Digitization of a Chess game using a moving camera

Abhishek Venkataraman, Atulya Shivam Shree, Ganesh Sevagamoorthy

abhven@umich.edu, satulya@umich.edu, gse@umich.edu

Abstract

With the recent craze for AR/VR games like Pokemon Go, developing mobile phone games which interact with the real world has become widely popular. Thereby, the need for applications that can process images from a hand-held camera, and are viewpoint invariant, is quite high. In this project, we have developed an application that can detect the moves of a chess game using a moving camera. We test our algorithm on a video recorded from a hand-held device and are able to detect the moves correctly even with small changes in the camera position. A plausible application of this would be by players who wish to record their game for maintaining records. Another application would be to input the moves to a chess-game engine and analyze or improve the game strategy.

Keywords: Chess, View-point invariance

1. Introduction

Board games such as chess, are widely popular throughout the world. With a simple mobile phone camera and some AR tags we have developed an application that can provide a digital interface to the game. Most players like to analyze their games and in order to do this, they would have to write down their moves. This is a cumbersome process and they cannot focus completely on the game. Our application can help them record the moves made in the game in the hand-written chess notation, for later reference. Being able to extract data from a game opens up a wide range of tools that can be built on top of it. If it works in real time, we could augment the board with virtual information on a screen. Players could also get advice from an AI chess engine after every move. A similar approach could be applied to different board games and would prove to be a game changing experience.

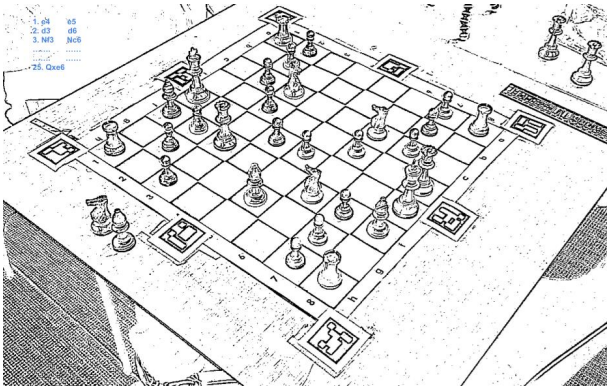


Figure 1: An artistic rendering of the chess game using open-CV

In this report, we discuss the development of our project on chess recognition for the course on EECS442 at University of Michigan. It has been organized in the following order: Section 2 talks about the various approaches used to solve this problem and how our approach is different from the existing ones. Section 3 details the algorithms used in sub-tasks, and the flow of data for achieving the end result. In section 4, we demonstrate the working on set of videos which we had created. Finally we present our remarks and conclusion in 5.

2. Review of previous work

In the work done in [1] the authors have taken a monopoly board game and by detecting the board and the pawns include virtual information into it. This approach allows a user to get helpful statistics of the game augmented on a screen. There has also been work done to build a completely virtual chess game and provide all information directly to a user's head mounted display in [2]. In [3] work has been done on using bare-hand gestures such as pinch and release to interact with the virtual world. While including real world interaction with the hand is stimulating for players, we think that interacting with real world pieces could be a better alternative. In [4] the authors have attached fiducial markers to each of the pieces and are tracking them using a camera. They can then play a game where the user moves the real pawns while wearing an AR headset. The moves of the opposition are virtually added into the display. A similar work has been done in [5] with the key difference being that here all the pieces move with the aid of some electronic actuators. These are good approaches for making the game interactive, however it requires having special pieces and an additional overhead/undertable camera for tracking the pieces.

We found a project on [6] where a similar project had been implemented at the Stanford University. The authors are able to detect real piece movements on a chess board. However they use a fixed camera for their identifying movements. While it

helps in the detection process it would not work if the camera was present in either a head mounted display or was being moved around by a person.

2.1. Problem Definition and Assumptions

We define our objective as to record the moves of a chess game being played between two players. Since we want our approach to work on hand-held devices as well we use a mobile phones camera to capture the game. For simplicity the pawns are assumed to be of a different colour as compared to the colours of the chess-board. We also assume that fiducial markers can be kept at known location and orientation around the board.

The major differences in our approach are:

- Use of a moving camera for detection
- No fiducial markers on the pawns, we only have them on the chess board
- No need for a camera over or under the board.

