

Executive Summary

Chinese Chess is one of the most popular board game in China and some other Asian countries. However, it is hardly to be accepted by western countries. That is because, unlike western chess whose piece is recognized by the shape, Chinese chess pieces are distinguished by Chinese characters that engraved on the piece, and most people in western countries cannot read Chinese characters. For the purpose of enabling people who cannot read Chinese characters to learn or play Chinese chess, this project looks at using vision to analyse the current state of the game and implements a system for real-time machine vision recognition of Chinese chess playing and reconstruct the board state in symbolic images. Also, the system could be used as vision system in chess-playing robot, or applied in some intelligent system which enables people playing Chinese chess against computer to create an interactive game in the natural world as opposed to via a screen.

The system use a camera mounted above the chess board, which will monitor the state of the chess pieces and board in real time. Then it will extract static pictures from video frames to detect static board situation. No prior knowledge of the chess rules is used by the system, so there are no assumptions on how and where particular chess pieces are allowed to move and the initial configuration of pieces on chess board is not necessary to be provided in advance.

MATLAB algorithm implementations and libraries are used for most of the image pre-processing and analysis, including the SVM classifier. The core technical points are: board calibration and piece recognition. This paper will review and compare the different solutions that have been done from previous literatures, and points out the weak points and strong points of each solution. Also, more robust and elegant algorithms are proposed and implemented in the research list as follows:

1. Corner-line matching algorithm in Chinese chess board detection is proposed and implemented. (Chapter 6)
2. A new Chinese chess piece recognition algorithm based on projection histogram in polar coordinates space and Fourier descriptor is proposed and implemented. (Chapter 7)

Table of Contents

Declaration	1
Executive Summary.....	2
Acknowledgements.....	3
Table of Contents	4
1. Introduction	5
1.1 Aim and Objectives	5
1.2 System Overview.....	6
1.3 Technique Overview.....	6
1.4 Significance of Research	7
1.5 Report Overview	7
2 System Framework	8
3 Comparison of Approaches.....	11
3.1 Modified Board/Pieces	11
3.2 Differential Image.....	11
3.3 Object Recognition.....	13
4 Review of Board Calibration Methods.....	14
4.1 Line-based Detection	14
4.2 Grid-based Detection	16
4.3 Corner-based Detection.....	17
4.4 Template Matching-based Detection	18
4.5 Pixel Classification	18
4.6 Square Detector	20
4.7 Cross Corner Detection based on Projection	20
4.8 Other Solutions	22
5 Review of Piece Recognition Methods	23
5.1 Template Matching Method	23
5.2 Ring Statistics	24
5.3 Recognition Based on Connectivity and Holes	27
5.4 Other Solutions	28
6 Proposed Board Calibration Algorithm.....	30
6.1 Algorithm Procedures	32
6.2 Result Analysis.....	41
7 Proposed Piece Recognition Algorithm	45
7.1 Algorithm Procedures	46
7.2 Result Analysis.....	55
8 The Software	57
9 Evaluation	59
10 Conclusion and Future Work.....	60
Bibliography.....	61
Appendix – Essential Source Code	64

1.Introduction

With the permit of technical conditions, people are constantly trying to find a more natural and convenient way to interact with machines. The development of IT technologies especially computer vision technique nowadays allows the way that people communicate with computer not only limits to interact through interface on screens. One of many ways that humans and machine can interact is through physical game play, using board games such as Chinese chess.

Chinese Chess is one of the most popular board game in China and some other Asian countries. It has a long history and can be date back to 9th century, which is a precious heritage of Chinese culture. Chinese chess is a two-player board game in the same family as Western chess. However, it is hardly to be accepted by western countries. That is because, unlike western chess whose piece is recognized by the shape, Chinese chess pieces are distinguished by Chinese characters that engraved on the piece, and most people in western countries cannot read Chinese characters. For the purpose of enabling people who cannot read Chinese characters learning or playing Chinese chess, this project looks at using vision to analyse the current state of the game and implements a system for real-time machine vision recognition of Chinese chess playing and reconstruct the board state in symbolic images. Also, the system could be used as vision system in chess-playing robot, or applied in some intelligent system which enables people playing Chinese chess against computer to create an interactive game in the natural world as opposed to via a screen.

In the near future, the capability of robots will be broken down into more detailed capabilities or functions. According to different functionalities, future robots can be classified into: entertainment robot, utility robot, immovable robots, movable assistant robot, and human-like robot. Chess-playing robot is one of the entertainment robots. With the assist of camera, the chess-playing between human and robot can be very similar to gaming between human beings which make the robot feels more humanized and friendly. The premise of developing such kind of robot is to let machine make a better sense of physical world. Computer vision technique plays a pivotal role in such kind of application. Physical board games are a rich problem domain for human-robot cooperation research, because such kind of games has an intermediate and easily adjustable degree of structure. This research looks at the vision part of chess robot.

1.1 Aim and Objectives

This project is aimed to reconstruct Chinese chess game-playing in physical world. To achieve this aim, the project looks at using a camera to monitor the chess board and using vision technique to recognize and analyse the current board state. The result

will be shown in screen in symbolic images to reconstruct Chinese chess game-playing in real time. To achieve the aims, following tasks should be accomplished.

- (1) correctly calibrate the chess board
- (2) automatically recognize the identity of each chess piece on the chess board
- (3) display the state of chess board in real-time

1.2 System Overview

The system use a camera mounted above the chess board, which will monitor the state of the chess pieces and board in real time. Then the system will extract static pictures from video frames to detect static board situation. No prior knowledge of the chess rules is used by the system, so there are no assumptions on how and where particular chess pieces are allowed to move and the initial configuration of pieces on chess board is not necessary to be provided in advance. MATLAB algorithm implementations and libraries are used for most of the image pre-processing and analysis, including the SVM classifier. The core technical points are: board calibration and piece recognition.

1.3 Technique Overview

As is mentioned, the project is mainly about how to recognize and track real chess game playing. Jason E. [1] concluded three major approaches to recognize chessboard: differential imaging, modified board/pieces, and object recognition. In this project, the third approach is adopted as it is a more natural and also more elegant way. Also, further research into that approach will yield big improvements to robotic chess. Chess recognition is a complex task; it involves application of various algorithms and methods from different fields such as computer vision, AI, machine learning and signal processing. Therefore, several interesting problems are raised, including board calibration and piece recognition which is also the most challenging part in this project. Various solutions have been attempted, some of them require a modified chess set or place unreasonable constraints on board conditions and camera angles. Using computer vision technique to automatically determine arbitrary chessboard location and identify chessmen on a standard, unmodified chess set is a more general solution, and many works has been devoted [2] to that. This paper will review and compare the different solutions that have been done from previous literatures, and points out the weak points and strong point of each solution. Also, new methods are proposed and implemented in the research such as: 1. Corner-line matching algorithm in Chinese chess board detection; 2. Chinese chess piece recognition algorithm based on projection histogram in polar coordinates space and Fourier descriptor. In this paper, we will describe and discuss the whole system that implements all phases needed for tracking the progress of the chess game, including extracting valid frames of individual moves from the video sequence,

calibrate the chess board, locating individual fields, segmentation of chess pieces on the board and recognizing the identity of each piece and its position.

1.4 Significance of Research

As researches on computer vision technique mature, the problem of chess vision becomes more engaging to academic community for researching on chess-playing robot based on computer vision. [3][4] The results of this research have a certain practical application value. As the low cost of the system, it will be easily promoted to the market. As is mentioned, the Chinese chess playing robot can also enable people who cannot read Chinese characters learning and playing Chinese chess and can help popularizing Chinese chess game. In addition, this research outcome can also be used to do secondary development or applied in some intelligent systems. By studying chess robot vision system, research achievements can also be generalized into some other areas, such as product inspection, quality control, agriculture industry.

1.5 Report Overview

This report first gives an overall introduction in Chapter 1, and then briefly discusses the system framework in Chapter 2. Chapter 3 gives a comparison of main stream approaches in chess recognition. Then, Chapter 4 reviews the previous solutions have been studied on how to calibrate the chess board. While Chapter 5 reviews the previous solutions have been studied on Chinese piece recognition. In Chapter 6 & 7, we introduce our proposed algorithm on Chinese chess board calibration and the chess piece recognition respectively. Chapter 8 provides a brief introduction to the system software. Chapter 9 evaluates the works have been done. Chapter 10 concludes the report and gives a suggestion on future works.

2 System Framework

Before moving on to the algorithm details of each module, this section gives the framework and working conditions of my system.

As is mentioned, we use a CCD camera mounted above the chess board to monitor the state of the chess pieces and board in real time. Then the system will extract static pictures from video frames to do further detection. Following figures show our device and software GUI.

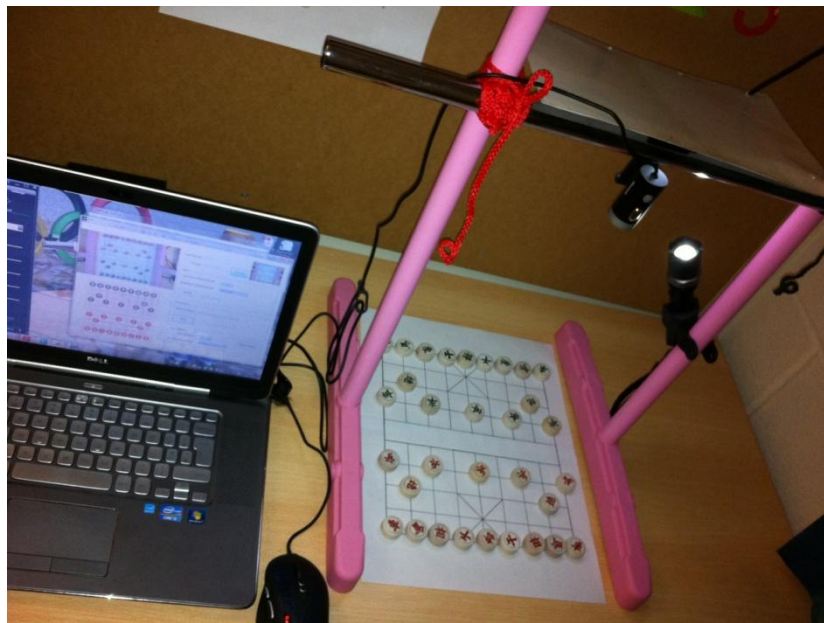


Fig. 2.1 System Device

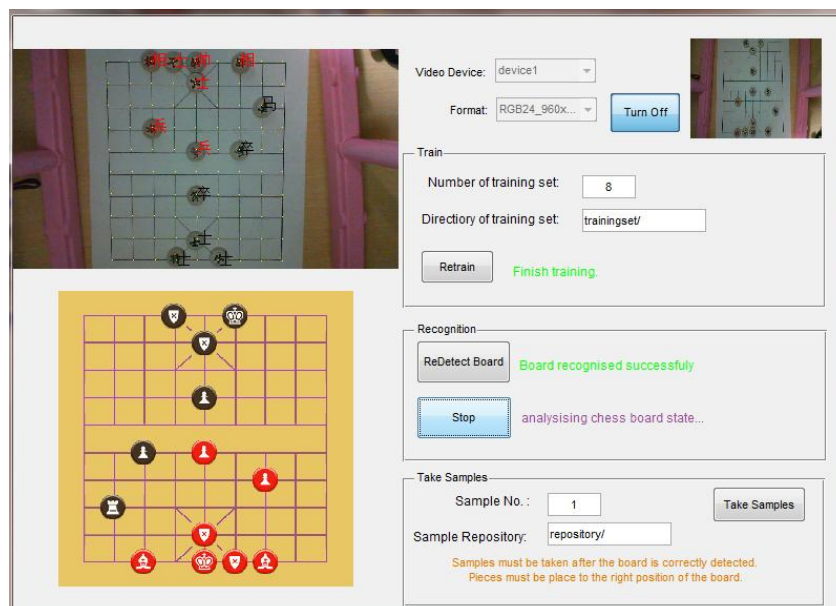


Fig. 2.2 Software GUI

The system architecture is illustrated as following flowchart.

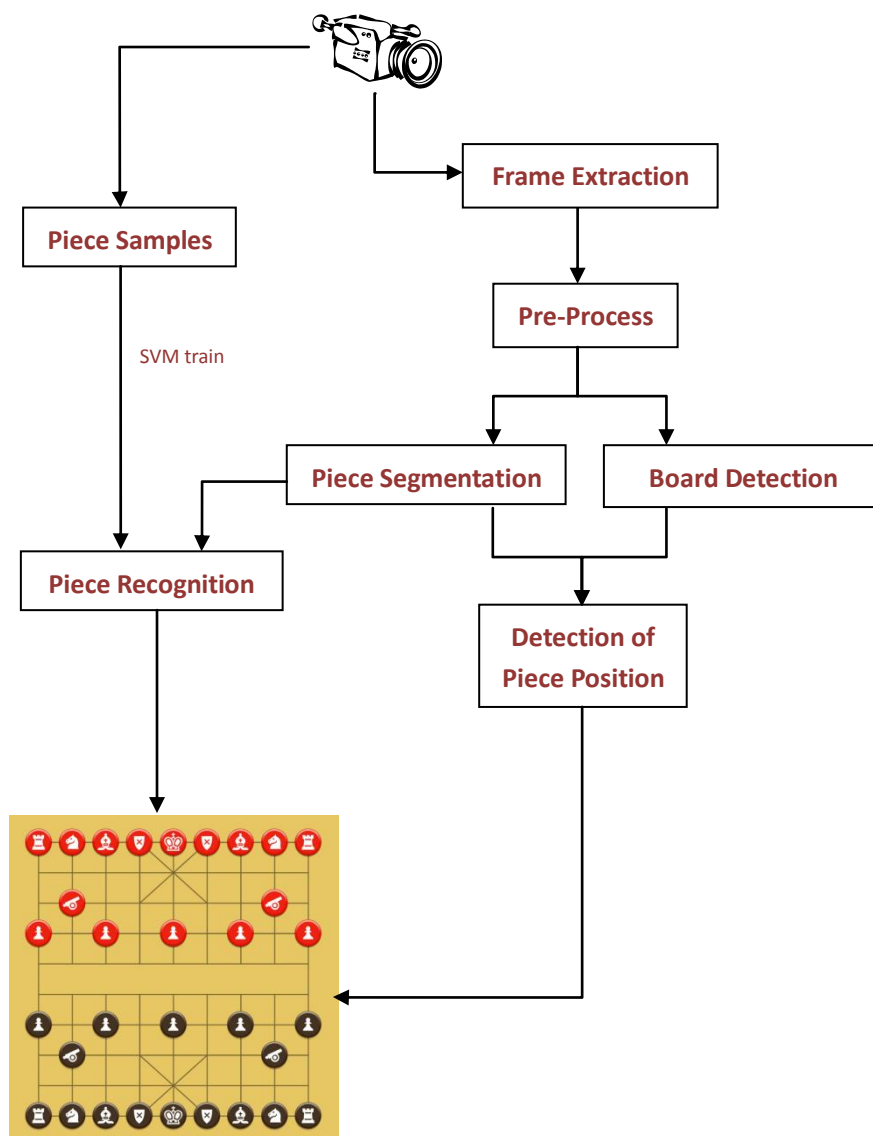


Fig. 2.3 System Flowchart

Here we give description on each module in details:

Frame Extraction

As the video frames receive the consecutive game-playing process, there must be a lot of useless image frames. While the player hand moves over the board to pick a piece, it is considered that a valid move has not been finished yet and those image frames should be ignored. The task in this stage is going to extract and detect the image frames with valid movement over the chessboard, then send notification to other components about the current board situation.

The valid static images of board situation can be detected by computing the image difference (by subtracting pixel values across entire images) of neighboring video frames. If the difference of image frames is below a specified threshold, then it is considered that the static image has been detected.

Pre-process

Before we beginning to detect chess board and pieces, image extracted from video frames must be pre-processed. For board detection, image is firstly changed from RGB format to gray-scale image and median filter is applied to remove noise. Then, we apply “Canny” edge detector to extract edges on the image for further detection. As for piece detection, we first transform image format from RGB to gray-scale. Also, median filter is applied to remove noise and “Canny” edge detector is applied to extract edges. To increase the rate of circle detection, we need to apply morphological algorithm on edge images. Here we adopt dilate operation. After circles have been detected, image should be binarized. To get image threshold, we adopt OTSU algorithm.

Board Detection

This stage is going to recognize and locate chess squares on the board, which is needed for further steps of processing and analysis. Different board detection algorithms will be given in Chapter 6.

Piece Segmentation

This step serves for the purpose to segmentation each chess piece from the image in order to recognize the piece identity in next step.

Detection of Piece Position

Piece position can be detected after calibrating chess board and piece segmentation.

Piece Samples

In order to train the classifier to recognize pieces, we must prepare a bunch of sample images of each type of chess pieces. The system provides the function to take piece samples.

Piece Recognition

In this module, we use SVM classifier to recognize chess pieces. Detailed algorithm and descriptor representation will be described in Chapter 7.

3 Comparison of Approaches

Many different solutions have been attempted in chessboard recognition problem. After background survey of existing approaches, Jason E. [1] summarizes three major solutions: modified board/pieces, differential image, and object recognition. This section will introduce each of the three methods and discuss their weakness and strong point.

3.1 Modified Board/Pieces

The most traditional way to implement physical chess-playing system is by the mean of pure electronic method which uses chess board and pieces modified or mount electronic sensors on the board or pieces. This is an efficient method, and individual piece can be recognized easily without mistake; electromagnetic boards can automatically move pieces. As it uses non-visual detection methods, it needs minimal computation complexity and can reach highest accuracy. However, it largely rely on external peripheral equipment, therefore the cost of the device will be high. Also, it requires changes to boards and pieces, which leads to that method only applies to tailor-made chess board and pieces instead of general ones.

One of the examples is a chess-play robot called Robochess made by Ebrahim Jahandar [5]. The robot has a special chess board with 64 Reed Relay(Magnetic Sensor) placed in the bottom of board, and every chess pieces have a small magnet in the bottom that's not visible so with magnetic sensitive board, the robot can track piece movements. This is an innovative method, as it based on magnetic force. However, unlike other similar robots, it doesn't use recondite methods such as computer vision for recognizing piece in chess board, so correct positioning of initial chess set is required.

3.2 Differential Image

This approach uses image process and simple computer vision method to infer moves on the board. [6] The core idea of this approach is to identify board changes by comparing before and after images to infer the move made by opponent. [7]

The process of differential imaging is divided into following three steps (according to [1] and [8]):

Step 1 Board Detection

In order to identify piece position, we must know board position and segment the image of a chessboard into chess squares. The simplest method is to mount the camera directly above and orthogonal to the board. However, this constraint leads to

less user-friendly. To detect board from arbitrary camera view is necessary. Tam, et al., has proposed a method of board detection at a flexible camera angle with little constraint on camera view. [9] There is a bunch of different method existing to deal with that problem. The most frequently utilized method is Hough transform. [10] As the problem of board detection is one of the big topics, I will discuss the issue in details in Chapter 4.

Step 2 Chess Squares Analysis

We assume here that we have already segmented the image into a matrix of squares. Now we are going to analyze each square to identify the pieces. In order to do that, we need to consider the following two questions: [8]

- (1) Is there a piece on the square?
- (2) If yes, what color is it?

Both of the questions can be answered by using thresholds depending on whether the square is dark or light. And different average threshold is used to determine whether a black piece or red piece is located on that square.

Step 3 Piece Identification

Piece identification is based on the square states of two adjacent chess board images. In order to determine the change on the board after one players move, we can subtract the pixel values of two adjacent images. From the difference of two adjacent chess board squares, we can infer the pieces from piece identities of last state. As you may expected, differential imaging approach does not actually recognize each piece. Instead, this method infers each piece by the changes happened on the chess board squares. This is the key idea of differential imaging approach.

The primary advantages of differential imaging approach are that it is the simplest implementation which doesn't need extra electronic devices and complex algorithms, and also, this approach works with almost any chess set.

As this approach infers the identity of each piece by comparing the between images before and after a move rather than truly recognizing each piece, so this method has low flexibility and any mistake happened in the interim of game playing will lead to breakdown of the whole situation. Also, as we distinguish each square state by using thresholds to compare with the lightness of squares which is much susceptible to lighting variations, so the stability is easily affected by the lighting condition. In addition, this approach requires that the game must started from the beginning, as computer needs to know the identity of each piece according to their initial position. However, if a game starts in an arbitrary position, it will require manual entry of piece locations into the chess engine. This is the weak point of differential imaging approach.

3.3 Object Recognition

The object recognition approach shares commonality with the differential image method in the areas of board detection. [1] They differ in the step of piece identification.

