MIPRO 2011, May 23-27, 2011, Opatija, Croatia

Computer vision system for the chess game reconstruction

M. Piškorec*, N. Antulov-Fantulin**, J. Ćurić*, O. Dragoljević*, V. Ivanac*, L. Karlović*
*Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia
** Division of Electronics, Laboratory for Information Systems, Ruđer Bošković Institute, Zagreb, Croatia
E-mail: matija.piskorec@gmail.com, nino.antulov@irb.hr, jura.curic@gmail.com, ognjen.dragoljevic@gmail.com, vedran.ivanac@gmail.com, luka.karlovic@gmail.com

Abstract - In this paper we describe the system for real-time machine vision recognition of chess table and figures. Input data are two synchronized video sequences from a top-view and side-view camera showing the game of chess between two players. The top-view is used mainly for determining the positions of the figures on the table while side-view enables correct recognition of figures in cases where they are not occluded by another figure in front of them. Even in cases where that happens the top-view helps to track the occluded figures along the table. No prior knowledge of the chess rules is used, so there are no assumptions on how and where particular figures are allowed to move. OpenCV algorithm implementations are used for most of the preprocessing and analysis, including the identification of the chess table and classification of figures with SVM classifier.

I. INTRODUCTION

Reconstruction of the game of chess from a video sequence is complex task that requires application of various algorithms and methods from the field of machine vision and machine learning. In this article we describe the ChessVision system that implements all phases needed for tracking the progress of the chess game: extracting the static pictures of individual moves from the video sequence, locating the board and individual fields, segmentation of figures present on the board and identifying their exact location and type.

In previous studies [1][2] initial configuration of the figures on the chess board was known and the camera was mounted on top-view of the stationary chess board. Many computer vision systems for Chinese chess [3][4] have also been successfully made. Development of research and algorithms in computer vision field made chess vision problem even more intriguing to the research community [5] for constructing autonomic chess playing robots based on computer vision [6][7].

In contrast to Chinese chess where figures are mainly two dimensional objects we have constructed chess vision system for normal chess figures used by World Chess Federation, although not with standard board. We have adapted the board to have distinct green and red squares, and have used two synchronized cameras placed in top-view and side-view of the stationary chess board. Our system does not imply any rules of chess figure moves. Initial configuration of figures on chess board is also not known. Furthermore, top-view camera cannot be used for figure recognition and side-view camera has problem of perspective distortion and overlapping figures. For that reason we used the top-view camera for figure detection and side-view camera for figure recognition.

In next section we describe implementation of the chess vision system. We extract static pictures that define each chess move. For each chess move we detect start and end positions on the board and recognize the figures involved in the move.

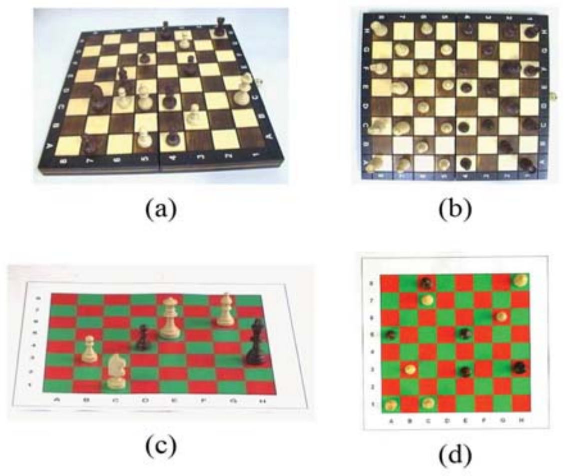


Figure 1. Different boards on which we tested the ChessVision system. The (a) and (b) are examples where the figures are of the same material as the board. That makes the segmentation of individual figures harder so we used board that is significantly different from the figures.

II. DESCRIPTION OF THE IMPLEMENTATION

Systems for chess recognition is developed in C++ programming language with extend usage of OpenCV libraries. Architecture of system consists of graphical user interface and components specialized for solving subproblems in chess recognition: Extraction of static picture, Detection of board and fields, Figure segmentation, Detection of figure positions and Figure recognition. Everything begins with creating video sequences of some chess game and preprocessing them with component "Extraction of static pictures". Results are represented in sequence of static images from top and side camera that represents static moments between movements in game. "Detection of board and fields" component use images with empty chess board from top and side camera to determine the positions of the fields on the images. The result is the coordinates of fields in top and side image. This is important because it provides information for figure segmentation and figure position. Purpose of "Figure segmentation" is to use pairs of side images from previous and current static movement and to prepare extracted segment of picture for recognition. "Figure segmentation" component uses the knowledge of "Detection of board and fields" and "Detection of figure positions". "Detection of figure positions" component uses the static images from top camera and knowledge of chess fields on image to determine if the field is empty or occupied with figure. In the end, "Figure recognition" component uses segmented image parts to recognize type and color of the figure.