However we make a key assumption that the move detection engine is manually triggered by an operator. Ideally we would want to have another function for detecting the presence of hand and use that to trigger the move detection. However we leave that outside the scope of this project and just inform the algorithm when to look for moves manually.

3. Methodology

3.1. Overview

Our implementation has been done on videos that are recorded from a smart-phone camera. The programs are written in python using open-CV libraries. Each frame in the video is first processed by the board detector, followed by the corner detector. The image is then split into sub-images and these are analyzed to detect the moves. Finally, the moves are then fed to the Chessnut library [7] to validate and then store the game in Forsyth-Edwards Notation. Chessnut maintains only the state of the board and it does not interact with the CV part of the implementation. Each step in the pipeline is discussed in detail in the following subsections.

3.2. Board detection and Isolation

For any detection to work on the chessboard it is necessary to work in the coordinates in the board alone. To do this we isolate the board from the remaining background clutter so that further detection algorithms can work on it. We use Aruco markers from [8] for computing known locations in the image. The AR tags have a 5x5 binary message stored in them. This gives us a high confidence that the quadrilateral we are viewing is a projection of the square. In addition different AR tags can also be uniquely identified using their IDs.

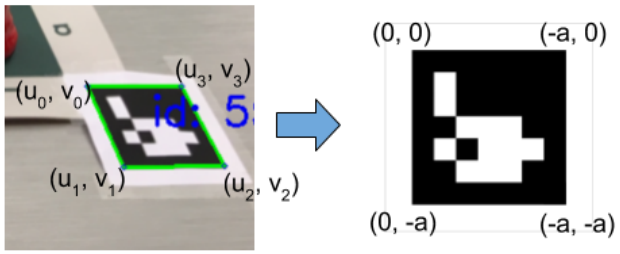


Figure 2: Point correspondence between real world and image coordinates

3.2.1. Computing the Homography matrix

Fig 2 shows the corner points of the tag detected in the image frame and their corresponding real world coordinates. If we are able to compute more than 4-point correspondences we can compute the homography which transfers the board to real world configuration. This is given by the following equation:

$$\begin{bmatrix} \vec{x}_i \\ 1 \end{bmatrix} = H \begin{bmatrix} \vec{p}_i \\ 1 \end{bmatrix} \tag{1}$$

where $\vec{p}_i$ is the image coordinate of the corner and $\vec{x}_i$ is its real world coordinate. If we can detect n markers we get $4n$ correspondences and use them for computing the Homography transform.

Figure 3: Detecting markers around the board

3.2.2. Computing a score for the homography transform

Using too few markers can lead to a homography which has large errors and is insufficient to detect the board. Having multiple markers which are almost along the same line also leads to inaccurate homography matrix in the direction which has less variance in measurement. Hence we need to define a score which rates the marker detection on the basis of variance in measurement along different directions. To compute it lets assume that the real world point correspondences are given by $\vec{x}_i$,

where $i \in 1, .., 4n$ with n being the number of markers detected.

$$\begin{aligned} Cov &= \sum_{i=1}^{4n} \frac{1}{4n}(\vec{x_i} - \bar{x})(\vec{x_i} - \bar{x})^T \\ \lambda_1, \lambda_2 &= eig(Cov) \\ R &= \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2} \end{aligned} \tag{2}$$

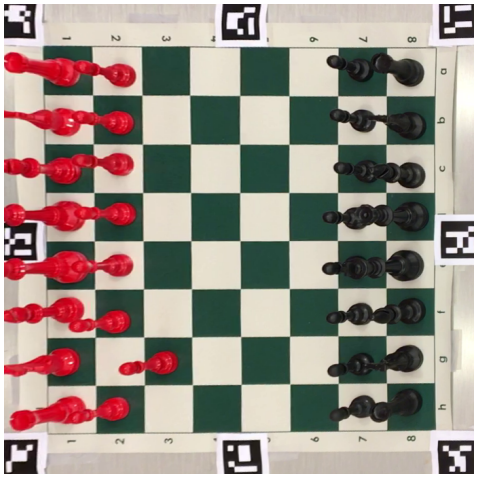


Figure 4: Board isolation with a good R_score of 142.16

Fig 4 shows the result of a good detection. Qualitatively we found that thresholding the R score with a value of 0.5 led to good final images.