Object recognition mainly uses computer vision and pattern recognizing means to really recognize each piece rather than to reason each piece by using the differentiation. This is the most challenging and complex approach. And there is a plenty of methods can be used to implement object recognition. Meanwhile, further research into that approach will yield big improvements to robotic chess. Also, most of the methods prefer the camera view at a perspective angle rather than a simple top-down view, in order to recognize the identities of the pieces effectively. In addition, almost all of the current recognizers need to be pre-trained or told the features of each piece.

Drawbacks include the complexity of the implementation of recognizing algorithm and the requirement of pre-training of classifier. Meanwhile, the advantages of this method are obvious. This approach can recognize individual pieces; the system can be used with arbitrary board configurations, which provides more flexibility; arbitrary chess sets are supported, which ensures the adaptability; less constraint on camera view angle, which simplifies camera mounting and setup; move of camera during the chess-playing are allowed, which reveals the better robustness. The advantages outweigh the disadvantages of this approach. Comparing with the previous two solutions, object recognition appears to be more worthwhile to be researched and discussed. As this method allows building a robust system, and is easier to setup due to the simplicity of hardware. Therefore, this approach will be adapted and implemented in this project.

As object recognition is a big topic, a more specific and detailed description of different algorithms and ideas on this topic will be given in Chapter 5.

4 Review of Board Calibration Methods

Before any further processing, the problem that confronts us is chess board calibration. In this step, we should detect squares on the chess board from video frame and map the chess board to the chess notation. In order to let our vision algorithm to adapt to any kind of chessboard, the initial chessboard segmentation must be capable of recognizing the squares on chess board under the following situations:

- (1) Various square sizes
- (2) Various chessboard orientations
- (3) Various light conditions
- (4) Some squares may be partially occluded
- (5) Noise may appears on the chess image

Various approaches have been attempted to deal with board detection. In [1], a Hough transform-like technique is used to detect line in order to calibrate board squares. In [9], the grid detection method is proposed. In [11], corner detection method to locate board. In [12], the system use an OpenCV embedded function to detect chess board. In [13], pixel classification method is adopted to determine the class to which each pixel belongs by means of the (color) attributes of these pixels. In [8], a template matching based method was used to detect the corner of the grid. Various approaches used to calibrate chess board are generally divided into three groups:

- (1) corner-based detection
- (2) line-based detection
- (3) pixel classification

This section will discuss all of those approaches and compare the strong points and weak points of each approach.

4.1 Line-based Detection

Line-based detection method works by detecting the lines of the chessboard. Then use the line equations to form squares on the board. Line-based detection can be divided into following steps (Figure 4.1).

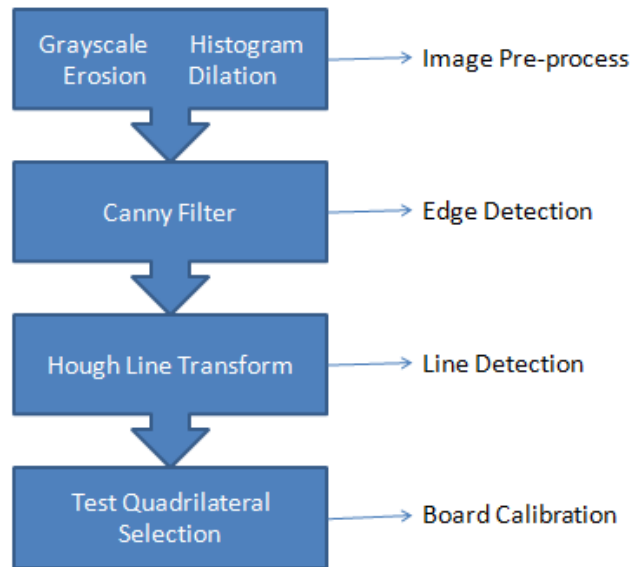


Fig. 4.1 Major steps used in line-based board detection algorithm

Step 1 – Image Pre-process

Before formally beginning to apply line-based detection approach, some basic filters must be applied to the image first, so as to make the detecting process more efficient and reduce noise. According to Jason E. [1], following filters may be needed:

- (1) A grayscale copy of the input image is used for both line detection and final board comparison;
- (2) The image histogram is equalized to normalize brightness and increase contrast;
- (3) Erosion and dilation filters are applied in succession to smooth out noise in the image and to reduce unnecessary lines from being detected. "

Step 2 – Edge Detection

Before line detection, some form of edge detection should be applied on the filtered image. The most commonly-used edge detection algorithms are Canny edge and Marr-Hildreth detectors. Mohsen Sharifi, et al., [14] gives description and comparison of various edge detection algorithms.

Step 3 – Line Detection

To detect lines on chess board, Hough line transform [10] is the most frequently utilized method.

Step 4 – board calibration

Square on the chess board can be detected by following steps: [1]

- (1) Match a horizontal and a vertical line that intersect in the upper left quadrant;
- (2) Select a vertical line that intersects with the same horizontal line in the upper right quadrant;
- (3) Select a horizontal line that intersects with the last vertical line in the lower

right quadrant.

- (4) The lower left intersection is found using the first vertical line and the last horizontal line.

The main advantages of line-based detector is that the detection probability is higher comparing with corner-based detector (which will be discussed later), especially when they are applied to calibrate populated chess board taken from a perspective angle view, as it is unlikely that a line is completely blocked by chessman. Also, this method is less sensitive to noise, since a line will have significantly higher count than isolated pixels.

However, the line-based detection method is sometimes unstable due to image distortion (lines are not straight). Although line-based technique works better than corner-based technique, the method alone will not perform well when the occlusion is high enough so that lines are not guaranteed to exist. To improve the reliability, Tam, et al., has proposed a grid detection algorithm to make better prediction, which combines line-based grid detection with knowledge of the chessboard. [9] The detailed description will be given in next section.

4.2 Grid-based Detection

Grid detection method [9] is proposed to overcome the occlusion problem. As when we use pure line detection method on a populated chess board from a lower angle view, the visibility of a number of lines will not be guaranteed. However, we can predict the missing lines by using the domain knowledge of the chessboard. Grid detection method is based on following four knowledge of the chessboard:

- (1) A chessboard has none lines in horizontal and vertical direction respectively.
- (2) Despite of the existing of perspective distortion, the size of neighborhood grid should be roughly approximate.
- (3) For any square within the grid, there will be intersections where board lines meet.
- (4) Each square contains the complete geometric information of the whole chessboard. If the edge of the board is known, only a single square is needed to grow into a complete board. Figure 4.2 illustrate the how the square above the current one can be constructed using simple construction lines.

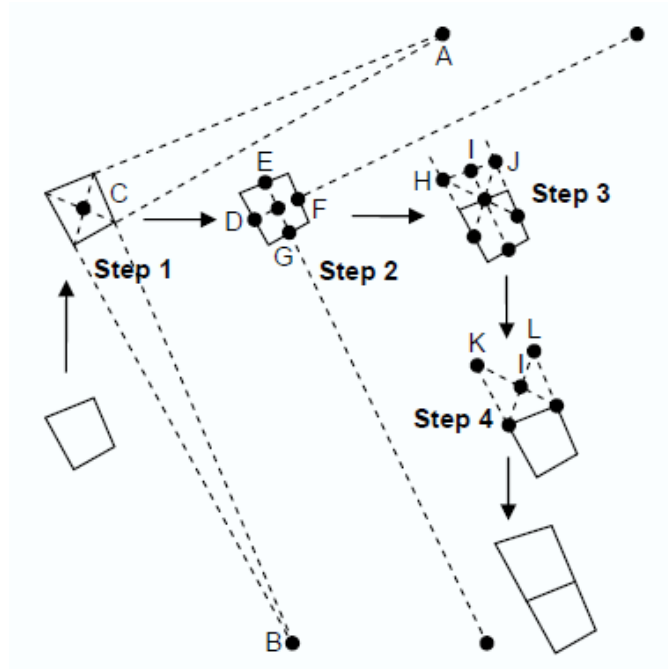


Fig. 4.2 Steps showing how the square above the current one can be constructed using simple construction line

Grid-based detection nearly eliminates false positives and misses, which is a big improvement to line-based detection method.

4.3 Corner-based Detection

An alternative method to calibrate chessboard is corner detection. Typical corner detection methods include Harris corner detector, SUSAN [15] corner detector (a method based on the smallest uni-value segment assimilating nucleus) and template matching-based [8] corner detector (which will be discussed in section 4.6).

Harris corner detection and SUSAN corner detection methods are quite similar. Both of the methods compare each pixel to its surrounding to check if it is a corner. However, the two methods differ in detection scale. Harris corner detection method compares a small neighborhood with its immediate neighbor, whereas SUSAN method compares the current pixel with its surrounding. [9] An example of applying corner detection in chess-playing system can be found in [11].

Comparing with line-based detection technique, the main advantages of the corner-based detector is high resistance to against camera distortion, as a corner can hardly be affected by image warping.

Whereas, the weak points are also obvious, corners are not guaranteed to be detected on a populated chessboard from perspective camera view. Moreover, the corners returned would have no information as to which points are co-linear with other points. Generally speaking, for most system, line-based detection algorithm

works better than corner-based detector.

4.4 Template Matching-based Detection

Template matching-based detection method (proposed by T. Cour, et al. [8]) is a kind of corner-based detection method. We use templates to detect the corners of the chess square. An example template is show as following Figure 4.3.

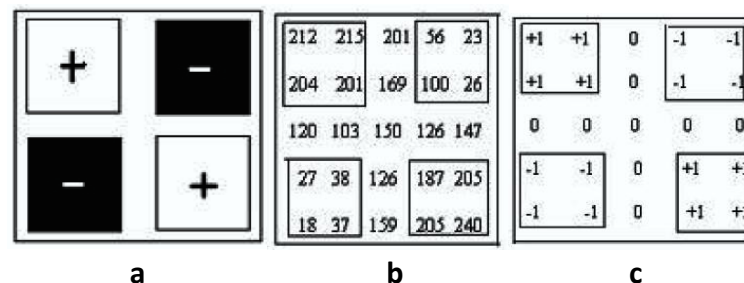


Fig. 4.3 sample template

Figure 4.3 (a) shows a corner in between four squares. Figure 4.3 (b) shows typical values of grey levels around a corner. Figure 4.3 (c) shows the corresponding template with 5x5 coefficients. To deal with light condition, we use a threshold. As grey level values are seldom significant at the frontier between two squares, the coefficients of the template in the two median lines are equal to zero. Any pixel that matches those templates will be detected.

Now, we have a list of corners. But, we don't know squares, and those corners are not ordered in a matrix. Corners on the border of the chess board are usually not detected. Due to light conditions, some corners may not have been detected. Also, some corners may be detected by mistake, due to the noise. In order to deal with the above issues, the algorithm will search for maximum alignments of corners in both horizontal and vertical direction and then eliminate corners that don't fit in those alignments. Also, square size can be calculated by computing the average distance between two consecutive lines with the same direction.

4.5 Pixel Classification

David U. [13] uses pixel classification method in his chess-playing robot to recognize chess board. As David U. describe, "Pixel classification attempts to determine the class to which each pixel belongs by means of the (color) attributes of these pixels."

Step 1

Before being able to classify pixels, color domain for each class should be computed. We define four classed: (1) light square (2) dark square (3) light piece (4) dark piece. However, the classes 'light square' and 'light piece' overlap, when pixels are represented in RGB color space, which will lead to misclassification. To deal with that problem, we therefore switch the chessboard image from RGB color space to a HSB (Hue, Saturation, and Brightness) color space. After computing the color domain of

each class, we then classify all pixels in the image. A sample result of this step is showed in Figure 4.4.

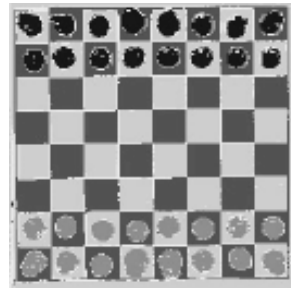


Fig. 4.4

Step 2

Now, we are going to calibrate chessboard. To position the board and pieces in the image, we firstly extract a reference image that has the same color as the dark squares, and then the algorithm performs a search at the extremities of the image for pixels belonging to the 'dark square' class to detect corners. Based on the knowledge that all squares are equal in size and there are 8 squares for each dimension, we can then easily determine the position of each square by means of interpolation. A sample result is illustrated in Figure 4.5

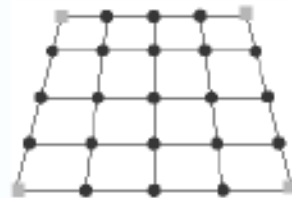


Fig. 4.5

Step 3

After determining squares, we can now analysis each square to detect whether a square is occupied by a piece and, if yes, what color is the piece. This can be achieved by calculating the number of pixels within a square that belongs to the class 'light piece' and 'dark piece' and compare the number to a particular threshold to decide if a piece on that square.

Step 4

Now, we get a list of squares and know whether they are occupied by a chess piece of a particular color. However, we still don't know the type of each piece on the squares. In order to determine the type of each piece, we can simply compare current situation with the situation before the last move, which is similar to the method of 'differential image' in Section 2.2.

This is a simple and robust method for recognizing board situations. Whereas, there are some constraints such as, the camera must be mounted about the chessboard; the algorithm requires that the game must started from the beginning, as computer needs to know the identity of each piece according to their initial position, but if a game starts in an arbitrary position, it will require manual entry of piece locations

into the chess engine.

4.6 Square Detector

In [16], the square detector is used to find chess squares. To cope with the rotation and scale, the system generates 20 square templates by rotating a square in different angles. We can then extract histograms of oriented gradients (HOG) features and obtain 20 HOG square training templates. In addition, different chess board and different view heights will result in different image scale. To fit with different scaling level of the squares, the system should chose the image scale automatically during the test time to ensure that square size in the image corresponds approximately to the size of square training templates. To do that, we firstly generate an image pyramid with different pre-specified scales, and then apply the square detector to the image pyramid to find the overall-most likely scale level.

4.7 Cross Corner Detection based on Projection

After image binarization, we can find it consists of lines, circles, and characters. Based on the method of Hough transformation as 3.1 describes, we design an algorithm to find 9 x 10 cross corners. It is proven by experiment that Hough transform method is easily affected by chess pieces on the board and it will produce some noise line. Also, the speed of calculation cannot satisfy real-time detection. Therefore, Fenglei Xu [17] in his thesis proposed a Chinese chess board corner detection method based on project.

Here is a brief description of the rationale. After preprocessing the image, we scan the image in horizontal direction from up to down and record the number of pixels of each line. After applying the projection method, we can get the distribution shown on Figure 4.6. As we can see, the black bar represents the number of black pixels on each line. The peaks on the image represent the board lines. Therefore, if we can find those peaks on the distribution, the horizontal board lines can be calibrated.

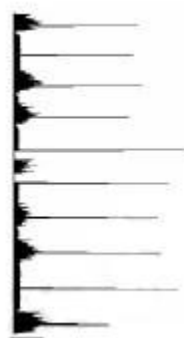


Fig. 4.6 Horizontal Projection

On the basis of above description, the detailed algorithm is elaborated as follows.

(1) Define an array (dis[10]) to record coordinate; define a array

- (ptempdata[2][height]) to record the number of black pixels on horizontal lines and its corresponding coordinate.
- (2) Scan each horizontal line, count the number of black pixels, and store it and its corresponding coordinate in the array (ptempdata). In order to decrease the calculation, we can set a threshold and discard data that is less than the threshold. The number of data is recorded in “count”.
 - (3) Sort the data in order.
 - (4) Get the first coordinate $dis[0] = ptempdata[0][0]$ (i.e., the horizontal coordinate with the maximum black pixels). On the base of this coordinate, the data in second line (ptempdata[0][1]) is compared with it. If the result is bigger than $height/15$, then it is regarded qualified; otherwise, continue to compare the data in third line, until we get the second coordinate $dis[1]$. Afterwards, we compare all of the data with the former one, record those that satisfy the condition in “dis” until 10 coordinate are found.
 - (5) Sort “dis” in order to prepare for the next steps.

After computing by the above steps, we can get the 10 horizontal coordinates of the board. During the experiment, we have process a bunch of images. In some cases, we cannot find 10 qualified coordinates. Therefore, when we discard useless points, we adopts adaptive threshold to solve the problem.

After we get horizontal coordinates, a similar algorithm can be applied on detecting vertical coordinates. However, due to the problem of viewing angle, the image will be deformed to some extent. The vertical projection result of the whole image is shown in Figure 4.7 (a). As we can see, there exists big noise. As the coordinate of each vertical coordinate is different from each other, we can only do the projection separately. The result is shown in Figure 4.7 (b).

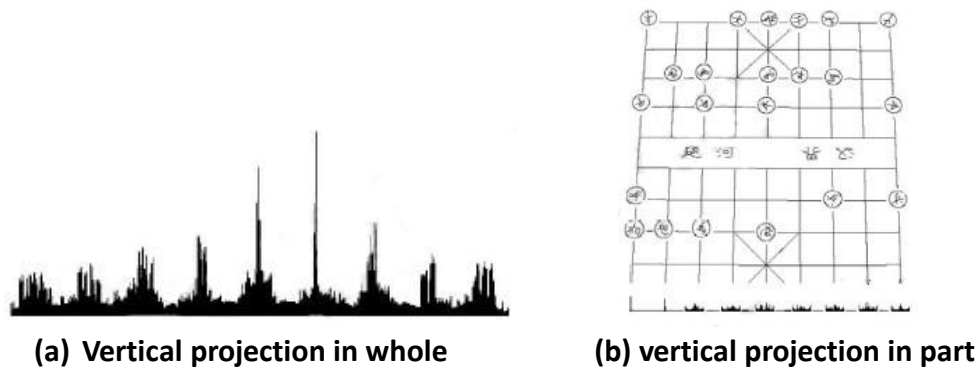


Fig. 4.7 Vertical projection

Synthesizing the above two steps, the cross-corner detection strategy can be concluded as:

- (1) Project the image in horizontal direction to get horizontal coordinates and save the coordinates in *hlinedis*[10];
- (2) Separate the image from top to bottom according to the results in (1).
- (3) Project the separated images in vertical direction to get the vertical coordinates and save the coordinates in *vlinedis*[9].

Experiment results show that this method to calibrate the chess board is highly affected by the position of chess board relative to the position of camera.

4.8 Other Solutions

In addition to the methods described in preceding sections, there also exist many other solutions. For instance, Jose G. [16] in his robotic system adopts a method that requires user to click the four chessboard corners in the image. Then, chess squares can be calculated automatically, based on the knowledge of introduced corners. In Chapter 6, we will propose a more robust and efficient Chinese board calibration algorithm.

5 Review of Piece Recognition Methods

Several solutions can be used to detect a chess piece. In Section 3.2 (differential image), we have discussed a method of inferring the type of chess piece by simply comparing current situation with the situation before the last move. But, this is not a robust and flexible method, as it has many constraints on the system. This section introduces more elegant algorithms based on pattern recognition [18] [19] technique. In order to recognize chess pieces correctly and efficiently, the key factor is how to represent the features that could distinguish different type of pieces, i.e. what is the descriptor.

5.1 Template Matching Method

The main idea of template matching [21] is to searching the corresponding pattern in an image based on a known pattern on another pre-prepared image. The aim of template matching is to check up whether the pattern image is existed on current image and to get the position of pattern image on current image. The most frequently used method of template matching is pattern matching. It adopts the gray-scale image of an object as known template and search the known pattern on an image.

Template matching is image processing technique based on 2-d image window. For instance, the template T ($n \times n$ pixels) is stacked on the searching image S ($w \times h$ pixels) and the overlap area is called sub-image S_{ij} . (i, j) is the top-left coordinate of the sub-image on S . The searching range will be:

$$1 \leq i \leq W - M$$

$$1 \leq j \leq H - M$$

The process of template matching will be done by comparing the similarity of T and S_{ij} , which can be measured by following formula:

$$\begin{aligned} D(i, j) &= \sum_{m=1}^M \sum_{n=1}^N [S^M(m, n) - T(m, n)]^2 \\ &= \sum_{m=1}^M \sum_{n=1}^N [S^M(m, n)]^2 - 2 \sum_{m=1}^M \sum_{n=1}^N S^M(m, n) \times T(m, n) \\ &\quad + \sum_{m=1}^M \sum_{n=1}^N [T(m, n)]^2 \end{aligned}$$

In the formula, the first item is the energy of sub-image; the third item is the energy of template. They are irrelevant to template matching. The second item is the cross-correlation between the template and sub-image and is changed along with (i ,

j). When matching the template with sub-image, this item has a maximum value. If we normalise this item, we can get correlation coefficient:

$$R(i, j) = \frac{\sum_{m=1}^M \sum_{n=1}^N S^M(m, n) \times T(m, n)}{\sqrt{\sum_{m=1}^M \sum_{n=1}^N [S^M(m, n)]^2} \sqrt{\sum_{m=1}^M \sum_{n=1}^N [T(m, n)]^2}}$$

When the template is completely the same with sub-image, the correlation coefficient $R(i, j) = 1$. When finishing searching on image S , we will get a maximum value of $R(R_{\max}(i_m, j_m))$, then its corresponding sub-image S_{i_m, j_m} is the matching result. Obviously, when applying this formula to do image matching, the amount of computation is fairly large and the speed will be low.