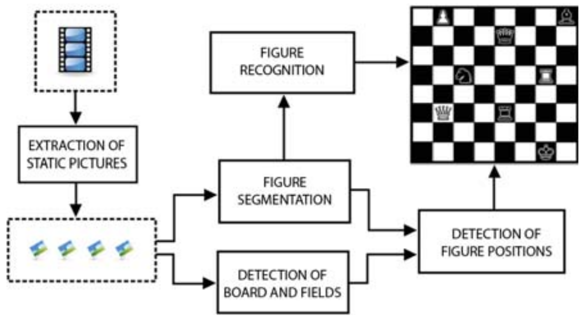


FIGURE RECOGNITION
EXTRACTION OF STATIC PICTURES
FIGURE SEGMENTATION
DETECTION OF FIGURE POSITIONS
DETECTION OF BOARD AND FIELDS

Figure 2. The overview of the implementation.

A. *Extractions of static pictures*

Figure detection and classification components cannot operate on images where player hand obscures figures or board fields. While player hand moves over the board, it is considered that valid figure move has not been completed yet. Static image extraction component detects any movement over the chessboard, waits for the movement to complete, then raises event which notifies other components about static board situation.

Static board situation is detected by comparing neighboring or nearby video frames. When frame difference is below specified threshold, for a specified number of frames, it is considered that static image has been detected.

Component can operate on two channels simultaneously (top and side view camera), raise separate events for each channel, or both channels simultaneously (when static image is detected on both).

Image difference is rated by subtracting pixel values across entire images, or by limiting subtraction to a specified region of interest (exact chessboard location on the video stream). Limiting image difference rating to a region of interest eliminates part of camera noise, which decreases problems with false detections. Images can be preprocessed to suppress noise (Gaussian blur).

Initial video frame is stored as a reference frame. Reference frame is then compared with next frame in sequence. If rated difference is above specified static threshold, it is assumed that movement is currently present at connected video stream. Reference frame is deleted and the next frame is set as a new reference frame. Process continues until rated difference between reference frame and next in sequence drops below static threshold (assumingly, due to lack of movement in front of the camera). Processing advances to the next frame, but old reference frame is now kept. While rated difference between reference frame, and newest in sequence is below specified static threshold, old reference frame is kept. When specified number of frames (with rated difference in relation to the reference frame below static threshold) is processed, static condition is declared on that camera channel. Event which indicates static board condition is raised, and an image from the middle of the frame interval between reference frame and newest processed frame (one which triggered static condition) is delivered to other components subscribed to receive static condition indication events.

Static condition is signaled only once. Reference frame is replaced as soon as new movement occurs. Described algorithm ensures that a slow movement can't be mistaken for a static board condition. However, if a player holds his hand very still over the chessboard, obscuring the camera view, false static condition can and will be triggered. This problem is possible but rare if normal game play is maintained.

Possible system upgrade could include hand color detection, but hand color can be similar to color of figures, so only large enough connected components should be detected as a human hand. Another upgrade could be implemented using board detection itself (with every board field and line), constantly running, to indicate hand over the chessboard, when there is a failure detecting some of the board elements). This approach is very computational complex, and needs to be tested, if capable to run in real time. Most simple possible upgrade is adaptive static threshold computation (to compensate for the lighting changes).

Most important component parameters are static detection threshold T and number of static frames required to declare a static board condition N. Optimal parameters need to be determined for both camera channels. For the purpose of testing, only one channel is used. Supplied video resolution is 640×484 pixels at 5 frames per second. Entire video sequence contains 74 real chess moves, with at least 2 seconds delay between moves.

First test is displayed at Figure 3. Chess figure movement successful detection rate is observed when varying number of required static consecutive frames needed to declare a static board condition N. Static detection threshold T is set fixed to 4×10^{6}. Static detection threshold T is a unitless value, as it represents total difference between all RGB pixel values of two compared frames. In this non-normalized form, number is dependent on video resolution and needs to be recalculated if video resolution changes.