3.2.3. *Multi-stage detection*

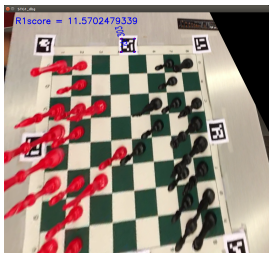
For the scenarios where we detect a very low R_score we attempt to detect more markers by using a cascade of detections. We use the H matrix computed in the initial stage to apply a homography transform to the image. Next we attempt to detect markers in the resultant image. Since the second stage input image is closer to the ideal image we expect the number of markers detected to increase.

Fig 5 shows the advantage of using multiple markers in the image. The AR tags on top of the board are either occluded, blurred are too skewed to be detected by the Aruco Library. Using multi-stage detection allows us to detect 2 more markers in the resultant image. The final output is more closer to the ideal transform as compared to the first stage output.

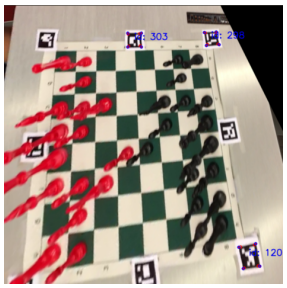
3.3. *Corner Detection*

After isolating the board from background, the next step is to segment it into sub-images. A good candidate for the sub-images would be the individual squares in board. These squares provide the information that case be used for move detection, piece detection, occupancy mapping, etc. Getting the corners accurately ensures that the squares are extracted accurately. We tried two methods, viz. Harris Corner and Template matching for corner detection.

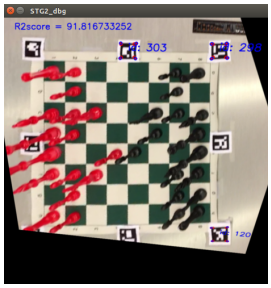
(a) Marker detection on input



(b) Homography transform using 1 marker



(c) Marker detection on output of stage 1

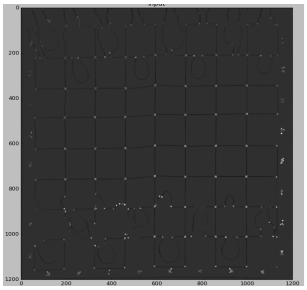


(d) Output after recomputing H and applying new transform

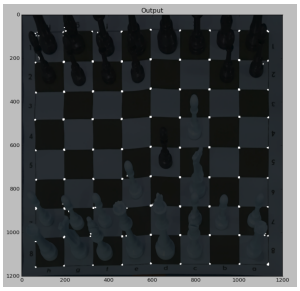
Figure 5: Multi-stage marker detection to improve image transform

3.3.1. *Harris Corners*

Harris Corner detector is a commonly used for detecting corners. It has a simple implementation has and the execution time is quite small. However, the detection accuracy is highly dependent on the selection of the threshold value, which is very hard to guess. Figure 6a , shows a large number of detection that are not of interest. The accuracy was improved by searching in a smaller windows around the expected corners and can been seen in Figure 6b. However even after improvising the detector,



(a) Standard Harris corners



(b) Localized Harris Corners

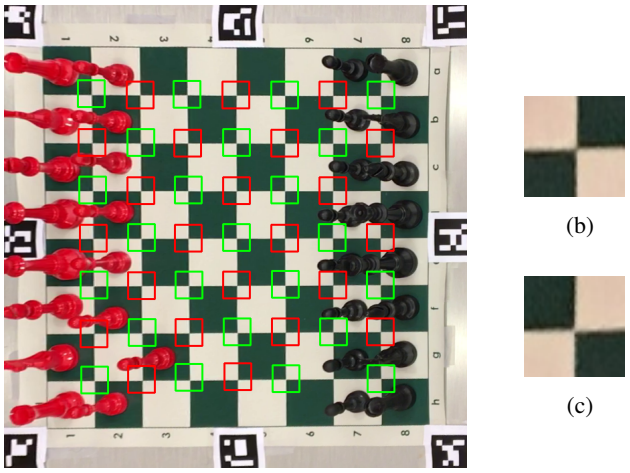
Figure 6: Detection of Corners using Harris Corners

the number of false positives is quite high. This becomes quite difficult especially when the corners are occluded.