Another formula to measure error between T and S_{ij} is:

$$E(i, j) = \sum_{m=1}^M \sum_{n=1}^N |S^M(m, n) - T(m, n)|$$

Where $E(i, j)$ get to its minimum is the matching result. In order to increase computing speed, we define an error threshold E_0 . When $E(i, j) > E_0$, we can stop calculation on that position and begin to compute in next position.

The result of template matching method is proven to be efficient. However, as chess piece has no direction, this method has its limitation. We must put the chess pieces in one direction to make sure the characters on the pieces are shown up-right.

5.2 Ring Statistics

There are 11 different types of characters on Chinese chess piece, shown as follows.



Fig. 5.1 Chinese Chess Pieces

In real-time chess recognition system, it requires that speed of recognition must be high enough and those methods with complex computation are discarded. Therefore, we are looking for simple, fast, efficient algorithms that are specific to those 11 distinct characters. Zhongkai Xiao in his postgraduate thesis proposed a method of ring statistics to recognize chess pieces. This section will describe this method in details.

By the feature of outline border on chess pieces, we can easily segment each piece from the binary image by template circle matching preparing for character recognition. As the randomness of character orientation, the key factor to recognize each character is the direction. Here is the main idea of this algorithm. We first find the centre of a character and use this as the centre to build concentric circles, whose radius are $R/4$, $R/2$, $3R/4$ (R is the radius of a chess piece) respectively. At the mean while, we count time of intersections between each circle and the character. The

statistical result is used as code to build the descriptor to classify each character. This method obtains the structure information of characters and is irrelevant to the orientation. The idea of this algorithm is shown as Figure 5.2.

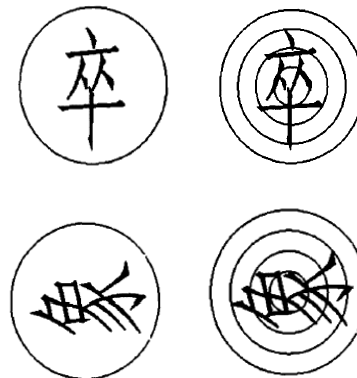


Fig. 5.2 Ring statistics diagram

The flow chart of ring statistics algorithm is shown as follows.

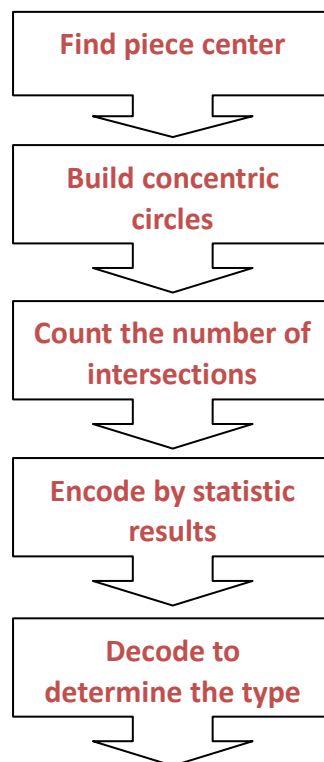


Fig. 5.3 Ring statistics flow chart

The first two images in Figure 5.2 represent the black pawn in Chinese chess. As we can see, the numbers of intersection between concentric circles and the character are 5, 6, and 1 respectively. Therefore, black pawn can be encoded as 561. Similarly, the last two images in Figure 5.2 represent black bishop in Chinese chess, and the numbers of intersection between concentric circles and the character are 7, 8, and 4 respectively. Therefore, black pawn can be encoded as 784. According to this rule, we can easily get a character code table shown as follows.

Table 1 – Character Code

	Chinese Characters	R/4	R/2	3R/4	code
Pawn(black)		5	6	1	561
Pawn(red)		5	4	3	543
Bishop(black)		7	8	4	784
Bishop(red)		6	8	2	682
Cannon		2	6	3	263
Guard(red)		3	5	1	351
Guard(black)		4	2	0	420
King(red)		4	8	1	481
King(black)		6	8	4	684
Horse		7	6	2	762
Rook		6	4	4	644

5.2.1 The Method of Counting Intersection

To count the number of intersections between each concentric circle and the characters, we first get the centre (x, y), and the radius (r), and then count along the circle anticlockwise starting from (x, y+r). To get pixel coordinate on the circle, we can apply the famous Bresenham [22] circle drawing algorithm.

5.2.2 Encoder Construction

If the radius of the outer-boarder of the chess piece is r , we should then count the number of intersection between the character and circles with radius ranging from 1 to r and we will get a result string. In order to get a minimum-length string that can represent tell each character, we must analysis and sift the string. The method of sifting is: if we delete a circle and the code of the rest of the string can distinguish each character, then we delete the circle; otherwise, we retrain that circle.

In this case, the result of sifting is the three circles whose radiuses are $R/4$, $R/2$ and $3R/4$ for each. Deleting of any circle will result in repetition of codes.

5.2.3 Experiment Analysis

The experiment result shows that this algorithm is efficient if we the light condition is good enough. Otherwise, pieces will be mistakenly recognized. Ideally, the code of horse is 762, be it will produce unexpected result in poor light condition shown as Figure 5.4.



Fig. 5.4 binary image of “Horse” in poor light condition

5.3 Recognition Based on Connectivity and Holes

Another recognition algorithm of Chinese chess pieces is based on connectivity and amount of holes of Chinese characters. [23] The so-called connectivity of a character means the number of connecting areas (8-connectivity) that are contained in a character; the so-called amount of holes means the number of connecting areas that are contained in the background of a character.

The problem of how to compute the number of connecting areas and holes can be translated into area statistics problem. We define the color of connecting areas as 0 and the color of holes as 1. Connecting area statistics algorithm is described as follows.

Algorithm Input: *color* - the color of connecting area; *count* - the counter of connecting area

Algorithm Step:

- (1) Initialize: set the initial position of scanning image to (0, 0); set *count* = 0;
- (2) If finish scanning, then terminate; otherwise go to (3)
- (3) Scan the image of chess piece, and find a pixel (x, y) whose value equals to *color*
- (4) Let the pixel (x, y) be a seed pixel, make *color* as original pixel value and make 1-*color* as new pixel value. Call the Flood-Fill Algorithm [24] to fill in the area where the pixel (x, y) located whose value equals to *color* by using pixel value 1 – *color*.
- (5) Add 1 to *count*; go to (2)












The algorithm of recognizing Chinese chess pieces based on connectivity and amount of holes of Chinese characters is described as follows.

- (1) Count the number of connecting areas: call the connecting area statistics algorithm with the parameter *color* set to 0 to get the number of connecting areas of black pixels.

- (2) Count the number of holes: call the connecting area statistics algorithm with the parameter *color* set to 1 to get the number of connecting areas of white pixels.
- (3) Build the feature code: combine the results from (1) and (2) to a string to match with standard identification code.

Following table shows the identification code of each character.

Table 2 – Character Code

	Chinese Characters	Number of connecting areas	Number of Holes	code
Pawn(black)		1	3	13
Pawn(red)		3	1	31
Bishop(black)		1	3	13
Bishop(red)		2	3	23
Cannon		5	1	51
Guard(red)		2	0	20
Guard(black)		1	0	10
King(red)		3	0	30
King(black)		6	1	61
Horse		5	3	53
Rook		1	4	14

Experiment result shows that this algorithm is fast and has a high reliability, but the feature codes are highly restricted to specific character font.

5.4 Other Solutions

Chinese piece recognition is a typical rotatable Chinese character recognition problem. The key factor to recognition is to extract the rotation invariant feature vectors. In addition to the algorithms that I mentioned, there are also many other solutions to that problem, among which, relatively mature methods are including Hu [25] and Zernicke [26] moment feature which are based on methods of algebraic invariants. Some other method firstly obtains the contour points and then adopts Fourier transform. All of them have been proven by experiments to be less resistant

to interference and less robust. Therefore, the proposed Chinese chess piece recognition algorithm based on projection histogram of polar coordinates spaces and Fourier descriptor will be elaborated in Chapter 7.

6 Proposed Board Calibration Algorithm

In this Chapter, we will introduce our proposed Chinese board detection algorithm which is based on corner-line matching method. In the end of this section, an analysis on experiment result will be given. Also, we will give a comparison between corner-line matching chess board detection algorithm and previous works that have been done.

Before we starting our topic, let's first have a look at the pattern of Chinese chess board. Figure 6.1 shows a standard Chinese chess board. As we can see, a standard Chinese chess board consists of 9 horizontal lines and 10 vertical lines. (Although the vertical line is split in the middle of the board, we still regard it as a single line) Our corner-line matching algorithm is designed based on that information. Also, there are also some additional lines or features in the chess board, but in our algorithm they are not necessary.

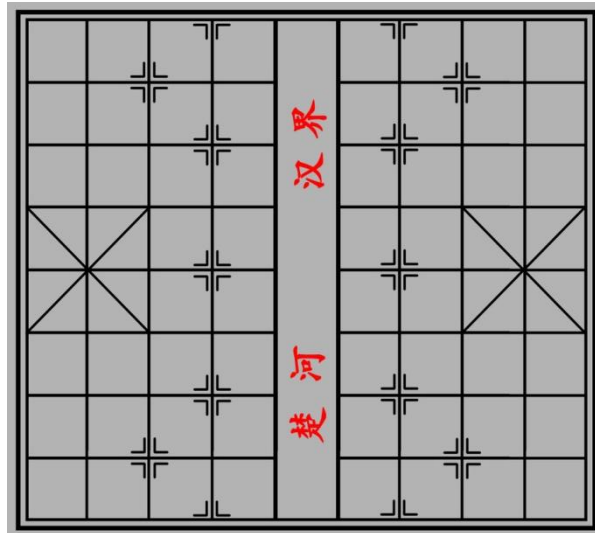


Fig. 6.1 Standard Chinese Chess Board

When we apply pure line-based algorithms (introduced in Chapter 4.1) or pure corner-based algorithms (introduced in Chapter 4.3) to calibrate the chess board, the results are fairly unstable and less robust, which are easily affected by the environment such as light sources, background stuff, board position and shadows on the board. That is because, when we apply line detection algorithms such as Hough transform or corner detection algorithms such as Harris corner detector, the noise is inevitable. A bunch of “fake” lines and “fake” corners will be detected, at the mean while, some lines and corners that are supposed to be existed might be missing in the result of detection.

Figure 6.2 shows the original chess board images and the results of applying Hough transform to detect lines in different cases.

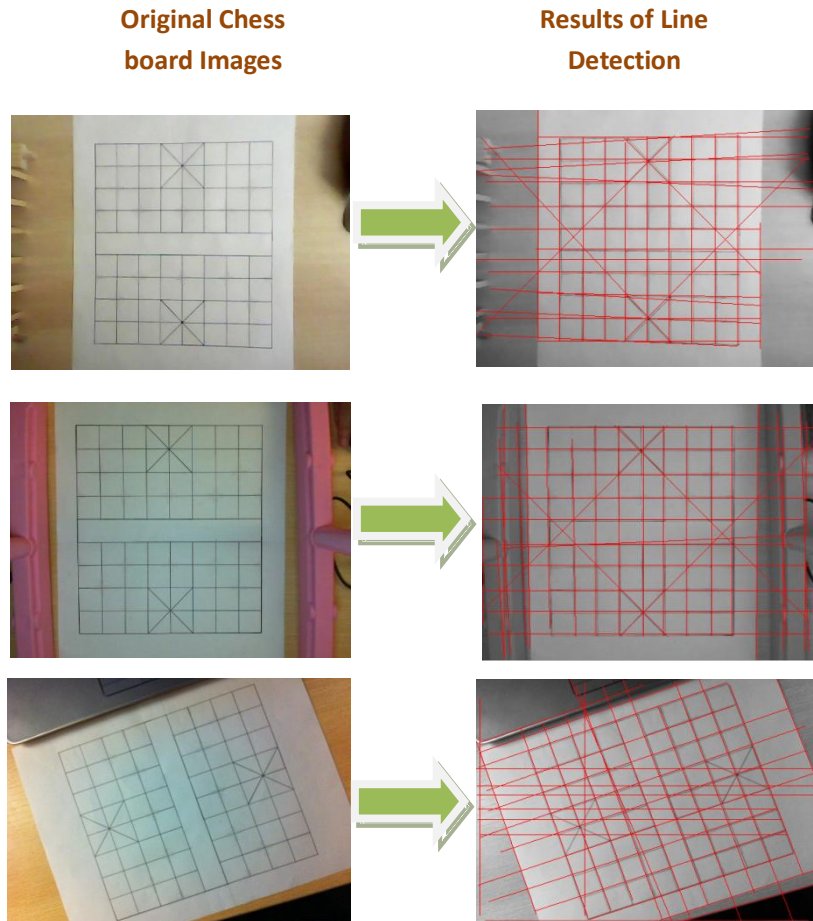


Fig. 6.2 Chess Board Images and the Results of Line Detection

Figure 6.3 show the original chess board images and the results of applying Harris corner detector in different cases.

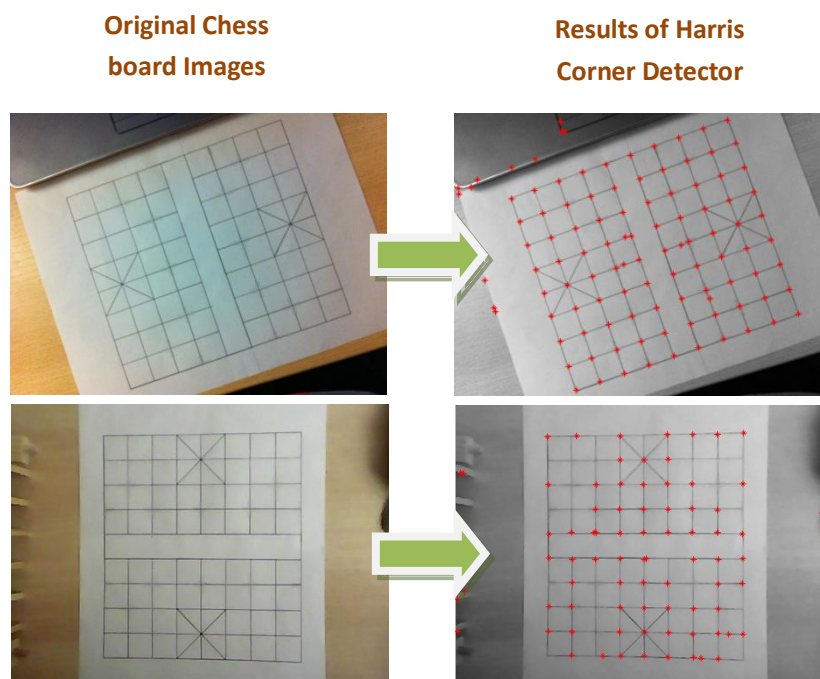


Fig. 6.3 Chess Board Images and the Results of Harris Corner Detector

As we can see from the results, the detecting result is easily affected by environment. Especially in real-time environment, the result will change all the time. In order to build a high fault tolerance algorithm, we make use of following basic rules that are applicable in standard Chinese chess board:

Rule 1: There are 9 horizontal lines in Chinese chess board.

Rule 2: There are 10 vertical lines in Chinese chess board.

Rule 3: Valid board corners in Chinese chess board must be intersected with at least two valid board lines.

Rule 4: Valid board lines in Chinese chess board must pass through a bunch of valid board corners. (We set the threshold of the number of valid board corners to 4 in our case)

Rule 5: The intersection angle between horizontal lines and vertical lines must be restricted within a threshold. (Here we set the threshold to 90 ± 10)

Rule 6: The intersection angle between lines in same direction (horizontal or vertical) must be less than a threshold. (Here we set the threshold to ± 10)

The idea of the algorithm is roughly described as follows:

Step 1: Pre-process.

Step 2: Detect lines by Hough Line Transform.

Step 3: Detect corners by Harris corner detector.

Step 4: Select “valid” board corners. (Passed by at least 2 lines)

Step 5: Select “valid” board lines. (Come across at least 4 “valid” corners)

Step 6: Select four boundary lines of chess board.

Step 7: Compute the four boundary corners by Cramer’s rule.

Step 8: Determine the direction of board.

Step 9: Compute all of the inner board corners by bilinear interpolation.

Each of the steps will be expounded in details in following sections.

6.1 Algorithm Procedures

Step 1: Pre-Process

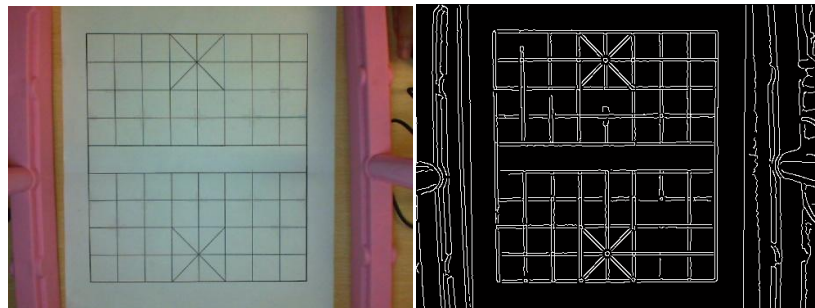
Before we move on to next steps, the chess board image should be processed in advance. Detailed steps are list as follows.

- (1) Transform the original image to gray-scaled image
- (2) Remove noise: As we adopt normal web camera, the image quality is not good enough and noise is obviously existed. In order to increase the detection effect, median filter [27] are applied to remove noise.
- (3) Resize image: For robustness purpose, this algorithm is design to fit different image format and size. However, if the size of image is big (e.g., the size of image captured from iPhone camera: 1936x2592), the speed of detecting cannot be tolerated. That is mainly because the computing time of Hough transform to detect lines will increase dramatically along with the increase of image size.

Therefore, we resize the image to 500 by 500 pixels and in the final stage, when we finish calibrating board, the detecting result will be restored to fit the original image.

- (4) Edge extraction: As Hough transform works on binary image, so before doing line detection we should extract edges from image and change it into binary image. There exist many mature edge extraction algorithms including Canny operator, Sobel operator, Laplace operator, Robert operator. After comparison of experiment results, Canny operator performs the best in our case.

Figure 6.4 shows the original image and the result of edge extraction.



a. Original image

b. result of edge extraction

Fig. 6.4 Edge Extraction Result

Step 2: Line Detection

As for line detection, we adopt the method of Hough Line Transform. [10] The result of Line detection is shown in following figure (the detected lines are plot in black color).

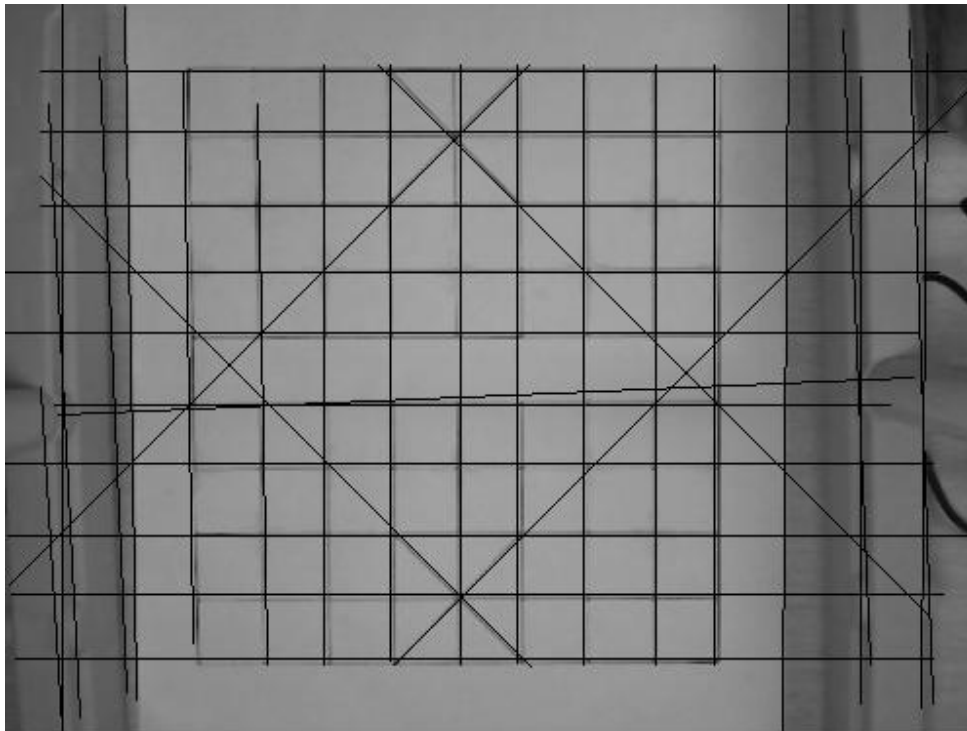


Fig. 6.5 Result of Using Hough Transform to Detect Lines

Step 3: Corner Detection

To detect corners on the chess board image, we adopt Harris corner detector. Harris corner detection method compares a small number of surrounding pixels with its immediate neighbor to check if current pixel is a corner. The result of corner detection is shown in following figure (the detected corners are plot in black color).