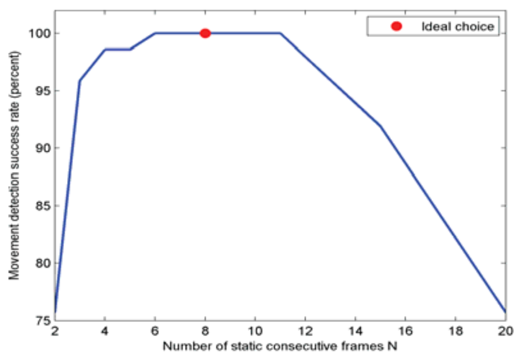


Figure 3. Chess figure movement detection success rate, when varying number of required static consecutive frames N, using fixed static frame threshold ($T = 4\times10^6$).

It is determined that the optimal number of required static consecutive frames N is 8.

Second test is displayed at Figure 4. Chess figure movement successful detection rate is observed when varying static frame threshold T. Number of static frames required to declare a static board condition N is set fixed to 8.

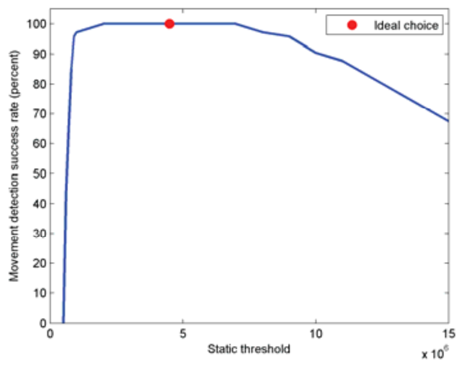


Figure 4. Chess figure movement detection success rate, when varying static frame threshold T, using fixed number of required static consecutive frames (N = 8).

It is determined that the optimal static frame threshold value T is about 4.5×10^6.

Although optimal component parameters are determined as above, by observing Figures 3. and 4. it can be seen that both parameters can be set to a broader range of values and still achieve 100% accuracy.

If the players played chess even faster than 2 seconds between moves, or if video frame rate dropped below 5 frames per second, graph segment displaying 100% success rate on Figure 3. would become narrower, and finding right parameter would be more difficult. If camera noise would increase, graph segment displaying 100% success rate on Figure 4. would become narrower, and finding right parameter would be more difficult.

It is shown that this simple algorithm can effectively detect chess moves, in non extreme conditions (with constant lighting).

B. Board detection

Before any further processing, it is necessary to locate the board, and the coordinates of individual squares. With the restriction that the board will not move, it is sufficient to perform detection only once at the beginning. After the board is detected, the functionality to determine which pixel belongs to particular square is provided. Such functionality is needed in further steps of processing and analysis.

To avoid the influence of chess pieces on the board detection, the restriction that the board must be empty at the beginning is introduced. Before the process of board detection, the image preparation and preprocessing is needed. As the squares are red and green (a contrasting colors), the first step is to extract the red channel. Ideally, the red squares would have maximum value while the green squares would have minimum value of lightness intensity. Because of influence of the noise, that's not the case, and to obtain the binary image the adaptive threshold is used. For the purpose of robustness, the adaptive threshold parameters (window size and a subtraction constant) are determined iteratively at the runtime. After obtaining the binary image, the board detection is performed.

The following approaches are considered and tested:

- Connected components - rejected due a lower percentage of success; the problem of determining borders and mapping of the squares.
- Hough transform - rejected because the method is unsuitable due to image distortion (lines are not straight); extremely sensitive to noise, the problem of grouping lines.
- OpenCV embedded function - the chosen method, with certain modifications; validation and extrapolation.

For detection of internal corners of the chessboard the OpenCV built-in function cvFindChessboardCorners is used. The function was originally intended for calibrating the camera, but due to its specific purpose it can be applied to a given problem.

Although the function uses an adaptive threshold, which itself also adjusts the parameters, it turns out that function itself gives very poor results over the original image. Therefore, the function uses already prepared binary images.

As the function only finds the inner corners, it is necessary to extrapolate those on the border. Extrapolation of external corners is done by reflection of the second layer over the first layer of the inner corners.

Higher order extrapolation is possible but it gives worse results due to errors in the corners positions.