3.3.2. *Template Matching*

Template matching is a technique used for finding small parts of an image which match a template image. In case of chessboard, the unique arrangement of the squares allows template matching to give a very good results for corner detection. Figure 7a shows the corner detection using the templates, Figure

7b and 7c. It can be seen that the detection accuracy is very high as compared to Harris Corner detection.



(a) (b) (c)

Figure 7: Template Matching done on image (a) using 2 templates (b) and (c)

A critical parameter for accuracy of template matching is to choose the size of template. A large template (size) increases the accuracy of detection. As a consequence, it also reduces the number of detection. The reduced number of detection becomes more pronounced when there are large occlusions. In our case, for an image of size ~700x700pixels, we have used a template of size 10x10 pixels.

Template matching gives multiple responses around the matching area. It is difficult to get a single response by just changing the threshold. We have used spacial clustering to bin responses within 10pixel radius and use the average of the bin as the final corner.

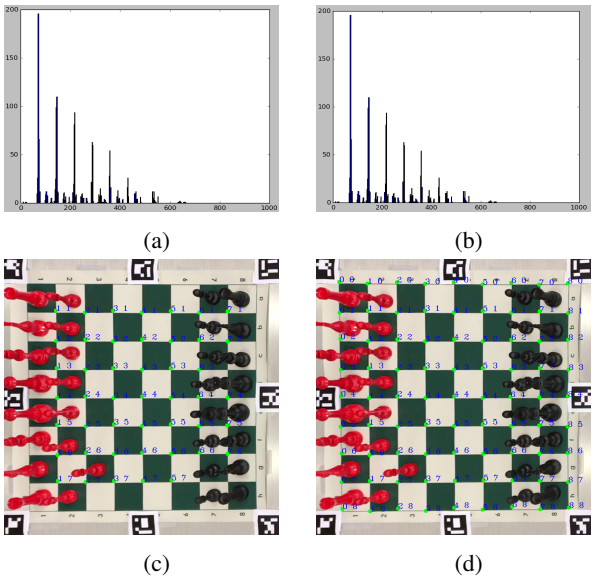
3.3.3. Estimating Missing Corners

Both the above methods work only where the corners are visible and not occluded by pieces. Additionally, the template matching algorithm provides the image coordinates of the corner, but does not specify the grid coordinates[1]. In order to solve this, first, x distance and y distance between every pair of corners is computed. The distances are then clustered into bins as shown in figures 8a, 8b . The average value of x lengths in bin with the maximum elements is then taken as the average grid length in x direction. In a similar way, the grid length in y direction is obtained. Using the grid lengths, each of the above corners is assigned a grid coordinate as shown in figure 8c.

Once we have the grid coordinate of the existing corners, it is easy to get the know which corners are missing. First, the seed corner[2] is chosen based and the missing corners in (x,y), $\{x, y \in \{1, 2.., 7\}\}$ are obtained by interpolation/extrapolation.

[1]Grid coordinate system has the origin at red side left. The square length is one unit. Hence x, y take values between 0 to 8. (0,0), (0,8), (8,8), (8,0) are the corners of the board.

[2]Seed corner is the corner with detected neighbors among (x,y), where $x, y \in \{1, 7\}$



(a) (b) (c) (d)

Figure 8: (a)Histogram of x lengths (b)Histogram of y lengths (c)Grid coordinates for detected corners (d)Complete set of Corners

Next the using the 49 corners, the corners in the periphery, (x,y), $\{x, y \in \{0, 8\}\}$ are found using extrapolation. The complete set of corners, after the above step is plotted in figure 8d.

3.3.4. Square Extraction

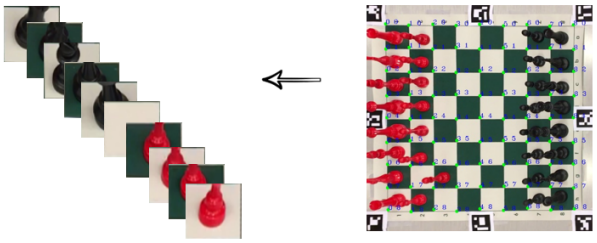


Figure 9: Logic flow for color classification

The corners of each of the 64 square can be queried from the 81 corners computed in the previous step. These corners are then used to project it onto a 60x60 pixel square and they are indexed as per standard chess notation[3].