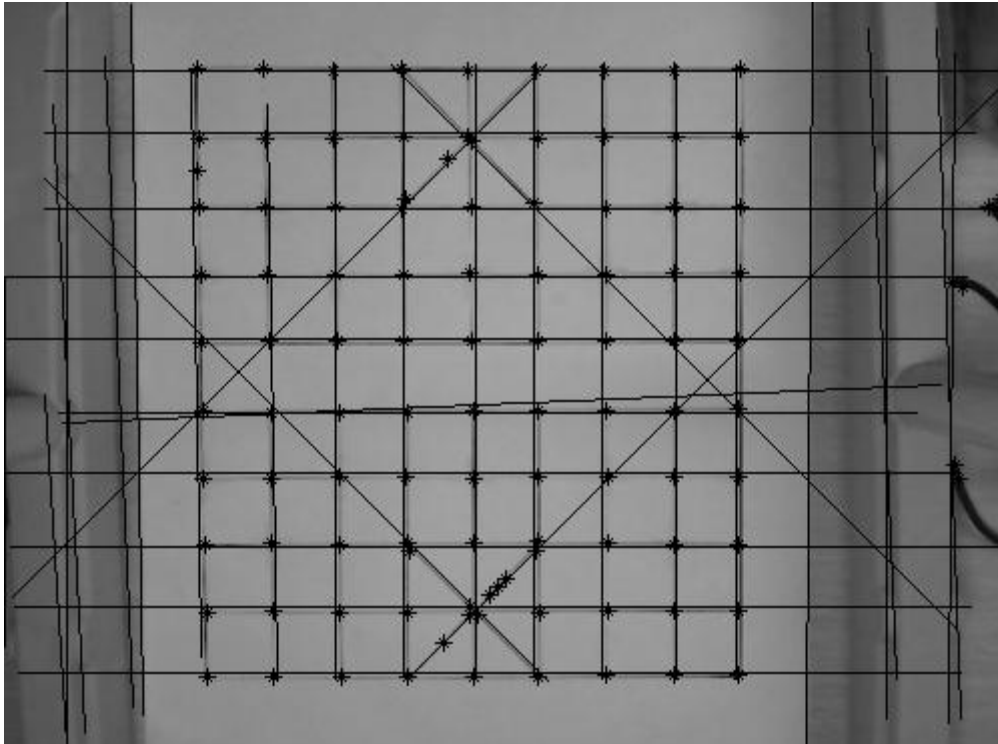


Fig. 6.6 Result of Using Harris Corner Detector to Detect Corners

Step 4: Select “Valid” Board Corners

As we can see from the result of Step 3, when we apply Harris corner detector, a bunch of corners will be detected including “invalid” corners which is actually not board corners. In this step, we will select “valid” board corners from the bunch of corners.

According to **Rule 3**, valid board corners in Chinese chess board must be intersected with at least two valid board lines. Based on that rule, we can eliminate a lot of useless corners. This Step is a prelude that serves for next Step.

The result of selecting “valid” board corners is shown on following figure, where the blue “*” marks represent the selecting result. As you can see, there are also some “fake” board corners selected. Therefore, it is not enough so far to calibrate the chess board.

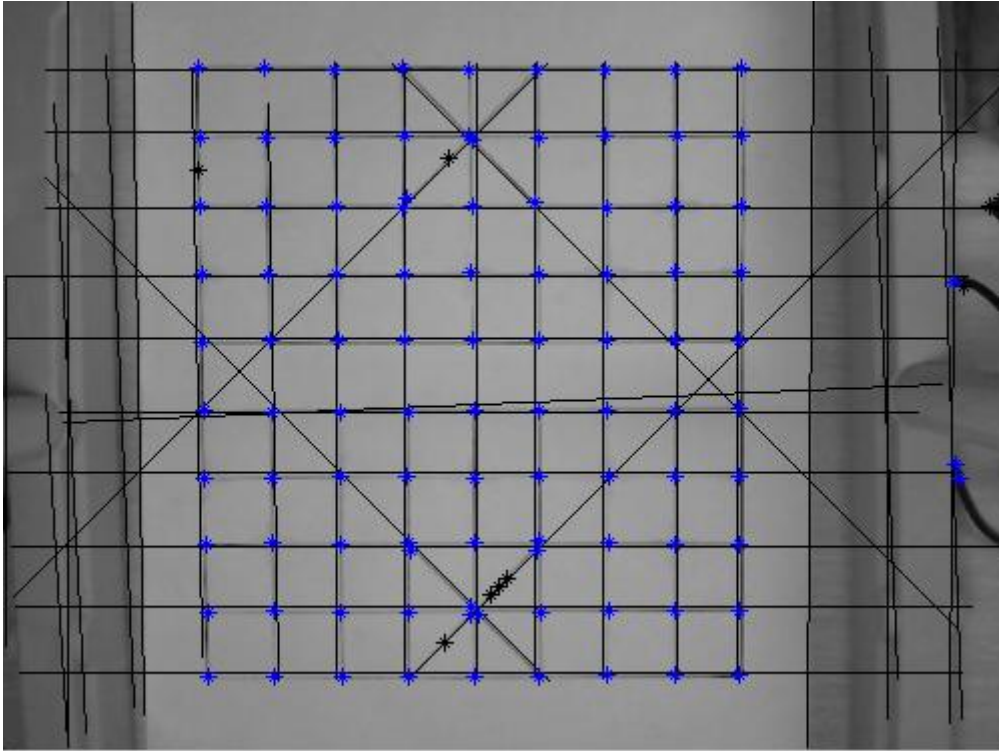


Fig. 6.7 Result of Selecting “Valid” Board Corners

Step 5: Select “Valid” Board Lines

“Valid” board lines selection can be done based on the result of step 4. According to **Rule 4**, valid board lines in Chinese chess board must pass through a bunch of valid board corners. In our case, we set the threshold of the number of valid board corners that a valid board line must come across to 4.

However, that is not enough. As we can see in Figure 6.1, on standard Chinese chess board, there exist four diagonal lines, which are also meet Rule 4. But, obviously these lines are not we want. In order to eliminate the diagonal lines, we introduce **Rule 5** (the intersection angle between horizontal lines and vertical lines must be restricted within a threshold) and **Rule 6** (The intersection angle between lines in same direction (horizontal or vertical) must be less than a threshold).

The procedure of selecting valid board lines is list as follows.

- (1) Use histogram to statistic the distribution of angles of lines detected on the chess board image.
- (2) Get two peaks (ave1, ave2) from the histogram in Step (1), and make sure their degree of intersection angle is bigger than a threshold. They represent angles of most lines, i.e., the average degree of horizontal lines and average degree of vertical lines.
- (3) Traverse all of the lines. If they satisfy: (a) the difference between its degree of angles and ave1 or ave2 is less than 10 degree; (b) the number of valid board

corners on the lines are bigger than 4, then those lines are regarded as valid board lines.

The result of selecting “valid” board lines is shown on following figure, where the green lines represent the selecting result.

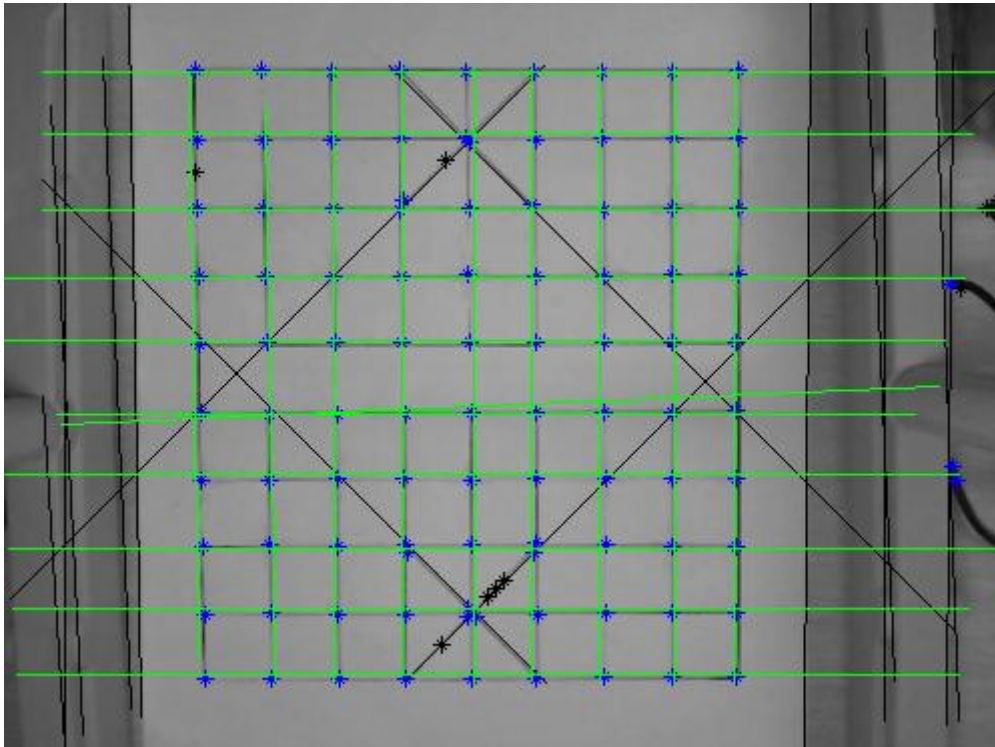


Fig. 6.8 Result of Selecting “Valid” Board Lines

Step 6: Select Four Chess Board Boundary Lines

Simply knowing valid board lines is not enough. That is because we cannot guarantee each of the board lines is detected or no redundant lines are detected. In order to enhance robustness, we must do further processing. The purpose of this Step is to select four chess board boundary lines.

To get the four boundary lines, firstly, we classify the valid board lines into two groups: the group of vertical lines and the group of horizontal lines. In the first group, we select the left most line and the right most line. In the second group, we select the upper most line and the line in the bottom. Those lines are chess board boundary lines.

The result of boundary lines is shown on following figure, they are drew in red, blue, yellow, green colour respectively in thick line.

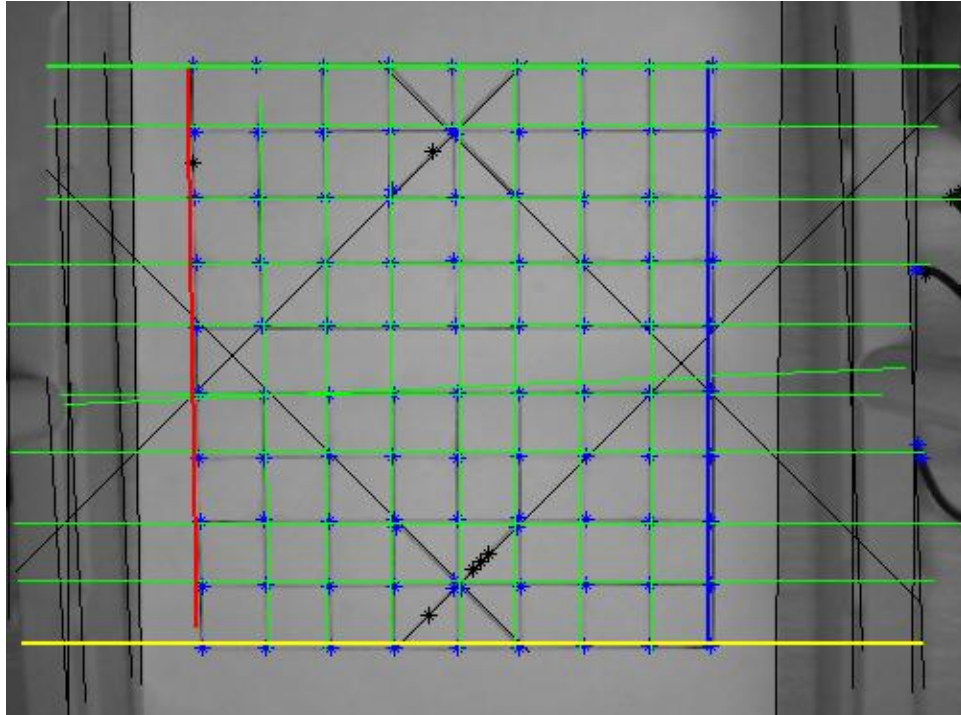


Fig. 6.9 Result of Selecting Boundary Lines

Step 7: Compute Four Boundary Corners

Boundary corners are the points of intersection between boundary lines pair wise, which can be get easily according to Cramer's rule [28]. Boundary corners are shown on following figure, they are drew in red, blue, yellow, green colour respectively in rectangle mark.

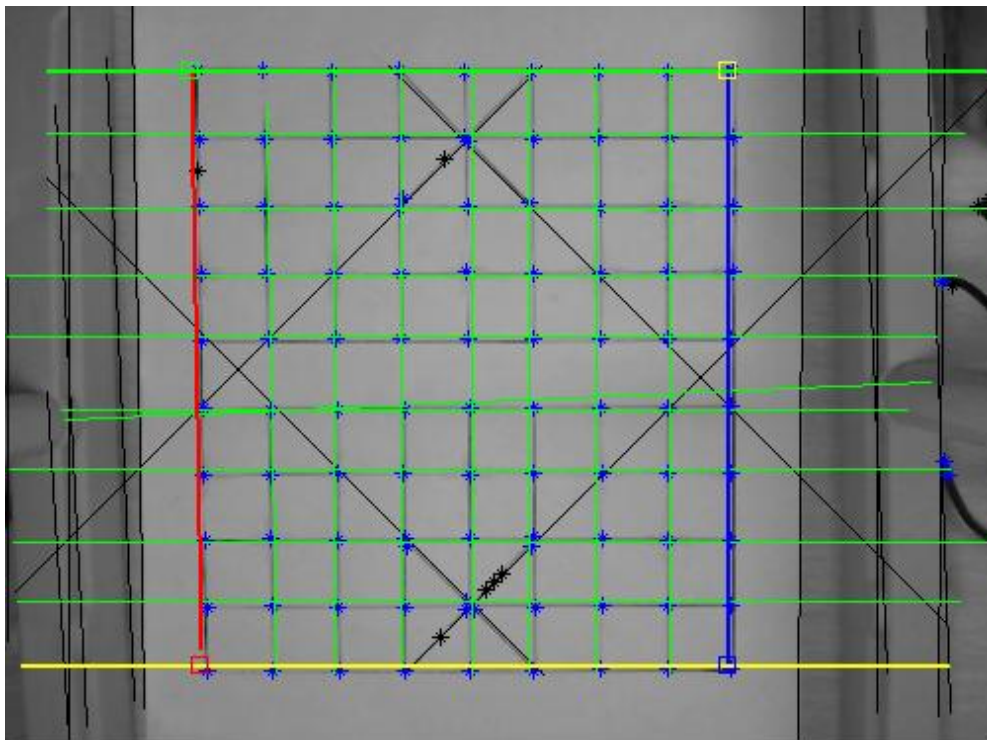


Fig. 6.10 Result of Selecting Boundary Corners

Step 8: Determine Board Direction

Unlike western chess, in Chinese chess, the number of horizontal board lines and vertical board lines are different. So far, we have got four boundary corners. In order to do further calibrate we must know the direction of current board, i.e., which direction of the board is occupied by 10 board lines and which direction of the board is occupied by 9 board lines.

The main idea is to build inter board lines by interpolation in both direction, then compute the distance error between interpolated board lines and detected board lines for each direction. The direction with smaller error is the correct answer. The algorithm of determining the board direction is list as follows.

- (1) Put the boundary corners anticlockwise, and define as "corner1", "corner2", "corner3", and "corner4".
- (2) Assume the board line "corner1->corner4" or "corner2->corner3" is "horizontal" line, i.e., it is intersected with 10 board lines. While the board line "corner1->corner2" or "corner4-> corner3" is "vertical" line.
- (3) Set *error1* to 0; set *counter* to 0.
- (4) Build 8 "vertical" inner board lines, whose "theta" is close to ave1 (see Step 5).
- (5) For *current_line* = each "vertical" inner board lines
 Traverse each valid board lines (see Step 5);
 If there is a valid board lines that has a similar theta as *current_line* and is close to *current_line*;
 Then, calculate the distance between the two lines and save as *dis*; let *error1* = *error1* + *dis*; let *counter* = *counter* + 1;
End
- (6) Build 7 "horizontal" inner board lines, whose "theta" is close to ave2 (see Step 5).
- (7) For *current_line* = each "horizontal" inner board lines
 Traverse each valid board lines (see Step 5);
 If there is a valid board lines that has a similar theta as *current_line* and is close to *current_line*;
 Then, calculate the distance between the two lines and save as *dis*; let *error1* = *error1* + *dis*; let *counter* = *counter* + 1;
End
- (8) Let *error1* = *error1* / *counter*;
- (9) Assume the board line "corner1->corner2" or "corner4->corner3" is "horizontal" line, i.e., it is intersected with 10 board lines. While the board line "corner1->corner4" or "corner2-> corner3" is "vertical" line.
- (10) Set *error2* to 0; set *counter* to 0.
- (11) Build 8 "vertical" inner board lines, whose "theta" is close to ave1 (see Step 5).
- (12) For *current_line* = each "vertical" inner board lines
 Traverse each valid board lines (see Step 5);

If there is a valid board lines that has a similar theta as *current_line* and is close to *current_line*;
Then, calculate the distance between the two lines and save as *dis*; let *error2* = *error2* + *dis*; let *counter* = *counter* + 1;
End

(13) Build 7 “horizontal” inner board lines, whose “theta” is close to *ave2* (see Step 5).

(14) For *current_line* = each “horizontal” inner board lines
Traverse each valid board lines (see Step 5);
If there is a valid board lines that has a similar theta as *current_line* and is close to *current_line*;
Then, calculate the distance between the two lines and save as *dis*; let *error2* = *error2* + *dis*; let *counter* = *counter* + 1;
End

(15) Let *error2* = *error2* / *counter*;

(16) Compare *error1* and *error2*.
If *error1* is bigger than *error2*
Then
Let *bc1* = *corner1*;
Let *bc2* = *corner2*;
Let *bc3* = *corner3*;
Let *bc4* = *corner4*;
Else
Let *bc1* = *corner2*;
Let *bc2* = *corner3*;
Let *bc3* = *corner4*;
Let *bc4* = *corner1*;
End

(17) Finally, we get the four variables *bc1*, *bc2*, *bc3*, and *bc4*. They are sort in anticlockwise order, and the board line “*bc1*->*bc4*” or “*bc2*->*bc3*” is “horizontal” line, i.e., it is intersected with 10 board lines. While the board line “*bc1*->*bc2*” or “*bc4*->*bc3*” is “vertical” line.

The corresponding positions of *bc1*, *bc2*, *bc3*, and *bc4* in Chinese chess board are illustrated in following figure.

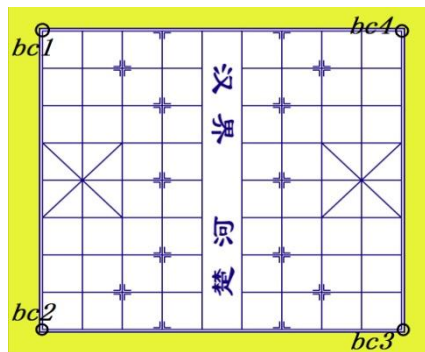


Fig. 6.11 Result of Determining the Board Direction

Step 9: Calculate Inner Board Corners

So far, we have got the four chess board boundary corners and the board direction. The final step to calibrate Chinese chess board is to get the positions of all other inner board corners.

Inner board corners can be computed by interpolating four boundary corners. Here we apply bilinear interpolation method. [29] Following code implement this step.

```
X = zeros(9, 10);
Y = zeros(9, 10);
for m = 1:9
    for n = 1:10
        x1 = ((n-1)*bc4(1) + (10-n)*bc1(1)) / 9;
        y1 = ((n-1)*bc4(2) + (10-n)*bc1(2)) / 9;
        x2 = ((n-1)*bc3(1) + (10-n)*bc2(1)) / 9;
        y2 = ((n-1)*bc3(2) + (10-n)*bc2(2)) / 9;

        X(m, n) = ((m-1)*x2 + (9-m)*x1) / 8;
        Y(m, n) = ((m-1)*y2 + (9-m)*y1) / 8;
    end
end
```

The result of board calibration is shown on following figure, the board corners are drew as cyan circles and magenta circles on each side.