It's not uncommon that the function returns a set of corners as a complete and correct, but it's actually deformed and incorrect. It is therefore necessary to validate the returned set of corners. Validation is done in a way to traverse through the points (corners) while trying to locate a new point on the current line. If the new point deviates more than a given threshold from the current line, the point is declared as the beginning of a new line. After traversing all the points, the number of lines is checked to be within the given limits. The parameters of the current line are obtained as the best approximation of a set of points by a line. Threshold is expressed as the relative measure compared to the average length of a line segment that is given by a ratio of a line length and the number of corresponding points.

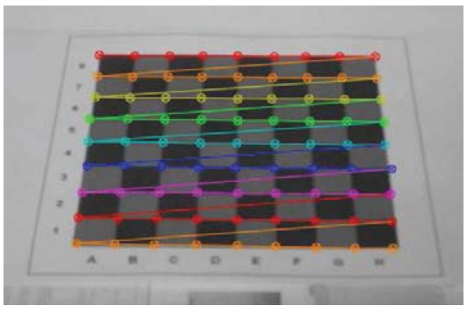


Figure 5. The correct set of corners.

C. *Feature extraction*

In order to determine the change on the board after one players move, two adjacent images (ones that determine this particular move) are subtracted. For example, on Figure 6. we can see that subtraction of previous and current state image, (a) and (b) gives result image (c) that is further converted in the grayscale image. Simple RGB to grayscale conversion results in (d). It can be seen that the red part of subtracted image is more prominent than the green one. Therefore, the RGB subtraction image is transformed into HSV (hue, saturation, value) space that is color-invariant and the V-channel (e) is used for further processing. Next step is image binarization in order to eliminate noise and shadow. Later steps showed that the higher threshold is better, despite the fact that some figure pixels will be lost.

In order to extract figure that had been moved, we should find minimal perimeter rectangle that contains key figure. The idea is to search the result binary image (f) pixel by pixel in order to determine coordinates of rectangle that contains the most of white pixels. The search is performed with pre-defined rectangle sides and search-step, i.e. the length of the rectangle shift after one iteration. Using lower values of shift will result in more accurate, but slower search. After complete search, the coordinates are found and the most of figure pixels are in rectangle defined with those coordinates (upper left corner) and its predefined side lengths. Next step is adjusting of rectangle side lengths. As long as there is a single outer white pixel near some of the rectangle side the rectangle is expanding. On the other hand, as long as there aren't any white inner pixels near some of the rectangle side the rectangle is contracting. The resulting rectangle is shown on (g).

Next step is to find whether the rectangle contains some figure or is empty. This can be easily done by subtracting image of the current position and image of empty chess board within area defined by the rectangle. If the energy of the subtracted part of picture is larger than some threshold, we assume that at current point, some figure is placed on area defined by the rectangle. Part of current state image bounded by the rectangle is used as classifier input and has to be classified. If the energy of subtracted image is lower than threshold, the reign of interest (part of image within the rectangle) is considered empty and no classification is needed since we know that the field is empty.

Previous step shows how to deal with only one change between two moves in a game. But, in a particular move, there are always two changes on the board since figure that has been moved by a current player has its start and end position, hence the change appears in two places. In order to find the next region of interest (part of image within the new rectangle), one should repeat previous procedure. Without any modifications, that would be useless, hence the same region of interest would be found. So, prior to repeating the procedure all pixels within the first region of interest should be set to zero, thus, they will represent an area with zero white pixels and repeating the procedure new region of interest will be found. This region represents the other area where change occurs.

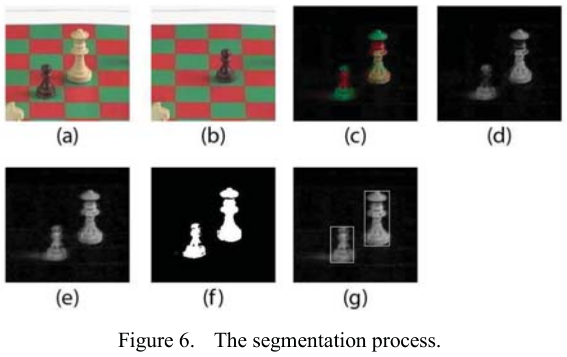


Figure 6. The segmentation process.

Problem of this approach is the situation when figures are overlapping. In this situation, the algorithm would find only one region of interest which would consist of real two regions of interest. So, it is impossible to determine which field is empty -start position of current player's figure and which field is its end position. For this purpose upper camera was used because from its position it is easy to determine which fields are empty and which are not.