3.4. Color Clustering

Each of the 64 square images obtained from the previous steps are clustered into 4 colors, viz. red, black, green and white using the decision tree shown in figure 10. A 4 dimensional color vector, {W,G,R,B}, containing the count of each color is then generated for each square.

3.5. Move Detection

From the previous step, we get the color vector containing the number of pixels of each color in every square in a particular

[3]Standard chess notation labels the squares as a1, a2.., h8. The row index is a,b,...,h and column index is 1,2,...,8

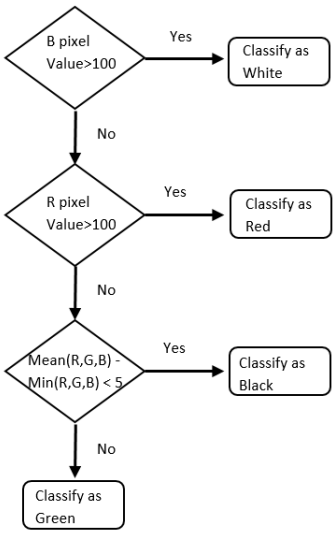


Figure 10: Logic flow for color classification

frame. A frame before the move and after the move are taken and the color vector of the old frame is subtracted from the color vector of the new frame for every square. We take the differences between the red and black components of the vector to construct one heat-map each for red and black pieces.

This difference is then normalized to be between 0.5 and -0.5. This normalized difference is compared to a threshold and only the squares where a significant change has occurred are kept. In the squares where there is a significant change, if the normalized difference of a particular color is positive, then we can say that the particular colored piece has moved to that square or around it and if the value is negative, then a particular colored piece has moved away from that square or around it.

Based on whose turn to move, we calculate all possible moves based on the heat-map by taking all combinations of the squares where the color is removed and the squares where the color is added. It is usually the case that colour information has been added to more than square or removed from one or two squares. This is so because every piece is present in more than one square. A score is calculated for all possible moves using the information about the colors removed and added. If it is reds turn and a move has been made, then information about the start of the move is calculated based on the squares where the there is a significant reduction of red color and if it is the case, the squares around which significant amount of black color has been added. Similarly the end point is calculated based on the addition of red color to a square along with reduction of black color in that square and around. This is done with the using algorithms 1,2,3.

The first algorithm is used to calculate the score of the start point, here the squares with significant reduction in red color is taken and is summed with the black color added to the squares around it but the square itself till no more significant amount of black is added. The second algorithm is used to calculate the score of the end point, here the squares with significant addition of red color is taken and is summed with the black color reduced

Algorithm 1 Determining score for start points

Require: $cellscore_{color,(x,y)}, cellscore_{color^c}$

1: **for** n in $range(1,7)$ **do**
2: $Contribution_{color^c} = Contribution_{color^c} + \frac{1}{n}$ (Sum of $cellscore_{color^c}$ for squares distance n from $square_{(x,y)}$)
3: **end for**
4: $Startscore_{(x,y)} = cellscore_{color,(x,y)} + Contribution_{color^c}$

Algorithm 2 Determining score for stop points

Require: $cellscore_{color,(x,y)}, cellscore_{color^c}$

1: **for** n in $range(0,7)$ **do**
2: $Contribution_{color^c} = Contribution_{color^c} + \frac{1}{n}$ (Sum of $cellscore_{color^c}$ for squares distance n from $square_{(x,y)}$)
3: **end for**
4: $Stopscore_{(x,y)} = cellscore_{color,(x,y)} + Contribution_{color^c}$

in the square and in the squares around it till no more significant amount of black is reduced. The score for the start and stop are multiplied to get the score of a particular move. The move with the highest score is taken and it's validity is checked with the past history using Chessnut. If the move is valid, then that move is given as the output, else the move with the next highest move is checked and so on. The move is applied to Chessnut to update the state of the game and the piece that has been moved can be retrieved from Chessnut.

4. Experimental Results

We collected 3 videos of chess game with our setup using an smart-phone camera and label them as A,B and C. The video sequences A and B were taken at a distance of around 50cm horizontally and 50cm vertically from the board edge. The tilt angle of the camera is almost always facing towards the board in a downward direction for each of the videos. The azimuth has a variation of almost $30° \pm 15°$ in sequence A, around $90° \pm 15°$ in sequence B and almost $\pm 15°$ in C. In video sequence C the person is standing still and trying to hold the phone as steady as possible. The video sequences are 62, 100 and 94 seconds long.