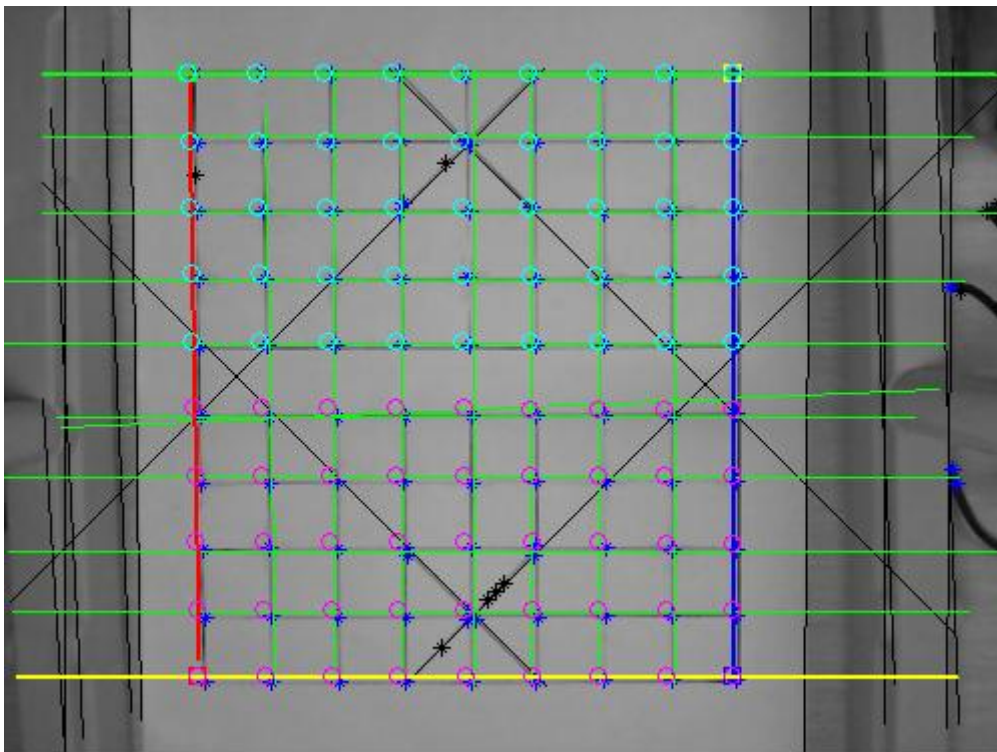


Fig. 6.12 Result of Board Calibration

6.2 Result Analysis

Experiments have been taken to use the proposed algorithm to detect Chinese chess board under different viewing angles, different backgrounds, and different light conditions. Following figures show the result in typical scenarios.

6.2.1 Different Viewing Angles

First, let's look at board calibration results in different viewing angles.

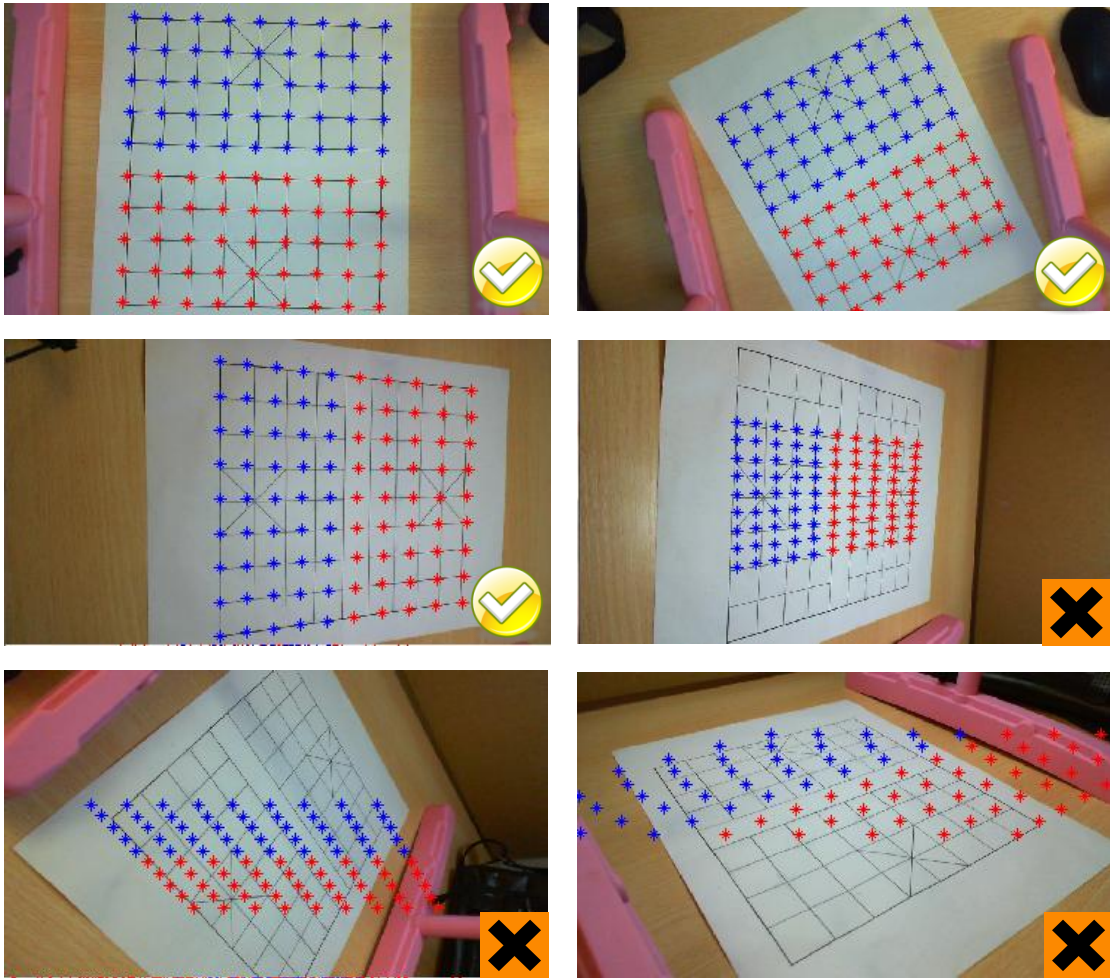
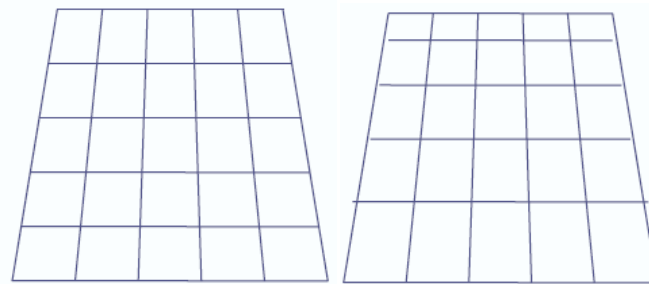


Fig. 6.13 Board Calibrate in Different Viewing Angles

As we can see from the first two images, chess board is successfully calibrated in top view with different rotation angle with different rotation angles, which indicates the proposed algorithm work well in top view and different rotation angles of chess board. The remaining figures indicate that with the viewing angle become lower and lower the result will be worse. It is worth noticing that the third image shows us the calibration result in perspective view. Although, the chess board is successfully calibrated, however, if looking carefully, we can found the inner board corners are not overlapping exactly with real board corners. That is caused by a defect of the proposed algorithm. As in the proposed algorithm, inner board corners are

calculated by bilinear interpolation of the four boundary corners. While in perspective view, the distribution of board lines is not linear any more. Following figures provide a good illustration of this.



a. Linear

b. Perspective

Fig. 6.14 Deformation Caused by Perspective View

6.2.2 Different Background

Now, let's look at the results affected by different background.

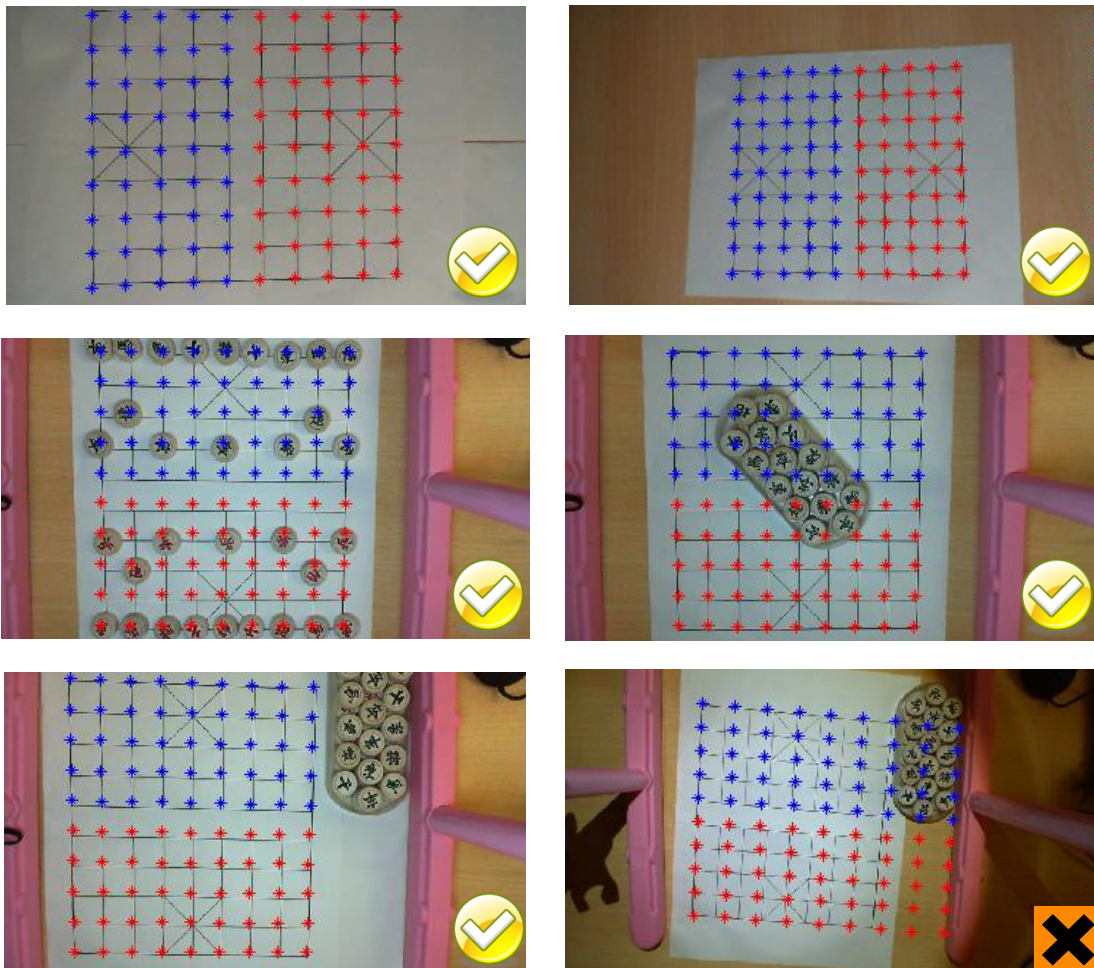


Fig. 6.15 Board Calibration Affected by Different Background

Image 1 and image 2 shows chess board in different background, both of them are successfully calibrated. Image 3 and image 4 shows chess board that is covered by objects. Both of them are successfully calibrated. Image 5 and image 6 shows the results of board calibration affected by surrounding objects. One of them fails in detection.

We can easily conclude that the algorithm is highly resistant to backgrounds, surrounding objects, or even object covered partially on the board.

6.2.3 Different Light Conditions

Here we show some experimental result under different light conditions.

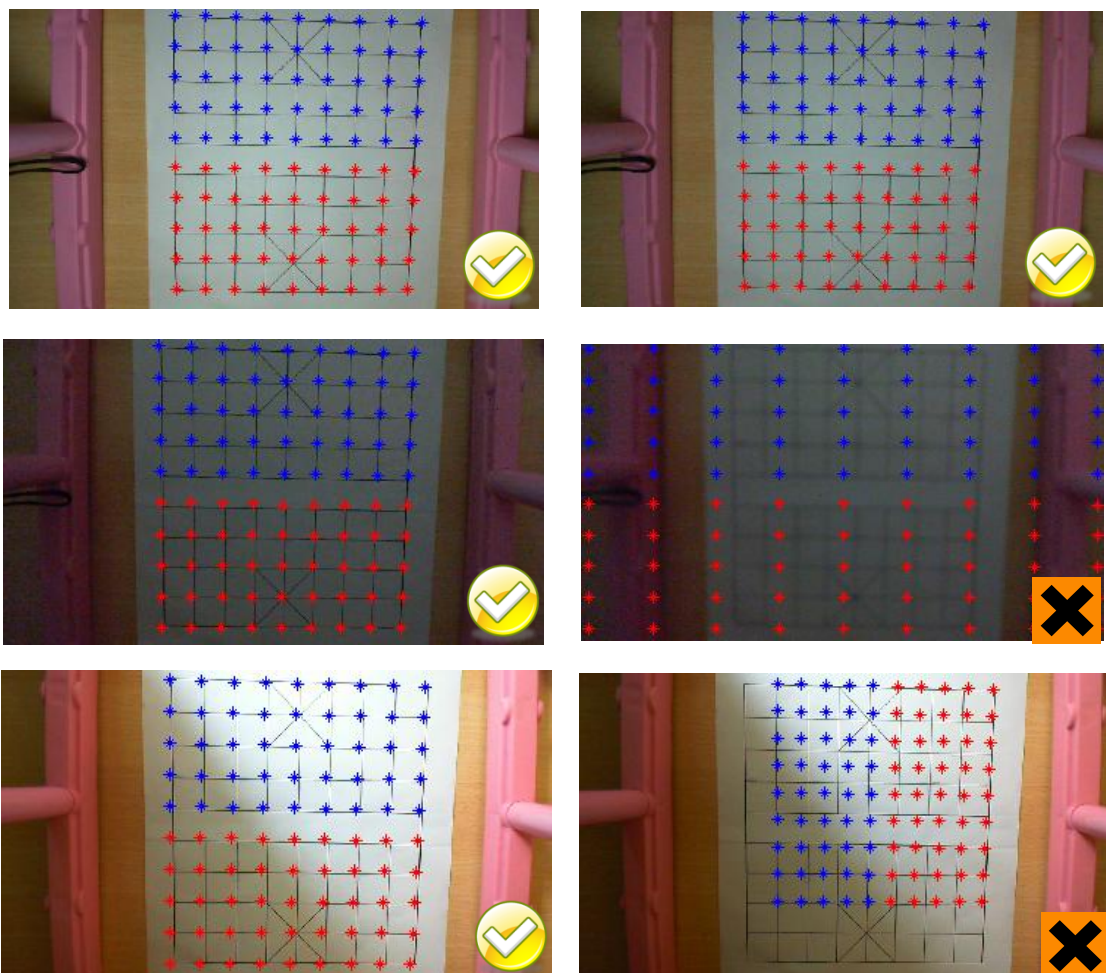


Fig. 6.16 Board Calibration Affected by Different Light Conditions

As can be seen, the first four images show the experiments under different intensity of light sources. The last fourth image in dark light condition failed the test. Last two images show the experiments under effect of non-uniform illumination. One of them failed the test.

The results of experiment indicate that the algorithm will be affected if the intensity of light source is quite low or the illumination is extremely non-uniform. However, the overall performance of the algorithm shows good robustness under the influence of different light conditions.

6.2.4 Conclusion

In conclusion, the proposed Chinese chess board calibration algorithm performance well in top view, is not affected by board rotation, shows great resistance to background and interference, and has a good robustness under the influence of different light conditions. However, the performance will decrease with the viewing angle become lower. Also, in some extremely condition of illumination, chess board will be mistakenly detected.

7 Proposed Piece Recognition Algorithm

The key problems of Chinese piece recognition are the extraction of pieces and the recognition of piece characters.

As for piece extraction, the author in [23] adopts a binarization method based on the difference threshold of neighbor pixels' gray-level. The papers in [30] [31] block the image base on the prior knowledge of Chinese chess board, use adaptive threshold method to binarise sub-images and use Hough transform to detect circles, and then extract the pieces. The method of using prior knowledge of chess board to extract chess pieces can only handle cases where the chess board must display with no rotation, no tilt and the position of pieces must be fixed precisely. The method of using threshold to binarise image requires that the color of chess pieces must differ greatly from the color of chess board. Also, it has a high requirement against illumination. Those methods can only deal with chess board in restricted condition and are lack of universality.

As for piece character recognition, methods including template matching, ring statistics and method based on connectivity and holes are introduced in Chapter 5. All of these methods are lack of robustness. In cases when reflection happens on the character or the camera is out of focus, then character strokes on binarization image might have holes, deform or adhere to each other. If these happen, all of the feature that those methods adopt will be invalid. Some other solutions are based of rotation invariant feature vectors. Some mature solutions to that problem include Hu [25] and Zernicke [26] moment feature which are based on methods of algebraic invariants. Some methods firstly obtain the contour points and then adopt Fourier transform. All of them have been proven to be less resistant to interference and less robust. Therefore, in this Chapter, we propose a new Chinese piece recognition algorithm based on projection histogram of polar coordinates spaces and Fourier descriptor.

Here is the basic idea. Firstly, we adopt Canny operator to extract the edge of the image, and morphological methods are applied to do some pre-process. Then, we use Hough circle transform to detect circles on the board so that chess pieces can be segmented to do further recognition. As for the feature extraction of pieces, we first convert the character image to polar coordinates space, and then build the projection histogram. Finally, Fourier transform is applied to solve the shift invariance.

Algorithm procedures are expounded in details in following sections.

7.1 Algorithm Procedures

Step 1: Piece Extraction

Before we could do recognition on chess pieces, we must segment the chess piece from the chess board. The segmentation steps are list as follows.

(1) Convert the original image to gray-scaled image

(2) Remove noise

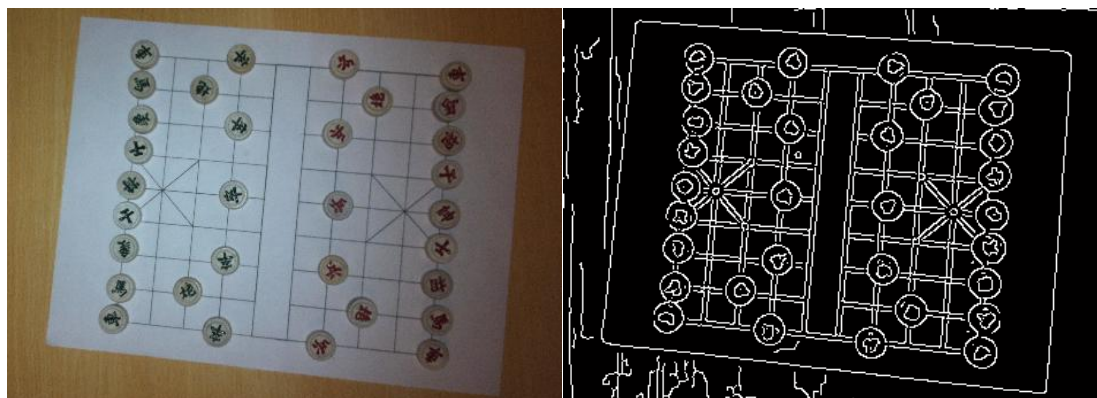
As we adopt normal web camera, the image quality is not good enough and noise is obviously existed. In order to increase the algorithm performance, median filter [27] are applied to remove noise.

(3) Resize image

For robustness purpose, this algorithm is design to fit different image format and size. However, if the size of image is big (e.g., the size of image captured from iPhone camera: 1936x2592), the speed of algorithm cannot be tolerated. That is mainly because the computing time of Hough circle transform will increase dramatically along with the increase of image size. Therefore, we resize the image to 500 by 500 pixels and in the final stage, when we finish extracting chess pieces, the segmentation result will be restored to its original size.

(4) Edge extraction

As Hough transform works on binary image, so before doing circle detection we should extract edges from image and change it into binary image. There exist many mature edge extraction algorithms including Canny operator, Sobel operator, Laplace operator, Robert operator. After comparison of experiment results, Canny operator performs the best in our case. Edge extraction result is shown in following figure.



a. Original Image

b. Result of Edge Extraction

Fig. 7.1 Edge Extraction Result

(5) Dilate

In order to increase the performance of circle detection, we need to use morphological algorithm [32] [33] to do pre-processing. Here we adopt dilate operation. The kernel of dilate operation we adopt is:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

The result of dilate operation is shown as follows.

