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 598-600, 1979.

[15] C. Harris, M. Stephens, “A Combined Corner and Edge Detector,” in *4th Alvey Vision Conference*, pp. 147-151, 1988.

[16] D. G. Lowe, “Distinctive Image Features from Scale-invariant Keypoints,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91-110, 2004.

[17] K. Mikolajczyk, C. Schmid, “A Performance Evaluation of Local Descriptors,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 257-263, 2003.

[18] D. Comaniciu, P. Meer, “Mean Shift Analysis and Applications,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1197-1203, 1999.

[19] J. Shi, J. Malik, “Normalized Cuts and Image Segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888-905, 2000.

[20] V. Caselles, R. Kimmel, G. Sapiro, “Geodesic Active Contours,” in *Proceedings of IEEE International Conference on Computer Vision*, pp. 694-699, 1995.

[21] C. Stauffer, W. E. L. Grimson, “Learning Patterns of Activity Using Real-Time Tracking,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 747-757, 2000.

[22] N. Oliver, B. Rosario, A. Pentland, “A Bayesian Computer Vision System for Modeling Human Interactions,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 831-843, 2000.

[23] K. Toyama, J. Krumm, B. Brumitt, B. Meyers, “Wallflower: Principles and Practice of Background Maintenance,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 255-261, 1999.

[24] A. Monnet, A. Mittal, N. Paragios, V. Ramesh, “Background Modeling and Subtraction of Dynamic Scenes,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1305-1312, 2003.

[25] C. Papageorgiou, M. Oren, T. Poggio, “A General Framework for Object Detection,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 555-562, 1998.

[26] H. A. Rowley, S. Baluja, T. Kanade, “Neural Network-Based Face Detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 1, pp. 23-38, 1998.

[27] P. A. Viola, M. J. Jones, D. Snow, “Detecting Pedestrians Using Patterns of

Motion and Appearance," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 734-741, 2003.

[28] R. Jain, H. Nagel, "On the Analysis of Accumulative Difference Pictures from Image Sequences of Real World Scenes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 1, no. 2, pp. 206-214, 1979.

[29] C. R. Wren, A. Azarbayejani, T. Darrell, A. Pentland, "Pfinder: Real-Time Tracking of the Human Body," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 780-785, 1997.

[30] D. Comaniciu, P. Meer, "Mean Shift: A Robust Approach Toward Feature Space Analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603-619, 2002.

[31] D. Comaniciu, V. Ramesh, P. Meer, "Kernel-Based Object Tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 5, pp. 564-575, 2003.

[32] V. Salari, I. K. Sethi, "Feature Point Correspondence in the Presence of Occlusion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 1, pp. 87-91, 1990.

[33] C. J. Veenman, M. J. T. Reinders, E. Backer, "Resolving Motion Correspondence for Densely Moving Points," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 1, pp. 54-72, 2001.

[34] T. Broida, R. Chellappa, "Estimation of Object Motion Parameters from Noisy Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 1, pp. 90-99, 1986.

[35] Y. Bar-Shalom, T. Foreman, "Tracking and Data Association," *Academic Press Inc.*, 1988.

[36] R. L. Streit, T. E. Luginbuhl, "Maximum Likelihood Method for Probabilistic Multi-hypothesis Tracking," in *Proceedings of the International Society for Optical Engineering*, vol. 2235, pp. 394-405, 1994.

[37] J. Shi, C. Tomasi, "Good Features to Track," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 593-600, 1994.

[38] H. Tao, H. S. Sawhney, R. Kumar, "Object Tracking with Bayesian Estimation of Dynamic Layer Representations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 1, pp. 75-89, 2002.

[39] M. J. Black, A. D. Jepson, "EigenTracking: Robust Matching and Tracking of Articulated Objects Using a View-Based Representation," *International Journal of Computer Vision*, vol. 26, no. 1, pp. 63-84, 1998.

[40] S. Avidan, "Support Vector Tracking," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 184-191, 2001.

[41] M. Isard, A. Blake, “CONDENSATION – Conditional Density Propagation for Visual Tracking,” *International Journal of Computer Vision*, vol. 29, no. 1, pp. 5-28, 1998.

[42] M. Bertalmio, G. Sapiro, G. Randall, “Morphing Active Contours,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 7, pp. 733-737, 2000.

[43] R. Ronfard, “Region-based Strategies for Active Contour Models,” *International Journal of Computer Vision*, vol. 13, no. 2, pp. 229-251, 1994.

[44] D. Huttenlocher, J. Noh, W. Rucklidge, “Tracking Nonrigid Objects in Complex Scenes,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 93-101, 1993.

[45] K. Sato, J. K. Aggarwal, “Temporal Spatio-velocity Transform and Its Application to Tracking and Interaction,” *Computer Vision and Image Understanding*, vol. 96, no. 2, pp. 100-128, 2004.

[46] J. Kang, I. Cohen, G. G. Medioni, “Object Reacquisition Using Invariant Appearance Model,” in *Proceedings of the International Conference on Pattern Recognition*, pp. 759-762, 2004.

[47] G. Kitagawa, “Non-Gaussian State-Space Modeling of Nonstationary Time Series,” *Journal of the American Statistical Association*, vol. 82, no. 400, pp. 1032-1041, 1987.

[48] K. Levenberg, “A Method for the Solution of Certain Non-Linear Problems in Least Squares,” *The Quarterly of Applied Mathematics*, vol. 2, pp. 164-168, 1944.

[49] D. Marquardt, “An Algorithm for Least-Squares Estimation of Nonlinear Parameters,” *SIAM Journal on Applied Mathematics*, vol. 11, pp. 431-441, 1963.

[50] M.-C. Hu, M.-S. Chang, J.-L. Wu, L. Chi, “Robust Camera Calibration and Player Tracking in Broadcast Basketball Video,” *IEEE Transactions on Multimedia*, 2010.