D. Detection of figure position

One of problems that must be solved in order to create chess recognition system is to determine figure positions on chess board. Viewpoint of top camera gives good view on figure positions on chess board because overlap of figures is not possible. First image from top camera is transformed to gray scale color scheme and then process of detecting figures is conducted for each field of chess board where the energy of field is compared with empirical defined threshold. If energy of field is less than threshold then field is empty and in other case it is occupied with figure. Energy of field is calculated based on sum of pixel values that belong to field. Value of pixel is defined on domain from 0 to 255. Where 0 represents black and 255 white color. Component is using result of board detection component that consists of coordinates of chess board fields. Based on coordinates of field it is possible to connect field in picture with field position on board, so as result of detection of figure positions the matrix is produced. Matrix dimensions are as dimensions of chess board (8x8) and each element defines one field with value 1 if field is occupied with figure and 0 if it is not. During implementation we used test set of pictures from top camera to define best threshold. Threshold is not uniquely defined and it depends on real environment and illumination of chess board.

E. Figure recognition

As there are six different figure types and two colors, the segmented figures are classified into one of 12 distinct classes. To obtain images suitable for training we used move sequences that have only one of the figures on the board. The preparation of move sequences was manual so we could obtain relatively low number of training images for each class. For that reason we decided to use Support Vector Machine (SVM) classifier [9][16] that shows good results even with the low number of training samples [10]. We used implementation provided with the OpenCV library.

The quality of the classification was tested on 96 image samples obtained by the segmentation process. Feature vector for each sample is obtained by converting each image to grayscale representation, scaling it to 32x32 pixels and then putting each row side-by-side. No additional feature extraction was performed. The resulting 1024 elements vectors with values ranging from 0 to 255 were used as an input to SVM classifier.

The Gaussian radial base function was used as a kernel for transforming the samples

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

where γ determines the width of the radial function. Large values of gamma correspond to narrow radial functions that too specific and usually result in large number of false negative classifications. In comparison, small values correspond to wide radial functions that are not specific enough, and so produce large number of false positives. The weight coefficients w and offset b should satisfy constraints

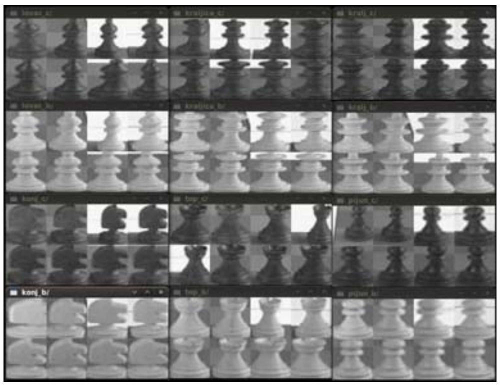


Figure 7. Image samples obtained by our segmentation method that we used for training the SVM classifier. There are eight samples for each of 12 classes.

$$d_i(w^T x_i + b) \geq 1 - \xi \quad za \;\; i = 1, 2, \ldots$$

where ξ is the penalty for classification into a wrong class. The optimal weight coefficients minimize the criterion function

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

where the C parameter determines the penalty for each sample classified into a wrong class. Penalty is crucial for avoiding the over fitting of the classifier on the training samples.

To find optimal values for the parameters γ and C, the initial set was divided into 60 samples for training and 36 samples for testing (ratio of 5/3 for every class). Figure 8. shows that γ parameter with value from 0.1 to 10 performs equally well regardless of the penalty C, but that even better results could be achieved by using γ of 0.01 and penalty C appropriately high.

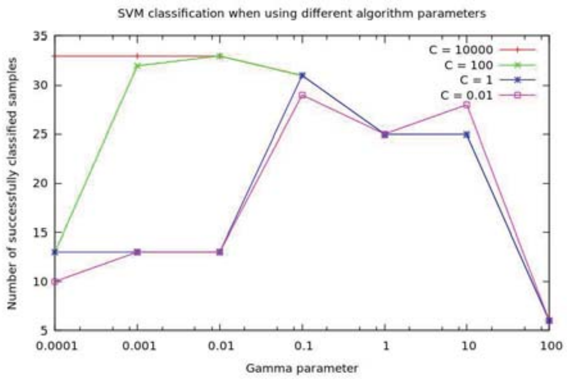


Figure 8. Number of succesfully classified samples in dependence on the parameters γ and C.