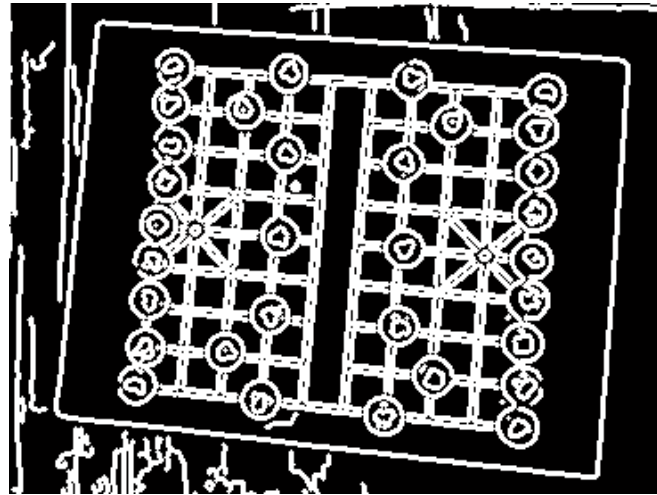


Fig. 7.2 Result of Dilate Operation

(6) Circle Detection

As for circle detection, we adopt the method of Hough Circle Transform. [10] The result of Hough Circle Transform is shown in following figure (the detected circles are drawn in red color).

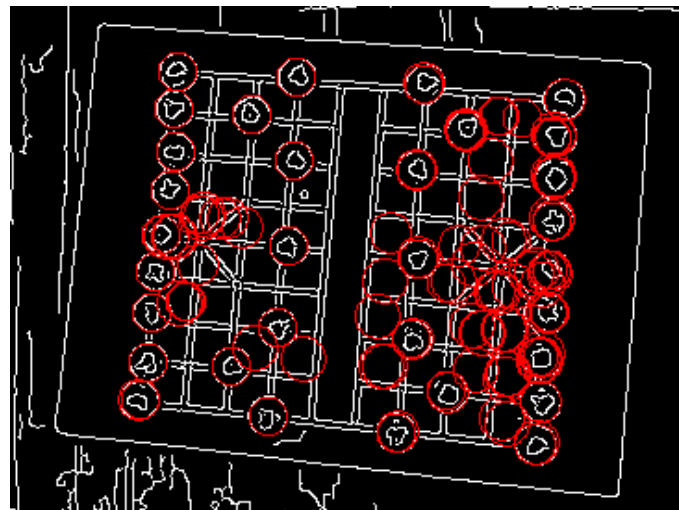


Fig. 7.3 Result of Hough Circle Transform

It is noticeable that, Hough Circle Transform method will detect a bunch of circles and some of them are “invalid”. In order to eliminate the invalid circles on the board, we choose the circles that are closest to their corresponding position of board corners (see Chapter 6). However, there are also some mistakenly detected circles have no corresponding pieces. We reject those circles by comparing the similarity of corresponding sub-image of foreground and background. If their similarity is less than a threshold, we can judge they are invalid circles and should be rejected. Following figure shows the result of circle detection. (The detected circles are drawn in red color)

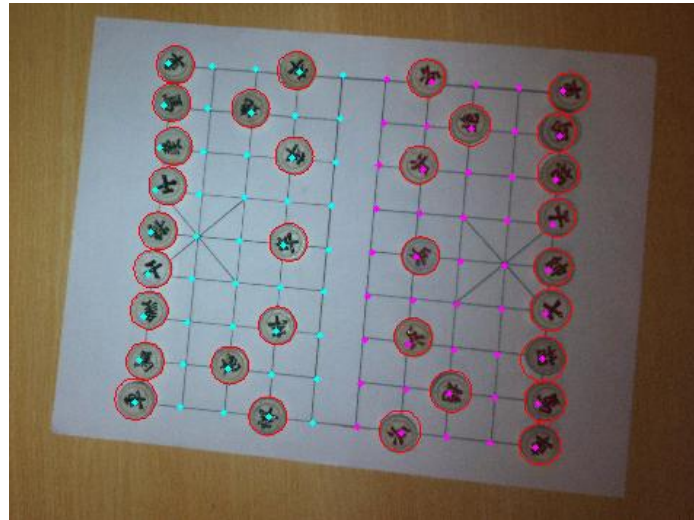


Fig. 7.4 Result of Circle Detection

(7) Piece Segmentation

After detection of circles, we can easily segment the corresponding chess pieces according to the position of the circles. Following figures show the segmentation result.

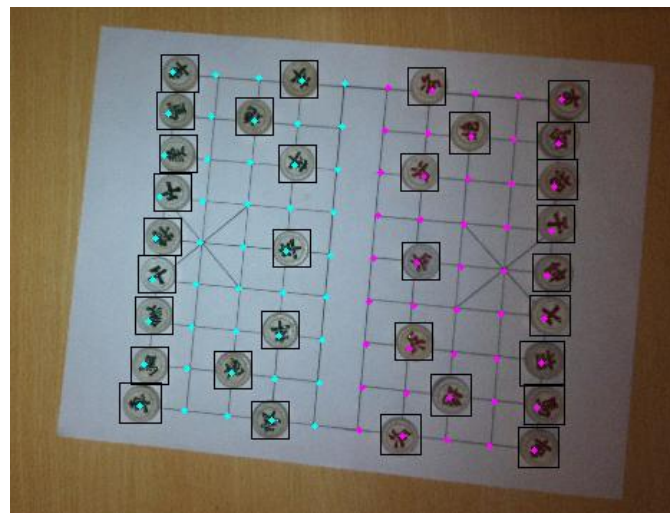


Fig. 7.5 Result of Piece Segmentation

Step 2: Binarization

Piece images that are segmented from chess board should be processed and binarized, so as to extract the feature of each piece character. We first convert the original image to gray-scale image. Then image is binarized by using Otsu's method [34] to get global image threshold. (Figure 7.6 (b) shows the result) To remove noise, median filter is also applied. Given that the pixels will be sparse when we convert the image from Cartesian coordinate space to polar coordinates space (we will talk in next section), it is necessary to apply morphological dilate operation to make the binarized character looks thicker. (Figure 7.6 (c) shows the result)



Fig. 7.6 Piece Image Binarization

Step 3: Convert to Polar Coordinates Space

The rotation transformation of image in Cartesian coordinate space corresponds to the translation transformation in polar coordinates space. The relation between the two coordinate spaces is indicated by formulas 7.1.

$$\begin{cases} f_r(x, y) = f(x\cos\theta_0 + y\sin\theta_0, -x\sin\theta_0 + y\cos\theta_0) \\ f_r(\rho, \theta) = f(\rho, \theta - \theta_0) \end{cases} \quad (7.1)$$

Before doing the conversion of coordinates, the center of the character should be found. We get the center point by doing statistics on position of each black pixel on the binarized image. Then we use the center as origin of Cartesian coordinate and convert the image to polar coordinate space. Following figures illustrate some examples (pawn) of the conversion.

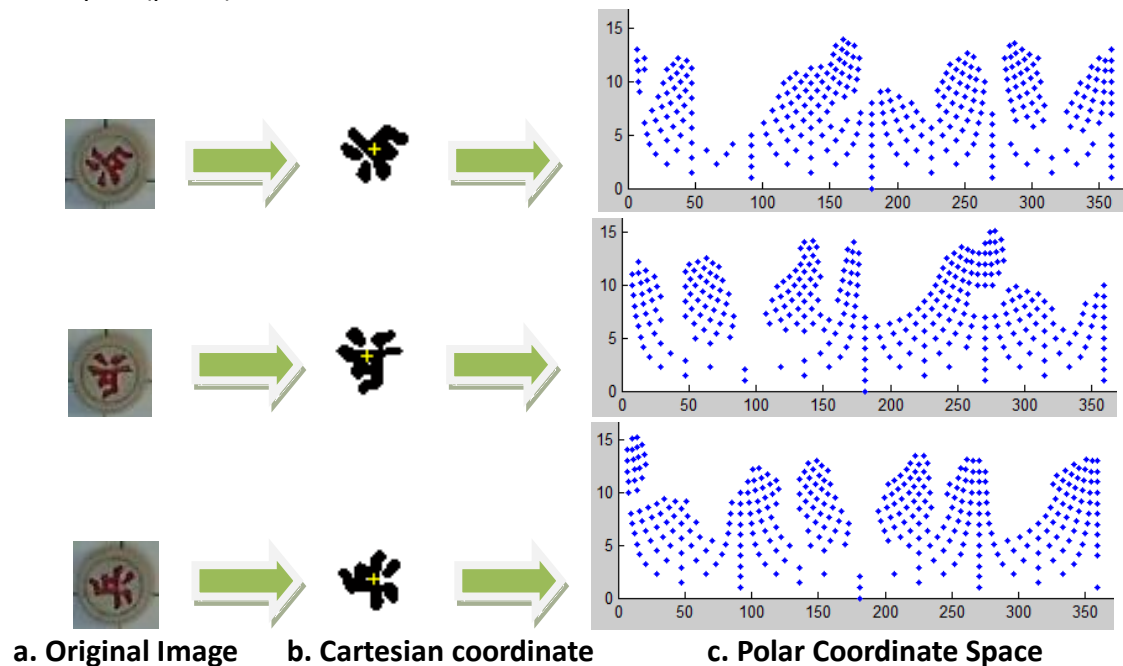


Fig. 7.7 Convert Image from Cartesian Coordinates to Polar Coordinates

The projection histogram of the two axes in polar coordinate space equals to the projection histogram of the radial direction and tangential direction in Cartesian coordinate space.

Step 4: Projection Histogram

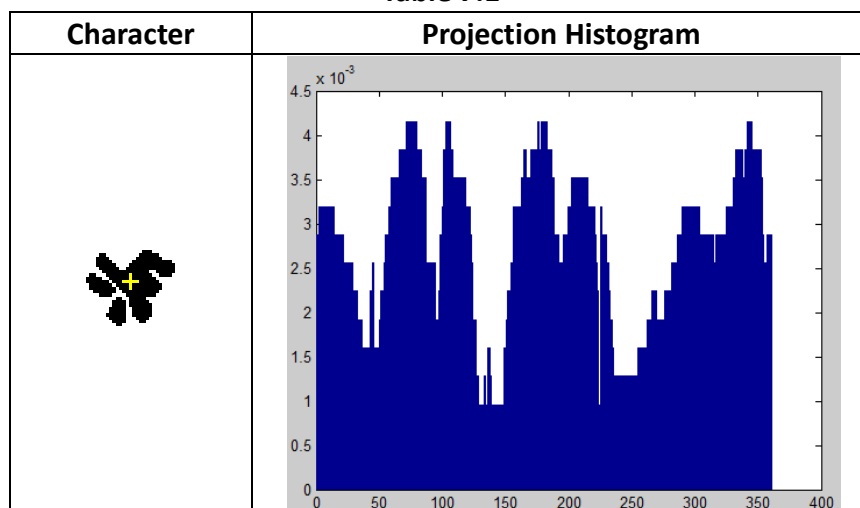
Projection histogram of the two axes in Cartesian coordinate space has been regarded as a mature technique, which has been applied on detecting and recognizing texts and is used to segment paragraphs, text lines and characters. Although this statistic feature has a high fault tolerance, it is very sensitive to the tilt of character. Also this statistic method lost lots of the spatial structure information and has poor representational ability of character features. However, in polar coordinate space, the spatial structure of a character has been unfolded in its tangential direction. By this way, the projection histogram of the image in polar coordinate space can keep the most information of character structures.

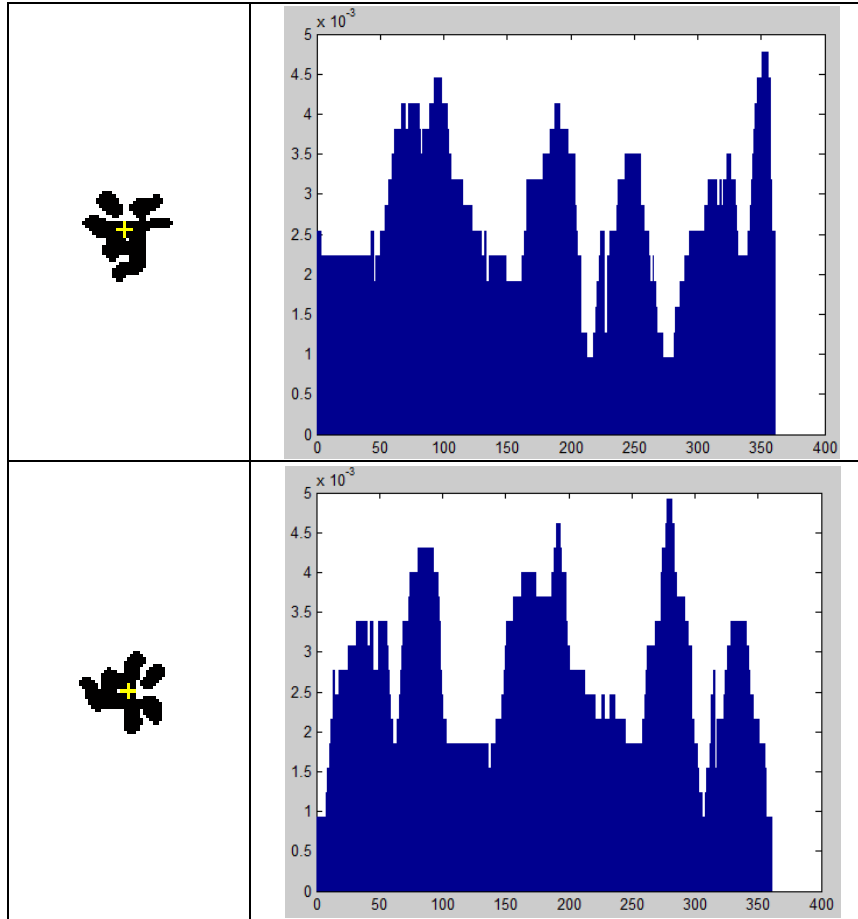
The formulas of projection histogram are:

$$\begin{cases} n_{\theta}(\theta) = \sum_{\rho} f_r(\rho, \theta) \\ n_{\rho}(\rho) = \sum_{\theta} f_r(\rho, \theta) \end{cases} \quad (7.2)$$

As $n_{\rho}(\rho)$ have a poor representational ability, our proposed solution adopts $n_{\theta}(\theta)$ to do character recognition. Following figures show the result of “pawn” in Chinese chess with different rotation angles.

Table 7.1





Step 5: Fourier Descriptor

In last Step, we have successfully transferred the problem of rotate invariance of the characters to the problem of shift invariance. In this Step, we are going to extract Fourier descriptor [35] [36] to solve the shift invariance.

Fourier Transform is one of the methods to analyze the signal in its frequency domain. Fast Fourier Transform (FFT) [37] is an efficient method to deal with discrete Fourier transform (DFT) [38].

The expression of discrete Fourier transform is:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi j}{N}nk} \quad (7.3)$$

There are many forms of fast Fourier transform. The most frequently used algorithm is Cooley-Tukey FFT [39], it recursively divides DFT into odd part and even part, until each part of DFT has only two items. The expressional forms are:

$$\begin{aligned}
X_k &= \sum_{m=0}^{\frac{N}{2}-1} x_{2m} e^{-\frac{2\pi j}{N}(2m)k} + \sum_{m=0}^{\frac{N}{2}-1} x_{2m+1} e^{-\frac{2\pi j}{N}(2m+1)k} \\
&= \sum_{m=0}^{M-1} x_{2m} e^{-\frac{2\pi j}{M}mk} + e^{-\frac{2\pi j}{N}k} \sum_{m=0}^{M-1} x_{2m+1} e^{-\frac{2\pi j}{M}mk} \\
&= \begin{cases} E_k + e^{-\frac{2\pi j}{N}k} O_k & \text{if } (k < M) \\ E_{k-M} - e^{-\frac{2\pi j}{N}(k-M)} O_{k-M} & \text{if } (k \geq M) \end{cases} \quad (7.4)
\end{aligned}$$

Signal shift in time domain corresponds to the rotation in frequency domain, where only phase-frequency can be changed the amplitude-frequency will not be affected. Then expressional formulas are:

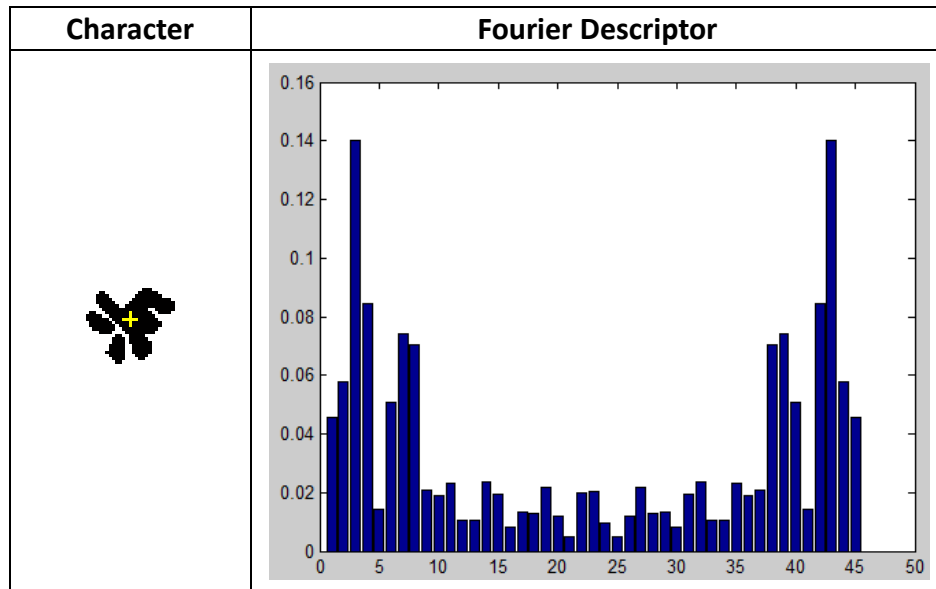
$$\begin{cases} f(x) \rightarrow F(\omega) \\ f(x + t_0) \rightarrow F(\omega) e^{-j\omega t_0} \end{cases} \quad (7.5)$$

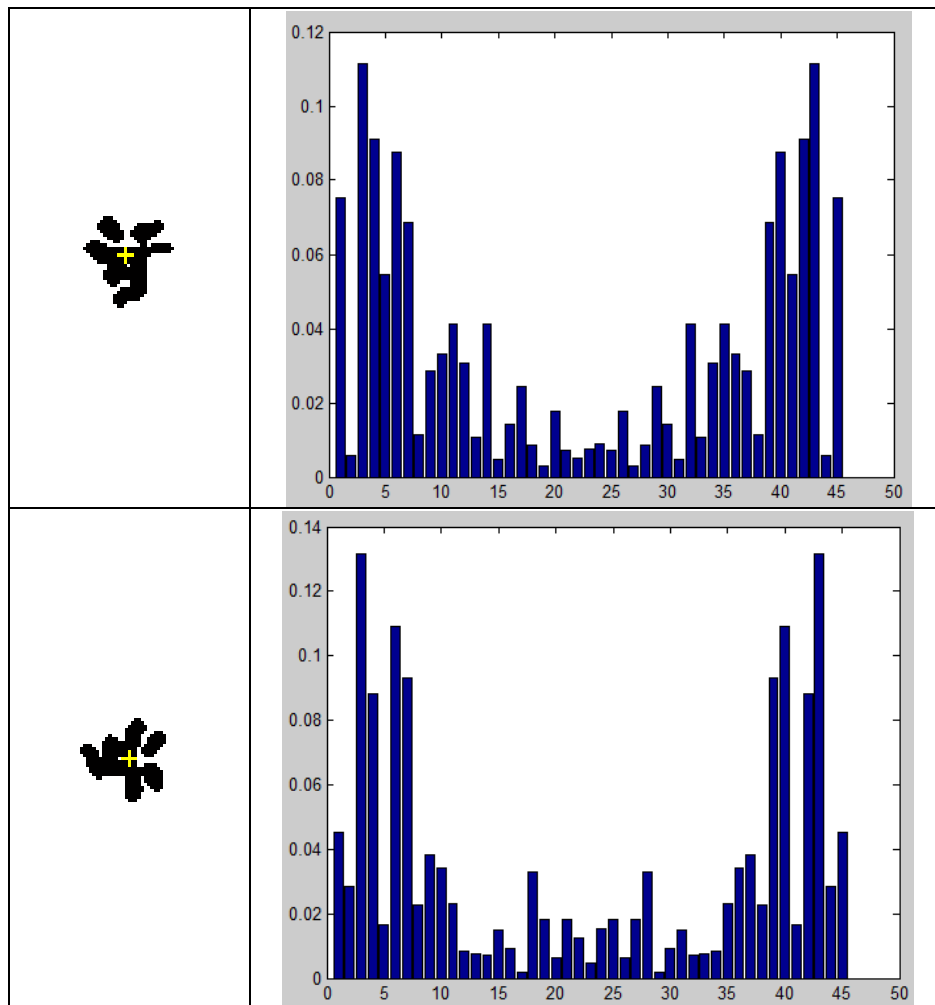
Therefore, as for the vertical projection of polar coordinate $n_\theta(\theta)$, to make it shift invariance, we should apply Fourier transform to it, and finally get the rotate invariance feature vector (V) of the piece character.

$$V = |F[n_\theta(\theta)]| \quad (7.6)$$

As most of the information converge at the front part and end part of the result of Fourier transform, so we only clip the front part and the end part to act as our descriptor. Following table shows the Fourier descriptor of “pawn” in Chinese chess with different rotation angles.