4.1. Board Isolation

We use the sequence B for computing the performance of board isolation. Fig 11 shows the result of R_scores over all frames. If we threshold the good images at 40 we observe that 6.8% of the images are classified as bad while we are able to view successfully isolate the board in 93.1% of the images.

Algorithm 3 Determining score for moves

Require: $Startscore_{(x,y)}, Stopscore_{(x,y)}$

1: $Movescore_{(x,y)} = Startscore_{(x,y)} \times Stopscore_{(x,y)}$

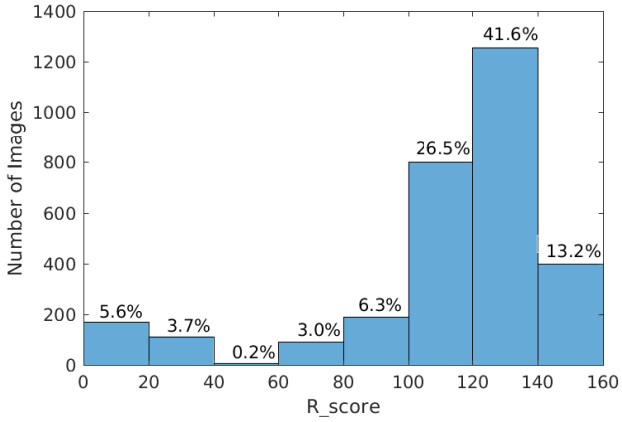


Figure 11: Histogram of R scores for a video with 3025 frames

The cases where we could not get a good score were those which had a lot of skew or are blurred due to motion. Table

	Case	Count	%
1	Total frames	3025	
2	$R_1 > 40$	2744	90.7%
3	$R_1 < 40, R_2 > R_1$	74	2.4%
4	$R_1 < 40, R_2 \leq R_1$	207	6.8%

Table 1: Detection scores for board isolation

1 shows the result of our board detection algorithm. We classify cases 2 and 3 as successful detection and case 4 as bad. Using the multi-stage detection allows us to increase the detection by 2.4%. Hence it is to some extent effective in increasing the number of successful isolations.

4.2. Color clustering

Figure 12 shows the implementation of the color cluster for the whole board. In order to have faster execution time, the algorithm used operations at image level, instead of accessing individual pixels. The execution time per frame was ~0.08s

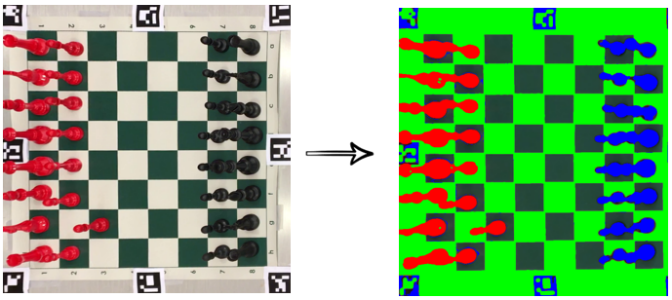
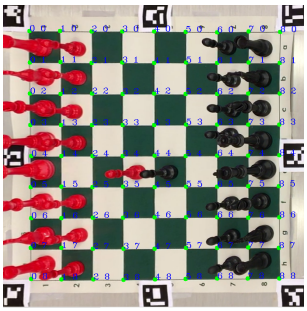


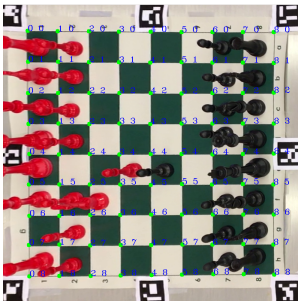
Figure 12: Logic flow for color classification

4.3. Move Detection

The frames before and after a move made on the red piece is shown in fig 13a and fig 13b respectively. The heatmaps for the two positions is shown in 13c. The blue squares in the heatmap indicates that a particular color has been removed from the square and red squares in the heatmap indicates that a particular color has been added to the square. The intensity of the red and blue color in a square is proportional to the the number of pixels added or removed from the square.