The optimal parameters (γ = 0.01 and C = 100) were tested with varying ratio of training/testing samples. Figure 9. shows that good results are achieved even with the relatively low number of training samples. The fluctuations for ratios 6/8 and 7/8 are due to the low number of testing samples.

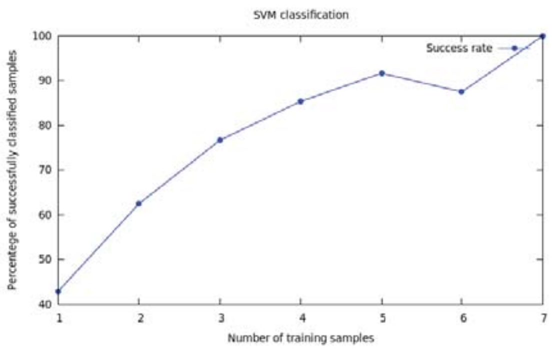
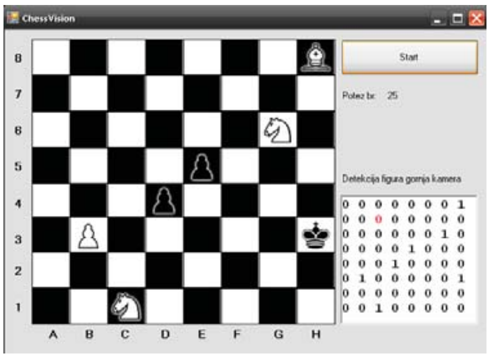


Figure 9. Percentage of succesfully classified samples when using different number of training samples.

F. *Graphical interface*

Figure 10. shows graphical user interface of chess vision project. It consists of virtual view of chess board with figures that represents state of chess board in real world. In bottom right corner is box that represents abstract view of chess board. Each element in box represents one fields of real chess board. 0 means that field is empty and 1 means that field is occupied with figure. The information is obtained from the „Detection of figure positions“ component. In top right corner are the button for next move and label that shows number of played movements since beginning of chess match.



Potez br: 25

Detekcija figura gornja kamera

```
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
```

Figure 10. Graphical user inaterface for the ChessVision system.

III. Testing and analysis

To test the efficiency of the system we performed analysis on the video sequence of 63 moves. Majority of the moves do not correspond to the legal chess moves, so there are situations where multiple figures appear or disappear from the board. In the following chapters we describe in detail some of the typical situations and give approximate statistics of success.

A. *Example test cases*

The Figure 11. shows several examples of how the board state changes between the moves. Figures (a) and (b) show a move where white bishop from the field C5 moves to the field C7. Although it now covers the black queen on the field C8, the result on the figure (c) shows that the system correctly identifies the situation because it is apparent from the top camera that the queen did not change the position, so it is safe to assume that the queen is in the same position as the move before.

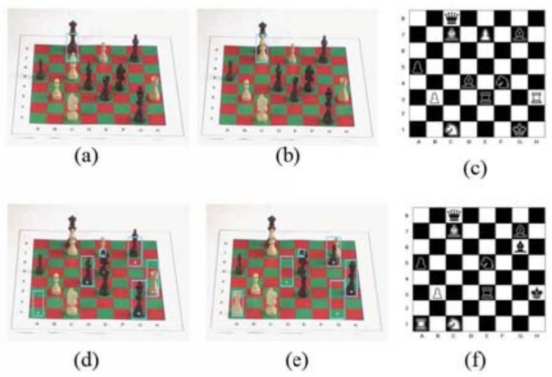


Figure 11. Two examples of chess moves. Images (a) and (d) show the states before the move, (b) and (e) after the move, and (c) and (d) states after the move as recognized by the system.

Figures (d) and (c) show a different sequence of moves where several figures change position. In this case the system correctly identifies the disappearance of black king on the field G1, black bishop on field D4 and white rook on the field A1, but it fails to detect the swap of black bishop on the field G7 with the black pawn on the field G6 and the white pawn on the field G7. Instead, it identifies them as two black bishops. The reason is that the occupancy of the field G7 didn't change (it is still occupied but with a different figure) and because of the overlap of the two figures the segmentation subsystem incorrectly classified front black pawn as a black bishop.