Table 7.2





The result indicates that the Fourier descriptors of the character “Pawn” in different angle of rotation are almost the same, which embody the rotation invariance of Fourier descriptor.

In following table, we list the Fourier descriptor of each piece characters.

Table 7.3

Original Image	Piece Character	Fourier Descriptor

Step 6: Classification

In standard Chinese chess, there are 14 distinct piece types. As relatively low number of training images of chess pieces can be obtained. We choose to adopt Support Vector Machine (SVM) classifier [40] [41], which work well even with low number of training samples.

In our system, we use the Gaussian radial base function as a kernel for transforming the samples.

$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right), \quad \gamma > 0 \quad (7.6)$$

The variable γ determines the width of the radial function. If the value of γ is large, then the radial function will be narrow which is too specific and usually result in more false negative classifications than false positives. Otherwise, if the value of γ is small, the radial function will be wide, which is not specific enough and may produce more false positives.

In addition, the weight coefficient \mathbf{w} and offset \mathbf{b} should satisfy following constraints:

$$d_i(w^T x_i + b) \geq 1 - \varepsilon \quad i = 1, 2, \dots \quad (7.7)$$

The variable ε is the penalty for misclassification. The optimal weight coefficients minimize the criterion function:

$$J(w) = \frac{1}{2} w^T w + C \sum_i \varepsilon_i \quad (7.8)$$

The **C** parameter represent the penalty for misclassification of each sample, which is crucial for avoiding the over fitting of the classifier on the training samples.

7.2 Result Analysis

To test the recognition accuracy and performance of proposed algorithm, we take 16 sets of samples. The results are analyzed by using the confusion matrix [42] (Table 7.4).

Table 7.4 – Confusion Matrix

	R1	R2	R3	R4	R5	R6	R7	B1	B2	B3	B4	B5	B6	B7	Total
R1	11	2	1	1	0	1	0	0	0	0	0	0	0	0	16
R2	3	12	0	0	0	0	1	0	0	0	0	0	0	0	16
R3	4	0	11	0	0	1	0	0	0	0	0	0	0	0	16
R4	0	0	0	15	1	0	0	0	0	0	0	0	0	0	16
R5	0	0	2	1	12	0	1	0	0	0	0	0	0	0	16
R6	2	0	0	1	0	13	0	0	0	0	0	0	0	0	16
R7	1	3	0	0	0	0	12	0	0	0	0	0	0	0	16
B1	0	0	0	0	0	0	0	12	2	0	0	0	1	1	16
B2	0	0	0	0	0	0	0	2	11	0	0	0	0	3	16
B3	0	0	0	0	0	0	0	1	0	13	1	0	0	1	16
B4	0	0	0	0	0	0	0	1	0	0	14	0	1	0	16
B5	0	0	0	0	0	0	0	1	0	0	2	13	0	0	16
B6	0	0	0	0	0	0	0	0	0	0	1	0	14	1	16
B7	0	0	0	0	0	0	0	1	3	0	0	0	1	11	16
Total	21	17	14	18	13	15	14	18	16	13	18	13	17	17	224

(Note: R1 = red king, R2 = red guard, R3 = red bishop, R4 = red horse, R5 = red rook, R6 = red cannon, R7 = red pawn, B1 = black king, B2 = black guard, B3 = black bishop, B4 = black horse, B5 = black rook, B6 = black cannon, B7 = black pawn)

According to the above confusion matrix, Table 7.5 and Table 7.6 give a list of assessment of the performance.

Table 7.5 – Performance Assessment 1

	Commission	Omission	Prod. Acc	User. Acc
R1 (red king)	5/16=0.313	10/21=0.48	11/21=0.52	11/16=0.688
R2 (red guard)	4/16=0.25	5/17=0.29	12/17=0.71	12/16=0.75
R3 (red bishop)	5/16=0.313	3/14=0.21	11/14=0.79	14/16=0.688
R4 (red horse)	1/16=0.063	3/18=0.167	15/18=0.833	15/16=0.938
R5 (red rook)	4/16=0.25	1/13=0.077	12/13=0.923	12/16=0.75
R6 (red cannon)	3/16=0.188	2/15=0.133	13/15=0.867	13/16=0.813
R7 (red pawn)	4/16=0.25	2/14=0.14	12/14=0.86	12/16=0.75
B1 (black king)	4/16=0.25	6/18=0.333	12/18=0.667	12/16=0.75

B2 (black guard)	5/16=0.313	5/16=0.313	11/16=0.688	11/16=0.688
B3 (black bishop)	3/16=0.188	0/13=0	13/13=1	13/16=0.813
B4 (black horse)	2/16=0.125	4/18=0.222	14/18=0.778	14/16=0.875
B5 (black rook)	3/16=0.188	0/13=0	13/13=1	13/16=0.813
B6 (black cannon)	2/16=0.125	3/17=0.176	14/17=0.824	14/16=0.875
B7 (black pawn)	5/16=0.313	6/17=0.353	11/17=0.647	11/16=0.688
Average	0.224	0.207	0.793	0.777

Table 7.6 – Performance Assessment 2

Overall Accuracy	174/224 = 77.7%
Kappa Coefficient	0.76

8 The Software

This Chapter provides a brief introduction to the system software.

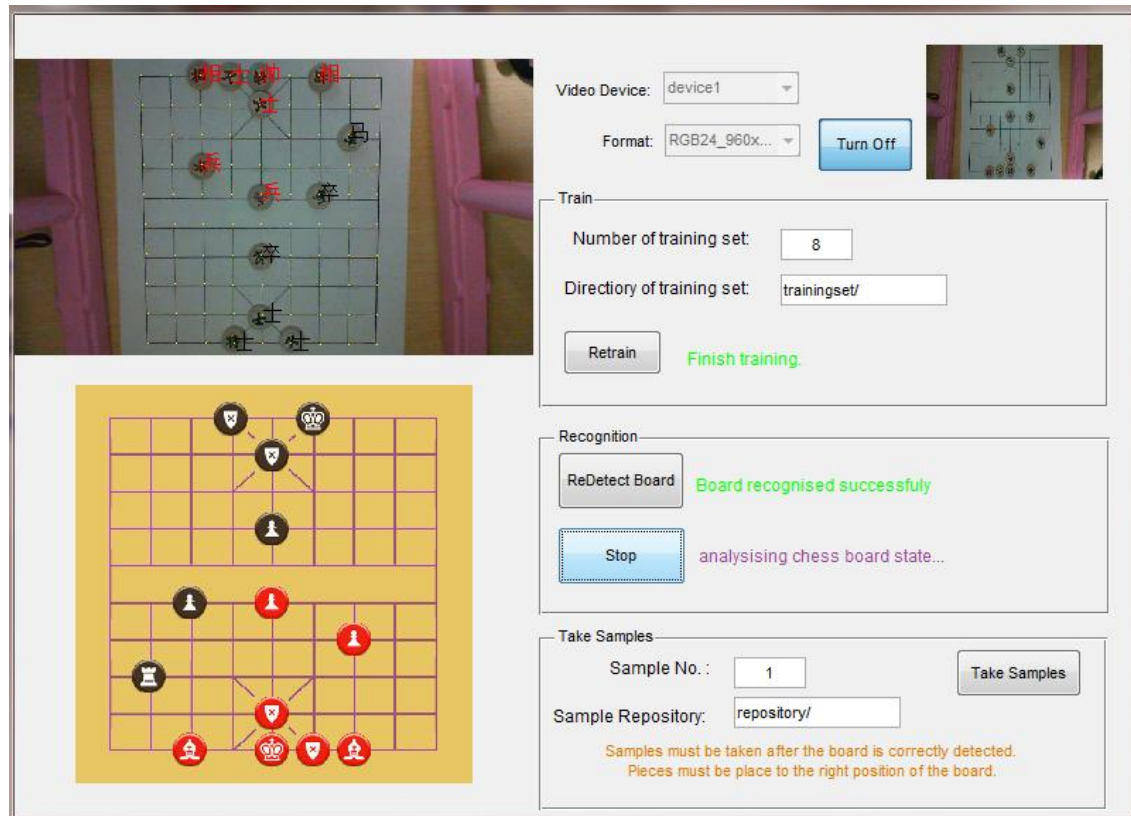


Fig. 8.1 System Software GUI

Figure 8.1 shows the GUI of the system software. Before we start running the program, we can select the camera device and choose the format of video output, and then press the “Turn On” button.

As can be seen from the user interface, the software can be divided into three modules according to its functionality: “Train”, “Recognition” and “Take Samples”.

Module 1 provides the function of training. User can input the directory where the training sets locate and the number of training sets then presses the button “Train” to start training. After finish training, a prompt will be shown in green color saying “Finish training”. Also, you can press the button “Retrain” to re-train the recognition system.

Module 2 provides the function of detecting the Chinese chess board and recognizing chess pieces in real-time, which can be stop or resumed in any time.

Module 3 provides the function of taking piece samples that can be used to train the recognition system. To take samples, user should input the target directory and put the chess pieces on the right position of the board according to their initial position.

Then, press the button “Take Sample”, the system will automatically take image sample of each piece according to where the piece locates.

9 Evaluation

The objectives of this project include:

- (1) correctly calibrate the chess board
- (2) automatically recognize the identity of each chess piece on chess board
- (3) display the state of chess board in real-time

As for the first objective, the proposed corner-line matching algorithm can effectively achieve the goal. The experiment result shows that the proposed Chinese chess board calibration algorithm performance well in top view, is not affected by board rotation, shows great resistance to background and interference, and has a good robustness under the influence of different light conditions. However, the performance will decrease gradually with the viewing angle become lower. Also, in some extremely condition of illumination, chess board will be mistakenly detected. Comparing with other solutions mentioned in Chapter 4, the proposed algorithm shows the best robustness, flexibility, anti-interference capability, and success rate in top view. However, in perspective view with low angle, it might not perform as good as some other solutions.

In addition to the success of the first objective of this project, the second objective: automatically recognize the identity of each chess piece on the chess board has also been completed. Comparing with the solutions mentioned in Chapter 5, the proposed algorithm is more robust and elegant, which needs fewer training samples but achieves better effect. Also, the proposed algorithm has no requirement and less constraints on specific chess set. The algorithm solves the problem of the rotate invariance of piece characters effectively; therefore unlike the template matching method, the algorithm has no limitation on how to put the chess pieces. In addition, unlike some other methods, no prior knowledge of the chess rules is used by the proposed algorithm, so there are no assumptions on how and where particular chess pieces are allowed to move and the initial configuration of pieces on chess board is not necessary to be provided in advance.

With regard to the third objective, the software provides the function to track the state of board, reconstruct and display on user interface in real-time. Given that most western people can read Chinese characters, our system displays the result in symbolic images instead of Chinese characters.

10 Conclusion and Future Work

This project studies and implements the vision system of chess-playing robot. In the project, we have proposed corner-line matching algorithm to calibrate the chess board and proposed a new Chinese chess piece recognition algorithm based on projection histogram in polar coordinates space and Fourier descriptor to recognize chess pieces, which is proven to be effective. Future work can be focused on how to increase the accuracy of piece character recognition. Also, based on our research, Chinese chess-playing robot or some other intelligent system can be developed.

Bibliography

1. Jason E., "Probabilistic Location of a Populated Chessboard Using Computer Vision", 2004
2. S. Blunsden, "Chess Recognition", Ph.D. dissertation, University of Plymouth, BSc (Hons) Computing and Informatics, 2003.
3. Y. Jia, Y. Duan, D. Wang, L. Xue, Z. Liu and W. Wang, "Pieces Identification in the Chess System of Dual-Robot Coordination Based on Vision", Web Information Systems and Mining (WISM), 2010 International Conference, 2010.
4. S. Zhao, C. Chen, C. Liu, M. Liu, "Algorithm of Location of Chess-robot System Based on Computer Vision", Control and Decision Conference, 2008.
5. E. Jahandar, "Chess Playing Robot (robochess)", [Online]. <http://www.jahandar.ir/Chess-Playing-Robot/>
6. F.Groen, G.der Boer, A. Van Inge, and R.Stam, "A Chess-playing Robot: Lab Course in Robot Sensor Integration", Instrumentation and Measurement Technology Conference, IEEE, 1992
7. E.Sokic and M. Ahic-Djokic, "Simple Computer Vision System for Chess Playing Robot Manipulator as a Project-based Learning Example", IEEE Symposium on Signal Processing and Information Technology, 2008.
8. T. Cour, R. Lauranson, and M. Vachette, "Autonomous Chess-playing Robot", undergraduate thesis, Ecole Polytechnique, 2002.
9. K. Tam, J. Lay, and D. Levy, "Automatic Grid Segmentation of Populated Chessboard Taken at a Lower Angle View", Computing Techniques and Applications, 2008
10. R. Duda and P.Hart, "Use of the Hough transform to detect lines and curves in pictures", Comm. ACM, vol. 15, no. 1, 1972
11. Matuszek, C., Mayton, B., et al. "Gambit: An Autonomous Chess-Playing Robotic System." In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2011.
12. M.Piskorec, "Computer Vision System for the Chess Game Reconstruction"
13. David Urting, Yolande Berbers, "MarineBlue: A Low-Cost Chess Robot", Proceedings of the IASTED International Conference on Robotics and Applications, 2003
14. Mohsen Sharifi, Mahmoud Fathy, Maryam Tayefeh Mahmoudi, "A Classified and Comparative Study of Edge Detection Algorithms," itcc, pp.0117, International Conference on Information Technology: Coding and Computing, 2002
15. S.Blunsden, "Chess Recognition", Undergraduate Thesis, University of Plymouth, 2002.
16. Jose Goncalves, "Chess Robot System: a Multi-disciplinary Experience in Automation"
17. Fenglei Xu, "The Research of the Vision Algorithm and Intelligent Control Software for Chess", Postgraduate Dissertation, Harbin Institute of Technology, 2006

18. S. Theodoridis, K. Koutroumbas, "Pattern Recognition (fourth edition)", Academic Press, 2009
19. Y. Amit, "2D Object Detection and Recognition", Cambridge, MA: MIT Press, 2002
20. S. Theodoridis, K. Koutroumbas, "An Introduction to Pattern Recognition: A MATLAB Approach", Academic Press, 2010
21. Z.K. Xiao, "Study on Binocular Vision System in Robotic chess player", Postgraduate Dissertation, Beifang University of Technology, 2008
22. Bresenham, J. E., "Algorithm for computer control of a digital plotter". IBM Systems Journal, 1965
23. DU Junli, ZHANG Jingfei, HUANG Xinhan, "Chess-board Recognition Based on Vision", Computer Engineering and Applications, 2007
24. John R. Shaw, "QuickFill: An efficient flood fill algorithm", [Online]. <http://www.codeproject.com/Articles/6017/QuickFill-An-efficient-flood-fill-algorithm>, 2004
25. M. K. Hu, "Visual Pattern Recognition by Moment Invariants", IRE Trans. Info. Theory, 1962
26. Khotanzad, A. and Y. H. Hong, "Invariant Image Recognition by Zernike Moments", IEEE Transactions on Pattern Analysis and Machine Intelligence, 1990
27. M. Sonka, V. Hlavac and R. Boyle, "Image Processing, Analysis, and Machine Vision (Second Edition)", Thomson Learning and PT Press, 1999
28. Cramer, Gabriel, "Introduction à l'Analyse des lignes Courbes algébriques" (in French), Geneva: Europeana, 1750
29. Meijering, Erik, "A chronology of interpolation: from ancient astronomy to modern signal and image processing", Proceedings of the IEEE, 2002
30. Zhiwei Zhang, Kong Fanrang, Jiwen Zhao, Qingbo He, and Zengrong Wu, "Image Processing and Recognition in the Vision of Chinese Chess Playing Robot", Computer Applications and Software, 2008
31. Jinwen Zhao, Zhiwei Zhang, Fang Xie, Yongbin Liu, Pu, Cheng, "Image Processing in Vision of Chinese Chess Playing Robot Based on SVM", Journal of System Simulation, 2007
32. Jean Serra, "Image Analysis and Mathematical Morphology", Academic Press, 1982
33. Edward R. Dougherty, "An Introduction to Morphological Image Processing", SPIE Optical Engineering Press, 1992
34. Nobuyuki Otsu, "A threshold selection method from gray-level histograms". IEEE Trans. Sys., Man., Cyber., 1979
35. Charles Zahn and Ralph Roskies, "Fourier Descriptors for Plane Closed Curves", IEEE Trans on Computers, 1971
36. Frank P. Kuhl and Charles R. Giardina, "Elliptic Fourier Features of a Closed Contour", Computer Graphics and Image Processing, 1981
37. Brigham, E. O. , "The Fast Fourier Transform", New York: Prentice-Hall, 2002
38. C. Candan, M. A. Kutay and H. M. Ozaktas, "The discrete fractional Fourier transform", IEEE Trans. on Signal Processing, 2002
39. Cooley, James W., and John W. Tukey, "An Algorithm for the Machine Calculation

- of Complex Fourier Series”, Math. Comput., 1965
40. C. Lin, C. Hsu and C. Chang, “A practical guide to support vector classification”, Technical report, Department of Computer Science, National Taiwan University, 2003.
 41. J.Shawe-Taylor, N. Cristianini, “Support Vector Machines and other kernel-based learning methods”, Cambridge University Press, 2000.
 42. Stehman, Stephen V., "Selecting and interpreting measures of thematic classification accuracy", Remote Sensing of Environment, 1997

Appendix – Essential Source Code

Descriptor Extraction

```
function f = descriptor(img)
% DESCRIPTOR
% extract the feature vector from a
Chinese chess piece image.
% the program adopts the proposed
algorithm based on
% projection histogram in polar
coordinates space and Fourier
descriptor
% Input:
% img - the image of a Chinese chess
piece (can be a string or image matrix)
% Output:
% f - the feature extracted from
current image

DRAW = 0; % indicate whether to show
the result image. 0: off 1: on
PI_DIV_180 = 0.0174;
if ischar(img)
    I = imread(img);
else
    I = img;
end
s = 50;
II = imresize(I, [s, s]);
% change to gray-scale
G = rgb2gray(II);
mean = sum(sum(G)) / (s*s) - 1;
trh = mean / 2 + 10; % 80 55
trh = trh / 255; % threshold for
binarizing the image
%trh = graythresh(G); % OTSU method to
get threshold
BW = im2bw(G, trh);
%figure, imshow(BW), title('the binary
image');

% apply median filter
BW1 = medfilt2(BW);

if DRAW == 1
    %figure, imshow(BW1),
title('median filter apply on binary
image');
end

% erode (black to white) - make the
character thicker
sel = strel('arbitrary', [0 1 0; 1 1 1;
0 1 0]);
BW1 = imerode(BW1, sel);
if DRAW == 1
    figure, imshow(BW1), title('eroded
image'), hold on;
end

B = BW1;
% calculate the center of the character
cx = 0;
cy = 0;
index = 0;
for i = 1:s
    for j = 1:s
        if B(i, j) == 0 % black
            index = index + 1;
            CX(index) = j;
            CY(index) = i;
            cx = cx + j;
            cy = cy + i;
        end
    end
end
assert(index ~= 0);
cx = round(cx / index); % center
coordinate of black pixels
cy = round(cy / index);
CX = CX - cx; % coordinate of black
pixels
CY = CY - cy;
if DRAW == 1
```

```

        %plot(cx, cy, '+y',
'MarkerSize',3);
end
% 360 pixel sum method
% compute the distances between the
contour and center
f = zeros(1, 360);
max_d = 100;
CC = zeros(360, 2);
total = 0;
for i = 1:360%360
    % apply bresenham algorithm
    x1 = cx;
    y1 = cy;
    x2 = round(x1 + max_d *
cos(PI_DIV_180 * i));
    y2 = round(y1 + max_d *
sin(PI_DIV_180 * i));
    dx = abs(x2 - x1);
    dy = abs(y2 - x1);
    iTag = 0;
    if dx < dy % if dy is bigger, then
swap x, y coordiante
        iTag = 1;
        [x1, y1] = swap(x1, y1);
        [x2, y2] = swap(x2, y2);
        [dx, dy] = swap(dx, dy);
    end
    if x2 > x1 % assure whether minus
1 or plus 1 on each step
        tx = 1;
    else
        tx = -1;
    end
    if y2 > y1
        ty = 1;
    else
        ty = -1;
    end
    x = x1;
    y = y1;
    incl = 2*dy;
    inc2 = 2*(dy-dx);
    d = incl - dx;

    while x < s && x > 1 && y < s && y >
1
        % loop to "plot" the dots on
the line
        if d < 0
            d = d + incl;
        else
            y = y + ty;
            d = d + inc2;
        end
        if iTag == 1
            if B(x, y) == 0 % black
                f(i) = f(i) + 1;
                total = total +1;
                CC(i, 1) = x;
                CC(i, 2) = y;
            end
        else
            if B(y, x) == 0 % black
                f(i) = f(i) + 1;
                total = total + 1;
                CC(i, 1) = y;
                CC(i, 2) = x;
            end
        end
        x = x + tx;

    end
end
f = f / total;
if DRAW == 1
    figure, bar(f); %
end
f = fft(f); % apply Fast Fourier
transform to turn signal into
frequency-domain
f = abs(f); % amplitude (translation
invariance )
f = [f(2:25), f(340:360)]; % extract
the foremost few data (as most
information gather at foremost range)
if DRAW == 1
    figure, bar(f);
end
end
end