(a) Frame before move



(b) Frame after move

(c) Heatmap for a move. Red heatmap(left) and Black heatmap(Right)

Figure 13: Generation of heatmap corresponding to a move

For the squares with intensity above the threshold, we calculate scores using algorithms 1,2,3. The scores for all possible start and end moves are shown in table 2. All possible moves are calculated by taking all the combinations of start and stop squares. The scores for each of the move is calculated and is shown in table 2. The moves with highest score is taken and

	Start	Score
1	g1	0.3130

(a) Scores for Start squares

	Stop	Score
1	f2	0.1800
2	f3	0.2988

(b) Scores for Start squares

	Move	Score
1	g1f2	0.0877
2	g1f3	0.145625

(c) Scores for possible moves

Table 2: Scores for start point,stop point and possible moves

compared with Chessnut to validate the move. In this case the move 'g1f3' has the highest score and it is valid move and it is given as the output.

4.4. Overall Performance

We run the entire code on 3 videos and evaluate the results of our move detection. One of the key assumptions we make is

that the move detection is triggered manually by a user. Hence after every move a user must press a key for the detection to key.

Dataset	Viewpoint variation	Total moves	Correctly Identified
A	±22.5 deg	10	9
B	±50 deg	15	5
C	±5 deg	14	14

Table 3: Overall performance on 3 datasets

Note that we use the previous states as well to determine whether a move is valid or not. Hence if we detect any move incorrectly the future state would be a wrong one and our detection won't be able to find any more correct move.

Table 3 shows the result of our detection engine on 3 different videos. The viewpoint variation is just an approximate estimate of the actual movement.

5. Conclusion and Future Work

In this project we implemented an application that takes a video of a chess game and outputs the moves as output. We have implemented our program to be able to work with a moving camera. For dataset C which has only minor camera movements we are able to detect all moves correctly. However currently our algorithm works only if the view-point variation is small of the order of ±15°. Since it only relies on the amount of colour for detecting movements it is not accurate if the view-point variation is large. This is the reason we are able to detect only 5 correct moves on video B. Also if the camera is moved too fast, the Board detection algorithm fails due to motion blur.

To increase the accuracy it is possible to include additional features such as the shape of the piece or its silhouette. It could also be possible to detect the different pieces on the chessboard using bag of words approach with a dictionary of piece images. Such an implementation would also allow our detection engine to be able to start from a mid-game state. Hence a user may not need to record the entire game sequence and use only a smaller section of the video to obtain a subset of moves.

Another major limitation of our implementation is that we use manual intervention to decide when a move has been made. We hope to automate the process to find the frames before and after the move, probably by using hand detection. If this is done we can make this application more interesting by integrating it with augmented reality.This would allow two players to play online against each other with the opponents pieces being augmented.

The link to the video showing our code in action is `https://www.youtube.com/watch?v=XkeRsffWFec&t=2s`.

The link to our Github repository is `https://github.com/abhven/CV_chess`.

6. References

[1] E. Molla, V. Lepetit, Augmented reality for board games, in: Mixed and Augmented Reality (ISMAR), 2010 9th IEEE International Symposium on, IEEE, pp. 253–254.

[2] L.-H. Chen, C. Yu Jr, S.-C. Hsu, A remote chinese chess game using mobile phone augmented reality, in: Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology, ACM, pp. 284–287.

[3] M. Bikos, Y. Itoh, G. Klinker, K. Moustakas, An interactive augmented reality chess game using bare-hand pinch gestures, in: 2015 International Conference on Cyberworlds (CW), IEEE, pp. 355–358.

[4] F. Rayar, D. Boas, R. Patrizio, Art-chess: A tangible augmented reality chess on tabletop, in: Proceedings of the 2015 International Conference on Interactive Tabletops & Surfaces, ACM, pp. 229–233.

[5] Wireless arduino powered chess, in: https://www.youtube.com/watch?v=dX37LFv8jWY.

[6] Cvchess: Computer vision chess analytics, in: `http://web.stanford.edu/class/cs231a/prev\_projects/chess.pdf`.

[7] The chessnut library, in: `https://github.com/cgearhart/Chessnut`.

[8] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, M. J. Marín-Jiménez, Automatic generation and detection of highly reliable fiducial markers under occlusion, Pattern Recognition 47 (2014) 2280–2292.