B. *Analysis of achieved accuracy*

Table I. shows the success rate of various parts of the chess reconstruction process as obtained from the test video sequence. As expected, rather high success rate can be achieved when there is no significant overlap between the figures as viewed from the side camera. The same is valid for the process of segmentation and recognition. On the other hand, because of the use of top view camera, we achieved perfect results when detecting the occupancy of a particular field.

TABLE I. Succesfully identified events

Move reconstruction (no overlap)	42 / 44 = 95.5 %
Move reconstruction	53 / 63 = 84.1 %
Figure segmentation	58 / 63 = 92.1 %
Figure classification	57 / 63 = 90.5 %
Field occupation detection	63 / 63 = 100%

IV. Conlusion

The goal of this paper was to describe implementation of a system that is able to reconstruct a chess board state based solely on video sequences obtained from two cameras. We have used standard chess figures with adopted chess board (red and green squares). Whole system was written in C++ and OpenCV library was used. System is divided into few modules that communicate with each other over a controller module. Extraction of static pictures delivers picture that identify each figure move. For each move we detect positions on the board from top-view camera and extract figures bounding box from side-view camera. Recognition is made by using kernel SVM classifier. Our chess vision system is a memory state system which updated chess states only for new moves. Therefore if new moves cover old figures in side-view camera memory is used for chess state reconstruction. But if a new move is covered with old figures chess state reconstruction from side-view camera is not possible. Top view camera is only used for detection of moves in a chess game. Chess vision system showed solid performance on our limited number of video sequences.

Acknowledgment

This project was made as a part of "Machine vision" class on the Faculty of Electrical Engineering and Computing, University of Zagreb, that authors attended in the winter semester of 2009/2010. We would like to thank Professor Slobodan Ribarić and assistant Marijo Maračić for their generous help and assistance during the course of the project.

References

[1] A.K.Farahat, A.M.Hassan and M.A.El-Nagar, A Vision System fo Chess Playing Robots, 46th IEEE Midwest Symposium On Circuits and Systems, December 27-30,2003

[2] David Urting, Yolande Berbers (2003), MarineBlue: A Low-Cost Chess Robot, Proceedings of the IASTED International Conference on Robotics and Applications

[3] H. Zhu, J. Lei and X. Tian, A pattern recognition system based on computer vision — The method of Chinese chess recognition, Granular Computing, 2008. GrC 2008. IEEE International Conference, doi: 10.1109/GRC.2008.4664655

[4] P. Hu, Y. Luo and C. Li, Chinese Chess Recognition Based on Projection Histogram of Polar Coordinates Image and FFT, Pattern Recognition, 2009. CCPR 2009. Chinese Conference, doi: 10.1109/CCPR.2009.5344001

[5] J.E. Neufeld, T.S. Hall, Probabilistic location of a populated chessboard using computer vision, Circuits and Systems (MWSCAS), 2010 53rd IEEE International Midwest Symposium, doi:10.1109/MWSCAS.2010.5548901

[6] 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, doi:10.1109/WISM.2010.124

[7] S. Zhao, C. Chen, C. Liu, M. Liu, Algorithm of location of chess-robot system based on computer vision, Control and Decision Conference, 2008. CCDC 2008. Chinese, doi: 10.1109/CCDC.2008.4598325

[8] S. Blunsden, "Chess recognition", undergraduate dissertation, University of Plymouth, 2003.

[9] C. Lin, C. Hsu and C. Chang, "A practical guide to support vector clasiffication", Technical report, Department of Computer Science, National Taiwan University, July 2003.

[10] A. Kaehler and G. Bradski, "Learning OpenCV", O'Reilly, 2008.

[11] OpenCV 1.0, "OpenCV 1.0 api reference", URL: http://cgi.cs.indiana.edu/~oleykin/website/OpenCVHelp/

[12] OpenCV 2.0, "OpenCV 2.0 api reference", URL: http://opencv.willowgarage.com/documentation/index.html

[13] B.G. Schunk, R. Jain and R. Kasturi, "Machine vision", MIT Press and McGraw-Hill, 1995.

[14] S. Ribarić, "Materijali za kolegij Računalni vid", unpublished

[15] T. Graf, A. Knoll and A. Wolfram, "Fuzzy invariant indexing: a general indexing scheme for occluded object recognition", Signal Processing Proceedings ICSP '98, vol. 2, 1998, pp 908-911

[16] J. Shawe-Taylor, N. Cristianini, "Support Vector Machines and other kernel-based leraning methods", Cambridge University Press,2000.