```

Board Calibration

```
function [X, Y] = boardDetect(img)
%BOARDDETECT calibrate Chinese chess board
% input:
%   img - the image of a Chinese chess board (can be a string or image matrix)
%   (upper left corner of image is the origin of coordinate system.)
% output:
%   X, Y - the coordinate of board corners

MAX_LINES = 35; % maximum number of lines can be detected by hough 30, 35
DRAW = 0; % indicate whether to show the result image. 0:off 1:on
RESTORE = 1; % map the result back the original size? 0:no; 1:yes
if ischar(img)
    I = imread(img);
else
    I = img;
end
% to gray-scale
G = rgb2gray(I);
% remove noise
G = medfilt2(G);
s = 500; % (scale factor) the width of scaled image
[s_h, s_w] = size(G);
current_h = round(s_h/s_w*s);
current_w = s;
K = imresize(G, [current_h, current_w]); % resize to make faster detection
% edge extract
BW = edge(K, 'canny');
[H,T,R] =
hough(BW, 'RhoResolution', 0.5, 'ThetaResolution', 0.5); %hough transfer
P =
houghpeaks(H, MAX_LINES, 'threshold', ceil(0.2*max(H(:)))));

% Find lines and plot them
lines =
houghlines(BW,T,R,P, 'FillGap',185, 'MinLength',2);
if DRAW == 1
    figure, imshow(K), hold on;
    title('The lines that have been found');
end
n_lines = length(lines);
for k = 1:n_lines
    xy = [lines(k).point1;
lines(k).point2];
    if DRAW == 1
        plot(xy(:,1),xy(:,2), 'LineWidth',1, 'Color','black');
    end
end
%detect corners
C = corner(K, 500);
n_c = size(C, 1);
if DRAW == 1
    plot(C(:,1), C(:,2), 'k*');
end
%correct lines and corners
%1. count number of lines passed on each corner.
l_on_c = zeros(n_c, 1); %store number of lines passed on each corner.
diff_thresh = 5; % corner-line detect threshold
for i = 1 : n_c
    for j = 1 : n_lines
        diff =
abs(C(i,1)*cos_d(lines(j).theta) +
C(i,2)*sin_d(lines(j).theta)...
- lines(j).rho);
        if diff < diff_thresh
            l_on_c(i) = l_on_c(i) + 1;
        end
    end
end
%2. select valid corner (the number of
```



```

lines passed on each corner is bigger
then threshold)
% improve? the line passed by the corner
must corssed with a big angle
thresh = 2;
temp = find(l_on_c >= thresh);
n_valid_corners = length(temp);
valid_corners = zeros(n_valid_corners,
2);
index = 1;
for i = 1:n_c
    if l_on_c(i) >= thresh
        valid_corners(index,:) =
C(i,:);
        if DRAW == 1

plot(valid_corners(index,1),
valid_corners(index,2), 'b*');
        end
        index = index + 1;
    end
end
assert(n_valid_corners == index-1);
% 3. count number of valid corners on
each line
c_on_l = zeros(n_lines, 1); %store the
number of valid corners on each line
for i = 1 : n_valid_corners
    for j = 1 : n_lines
        diff =
abs( valid_corners(i,1)*cos_d(lines
(j).theta)...
        +
valid_corners(i,2)*sin_d(lines(j).th
eta)...
        - lines(j).rho);
        if diff < diff_thresh
            c_on_l(j) = c_on_l(j) + 1;
        end
    end
end
end
% 4. select valid lines
% 4.1 choose the two average angles of
most lines in order to elimite

% cross lines
Y = zeros(181, 1); %(-90 -> 90)
for i = 1:n_lines
    theta =
int16(round(lines(i).theta));
    assert(theta>=-90 && theta<=90);
    Y(theta+91) = Y(theta+91) + 1;
    for k = -5:5
        Y(mod(theta+91+k-1,181)+1) =
Y(mod(theta+91+k-1,181)+1) + 1;
    end
end
ave1 = 0; % make sure ave1 is bigger than
ave2
ave2 = 0; %
for i = 1:181
    if ave1==0 || Y(i)>Y(ave1)
        ave1 = i;
    end
end
for i = 1:181
    if abs(ave1-i) < 10
        continue;
    end
    if ave2==0 || Y(i)>Y(ave2)
        ave2 = i;
    end
end
ave1 = ave1 - 91;
ave2 = ave2 - 91;
% 4.2 select valid lines
thresh = 4; % number of corners at
least on a valid line
index = 1;
angle_distance_thresh = 0.17;
for i = 1:n_lines
    if c_on_l(i) >= thresh &&
(angleDistance(lines(i).theta,
ave1)<=angle_distance_thresh ||...

angleDistance(lines(i).theta,
ave2)<=angle_distance_thresh)
        valid_lines(index) = lines(i);
        xy =

```

```

[valid_lines(index).point1;
valid_lines(index).point2];
    if DRAW == 1

plot(xy(:,1),xy(:,2),'LineWidth',1,'
Color','green');
    end
    index = index + 1;
end
end
n_valid_lines = index-1;
% 5. calculate the four outermost lines
up_left = 0;
bottom_right = 0;
up_right = 0;
bottom_left = 0;
angleType = zeros(n_valid_lines, 1); %
used to classify lines according to
whether
                                % their
angle belongs to ave1 or ave2
for i = 1 : n_valid_lines
    if
angleDistance(valid_lines(i).theta,
ave1) <
angleDistance(valid_lines(i).theta,
ave2)
        % angle 1
        angleType(i) = 1;
        if abs(ave1)<5
            if up_left == 0
                up_left = i;
            else
                bx_current =
valid_lines(up_left).rho/cos_d(valid
_lines(up_left).theta);
                bx =
valid_lines(i).rho/cos_d(valid_lines
(i).theta);
                if bx < bx_current
                    up_left = i;
                end
            end
        end
        if bottom_right == 0
            bottom_right = i;
        else
            by_current =
valid_lines(bottom_right).rho/cos_d(
valid_lines(bottom_right).theta);
            by =
valid_lines(i).rho/cos_d(valid_lines
(i).theta);
            if by < by_current
                up_left = i;
            end
        end
    end
    if bottom_right == 0
        bottom_right = i;
    else
        bx_current =
valid_lines(bottom_right).rho/cos_d(
valid_lines(bottom_right).theta);
        bx =
valid_lines(i).rho/cos_d(valid_lines
(i).theta);
        if bx > bx_current
            bottom_right = i;
        end
    end
end
else
    if up_left == 0
        up_left = i;
    else
        by_current =
valid_lines(up_left).rho/sin_d(valid
_lines(up_left).theta);
        by =
valid_lines(i).rho/sin_d(valid_lines
(i).theta);
        if by < by_current
            up_left = i;
        end
    end
    if bottom_right == 0
        bottom_right = i;
    else
        by_current =
valid_lines(bottom_right).rho/sin_d(
valid_lines(bottom_right).theta);
        by =
valid_lines(i).rho/sin_d(valid_lines
(i).theta);
        if by > by_current
            bottom_right = i;
        end
    end
end
end
else % angle 2
    angleType(i) = 2;
    if abs(ave2)<5

```

```

        if up_right == 0
            up_right = i;
        else
            bx_current =
valid_lines(up_right).rho/cos_d(valid_lines(up_right).theta);
            bx =
valid_lines(i).rho/cos_d(valid_lines(i).theta);
            if bx > bx_current
                up_right = i;
            end
        end
        if bottom_left == 0
            bottom_left = i;
        else
            bx_current =
valid_lines(bottom_left).rho/cos_d(valid_lines(bottom_left).theta);
            bx =
valid_lines(i).rho/cos_d(valid_lines(i).theta);
            if bx < bx_current
                bottom_left = i;
            end
        end
    else
        if up_right == 0
            up_right = i;
        else
            by_current =
valid_lines(up_right).rho/sin_d(valid_lines(up_right).theta);
            by =
valid_lines(i).rho/sin_d(valid_lines(i).theta);
            if by < by_current
                up_right = i;
            end
        end
        if bottom_left == 0
            bottom_left = i;
        else
            by_current =
valid_lines(bottom_left).rho/sin_d(valid_lines(bottom_left).theta);
            by =
valid_lines(i).rho/sin_d(valid_lines(i).theta);
            if by > by_current
                bottom_left = i;
            end
        end
    end
end
end
if up_left==0 || bottom_right==0 || up_right==0 || bottom_left==0
    disp('fail to detect board');
    X = 0;
    Y = 0;
    return;
end
if DRAW == 1
    plot([valid_lines(up_left).point1(1);
valid_lines(up_left).point2(1)],...
        [valid_lines(up_left).point1(2);
valid_lines(up_left).point2(2)],...
        'LineWidth',2,'Color','red');
    plot([valid_lines(bottom_right).point1(1);
valid_lines(bottom_right).point2(1)],...
        [valid_lines(bottom_right).point1(2);
valid_lines(bottom_right).point2(2)],...
        'LineWidth',2,'Color','blue');
    plot([valid_lines(bottom_left).point1(1);
valid_lines(bottom_left).point2(1)],...
        [valid_lines(bottom_left).point1(2);
valid_lines(bottom_left).point2(2)],...
        'LineWidth',2,'Color','red');
end
end

```

```

[valid_lines(bottom_left).point1(2);
valid_lines(bottom_left).point2(2)],...
... 'LineWidth',2,'Color','yellow');
plot([valid_lines(up_right).point1(1)
]);
valid_lines(up_right).point2(1)],...
[valid_lines(up_right).point1(2);
valid_lines(up_right).point2(2)],...
'LineWidth',2,'Color','green');
end
% 6. calculate the four board corners
according to Cramer's rule
boardCorner = zeros(4, 2);
% corner1
A =
cos_d(valid_lines(up_left).theta);
B =
sin_d(valid_lines(up_left).theta);
C = valid_lines(up_left).rho;
D =
cos_d(valid_lines(bottom_left).theta
);
E =
sin_d(valid_lines(bottom_left).theta
);
F = valid_lines(bottom_left).rho;
delta = A*E-B*D;
assert(delta ~= 0);
boardCorner(1, 1) = (C*E - B*F)/delta;
boardCorner(1, 2) = (A*F - C*D)/delta;
% corner2
A =
cos_d(valid_lines(bottom_left).theta
);
B =
sin_d(valid_lines(bottom_left).theta
);
C = valid_lines(bottom_left).rho;
D =
cos_d(valid_lines(bottom_right).thet
a);
E =
sin_d(valid_lines(bottom_right).thet
a);
F = valid_lines(bottom_right).rho;
delta = A*E-B*D;
assert(delta ~= 0);
boardCorner(2, 1) = (C*E - B*F)/delta;
boardCorner(2, 2) = (A*F - C*D)/delta;
% corner3
A =
cos_d(valid_lines(bottom_right).thet
a);
B =
sin_d(valid_lines(bottom_right).thet
a);
C = valid_lines(bottom_right).rho;
D =
cos_d(valid_lines(up_right).theta);
E =
sin_d(valid_lines(up_right).theta);
F = valid_lines(up_right).rho;
delta = A*E-B*D;
assert(delta ~= 0);
boardCorner(3, 1) = (C*E - B*F)/delta;
boardCorner(3, 2) = (A*F - C*D)/delta;
% corner4
A =
cos_d(valid_lines(up_right).theta);
B =
sin_d(valid_lines(up_right).theta);
C = valid_lines(up_right).rho;
D =
cos_d(valid_lines(up_left).theta);
E =
sin_d(valid_lines(up_left).theta);
F = valid_lines(up_left).rho;
delta = A*E-B*D;
assert(delta ~= 0);
boardCorner(4, 1) = (C*E - B*F)/delta;
boardCorner(4, 2) = (A*F - C*D)/delta;
if DRAW == 1
    plot(boardCorner(1,1),
boardCorner(1,2), 'rs');
    plot(boardCorner(2,1),
boardCorner(2,2), 'bs');
    plot(boardCorner(3,1),
boardCorner(3,2), 'ys');

```

```

        plot(boardCorner(4,1),
boardCorner(4,2), 'gs');
    end
    % 7. determine the direction of board
    % 7.1 assume (corner1, corner2) vs
    (corner4, corner3) - calculate the
    error
    e1 = 0.0; % error value
    en = 0;
    % build 8 "vertical" inner board line
    (whose theta belongs to ave2)
    innerLines1_v = zeros(8, 2, 2);
    point = zeros(2, 1);
    d_thresh =
    (pointDistance(boardCorner(1,:),
boardCorner(4,:)) + ...
pointDistance(boardCorner(2,:),
boardCorner(3,:))) / 2 / 9 / 2;
    for i = 1:8
        innerLines1_v(i, 1, 1) =
        ((9-i)*boardCorner(1, 1) +
i*boardCorner(4, 1))/9;
        innerLines1_v(i, 1, 2) =
        ((9-i)*boardCorner(1, 2) +
i*boardCorner(4, 2))/9;
        innerLines1_v(i, 2, 1) =
        ((9-i)*boardCorner(2, 1) +
i*boardCorner(3, 1))/9;
        innerLines1_v(i, 2, 2) =
        ((9-i)*boardCorner(2, 2) +
i*boardCorner(3, 2))/9;
        d = 10000.0; %record the smallest
distance
        for j = 1 : n_valid_lines
            if angleType(j) ~= 2
                continue;
            end
            point(1) = innerLines1_v(i, 1,
1);
            point(2) = innerLines1_v(i, 1,
2);
            temp_d =
pointLineDistance(point,
valid_lines(j));
            point(1) = innerLines1_v(i, 2,
1);
            point(2) = innerLines1_v(i, 2,
2);
            temp_d = (temp_d +
pointLineDistance(point,
valid_lines(j)))/2;
            if temp_d < d
                d = temp_d;
            end
        end
        if d < d_thresh
            e1 = e1 + d;
            en = en + 1;
        end
    end
    % build 7 "horizontal" inner board line
    (whose theta belongs to ave1)
    innerLines1_h = zeros(7, 2, 2);
    d_thresh =
    (pointDistance(boardCorner(1,:),
boardCorner(2,:)) + ...
pointDistance(boardCorner(4,:),
boardCorner(3,:))) / 2 / 8 / 2;
    for i = 1:7
        innerLines1_h(i, 1, 1) =
        ((8-i)*boardCorner(1, 1) +
i*boardCorner(2, 1))/8;
        innerLines1_h(i, 1, 2) =
        ((8-i)*boardCorner(1, 2) +
i*boardCorner(2, 2))/8;
        innerLines1_h(i, 2, 1) =
        ((8-i)*boardCorner(4, 1) +
i*boardCorner(3, 1))/8;
        innerLines1_h(i, 2, 2) =
        ((8-i)*boardCorner(4, 2) +
i*boardCorner(3, 2))/8;
        d = 10000.0; %record the smallest
distance
        for j = 1 : n_valid_lines
            if angleType(j) ~= 1
                continue;
            end
            point(1) = innerLines1_h(i, 1,

```

```

1);
    point(2) = innerLines1_h(i, 1,
2);
    temp_d =
pointLineDistance(point,
valid_lines(j));
    point(1) = innerLines1_h(i, 2,
1);
    point(2) = innerLines1_h(i, 2,
2);
    temp_d = (temp_d +
pointLineDistance(point,
valid_lines(j)))/2;
    if temp_d < d
        d = temp_d;
    end
end
if d < d_thresh
    e1 = e1 + d;
    en = en + 1;
end
end
e1 = e1 / en;
% 7.2 assume (corner1, corner4) vs
(corner2, corner3) - calculate the
error
e2 = 0.0;
en = 0;
% build 8 "vertical" inner board line
innerLines2_v = zeros(8, 2, 2);
d_thresh =
(pointDistance(boardCorner(1,:),
boardCorner(2,:)) + ...

pointDistance(boardCorner(4,:),
boardCorner(3,:))) / 2 / 9 / 2;
for i = 1:8
    innerLines2_v(i, 1, 1) =
((9-i)*boardCorner(1, 1) +
i*boardCorner(2, 1))/9;
    innerLines2_v(i, 1, 2) =
((9-i)*boardCorner(1, 2) +
i*boardCorner(2, 2))/9;
    innerLines2_v(i, 2, 1) =
((9-i)*boardCorner(4, 1) +
i*boardCorner(3, 1))/9;
    innerLines2_v(i, 2, 2) =
((9-i)*boardCorner(4, 2) +
i*boardCorner(3, 2))/9;
    d = 10000.0; %record the smallest
distance
    for j = 1 : n_valid_lines
        if angleType(j) ~= 1
            continue;
        end
        point(1) = innerLines2_v(i, 1,
1);
        point(2) = innerLines2_v(i, 1,
2);
        temp_d =
pointLineDistance(point,
valid_lines(j));
        point(1) = innerLines2_v(i, 2,
1);
        point(2) = innerLines2_v(i, 2,
2);
        temp_d = (temp_d +
pointLineDistance(point,
valid_lines(j)))/2;
        if temp_d < d
            d = temp_d;
        end
    end
    if d < d_thresh
        e2 = e2 + d;
        en = en + 1;
    end
end
% build 7 "horizontal" inner board line
innerLines2_h = zeros(7, 2, 2);
d_thresh =
(pointDistance(boardCorner(1,:),
boardCorner(4,:)) + ...

pointDistance(boardCorner(2,:),
boardCorner(3,:))) / 2 / 8 / 2;
for i = 1:7
    innerLines2_h(i, 1, 1) =

```

```

((8-i)*boardCorner(1, 1) +
i*boardCorner(4, 1))/8;
    innerLines2_h(i, 1, 2) =
((8-i)*boardCorner(1, 2) +
i*boardCorner(4, 2))/8;
    innerLines2_h(i, 2, 1) =
((8-i)*boardCorner(2, 1) +
i*boardCorner(3, 1))/8;
    innerLines2_h(i, 2, 2) =
((8-i)*boardCorner(2, 2) +
i*boardCorner(3, 2))/8;
    d = 10000.0; %record the smallest
distance
    for j = 1 : n_valid_lines
        if angleType(j) ~= 2
            continue;
        end
        point(1) = innerLines2_h(i, 1,
1);
        point(2) = innerLines2_h(i, 1,
2);
        temp_d =
pointLineDistance(point,
valid_lines(j));
        point(1) = innerLines2_h(i, 2,
1);
        point(2) = innerLines2_h(i, 2,
2);
        temp_d = (temp_d +
pointLineDistance(point,
valid_lines(j)))/2;
        if temp_d < d
            d = temp_d;
        end
    end
    if d < d_thresh
        e2 = e2 + d;
        en = en + 1;
    end
end
e2 = e2 / en;
% 8. output the X-Y coordinates of chess
board
if e1 < e2
    bc1 = boardCorner(1,:);
    bc2 = boardCorner(2,:);
    bc3 = boardCorner(3,:);
    bc4 = boardCorner(4,:);
else
    bc1 = boardCorner(2,:);
    bc2 = boardCorner(3,:);
    bc3 = boardCorner(4,:);
    bc4 = boardCorner(1,:);
end
X = zeros(9, 10);
Y = zeros(9, 10);
for m = 1:9
    for n = 1:10
        x1 = ((n-1)*bc4(1) +
(10-n)*bc1(1)) / 9;
        y1 = ((n-1)*bc4(2) +
(10-n)*bc1(2)) / 9;
        x2 = ((n-1)*bc3(1) +
(10-n)*bc2(1)) / 9;
        y2 = ((n-1)*bc3(2) +
(10-n)*bc2(2)) / 9;
        X(m, n) = ((m-1)*x2 + (9-m)*x1)
/ 8;
        Y(m, n) = ((m-1)*y2 + (9-m)*y1)
/ 8;
    end
end
if DRAW == 1
    plot(X(:,1:5), Y(:,1:5), 'mo');
    plot(X(:,6:10), Y(:,6:10), 'co');
    hold off;
end
if RESTORE == 1
    % map the result back into original
image size
    X = round(X * s_w / current_w);
    Y = round(Y * s_h / current_h);
end
end

```