

## I. Introduction

While the game of chess has greatly benefitted from the information age, including analytical algorithms, online playing environments and game databases, there have been relatively few advances that improve the experience of over-the-board chess. One such problem is the tediousness of recording the sequence of moves performed in a game.

We aim to improve this aspect of the game by creating a robust, automated system that leverages computer vision in order to provide insights into chess games played on physical boards. In particular, we would like to implement a set of CV algorithms that allow us to determine the full state of the board at all times. We will then provide users with recordings of their games that are easily portable to common chess databases and engines, so that a higher-level analysis might immediately take place. We believe that these algorithms could run real-time, such that two players could set up a simple camera rig (namely a macbook pro next to the board) with minimal restrictions on their location, then play a game with real-time insights provided.

This project is motivated by several established use-cases for similar systems, as well as problems with current systems that we can empathize with. Not only is the process of hand-recording moves tedious, but so is the process of typing moves into a computer program for analysis. Furthermore, real-time aids such as score-keeping, time-keeping, live analysis, and even automated coaching can be easily achieved with such a system. Given the prevalence of open research and tools for chess analysis, we suggest that our project will allow chess players the convenience and pleasure of playing chess on a physical board while maintaining the ability to leverage analytic advancements in computer chess.

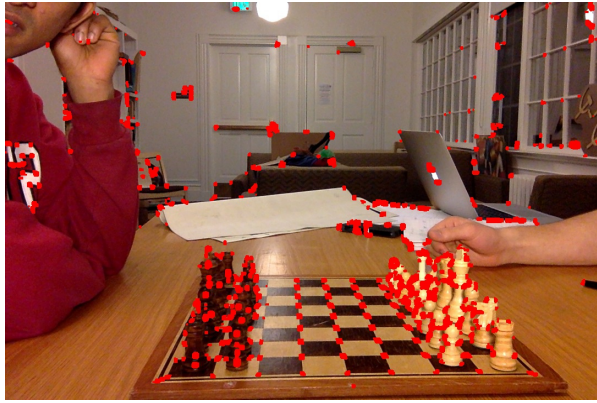
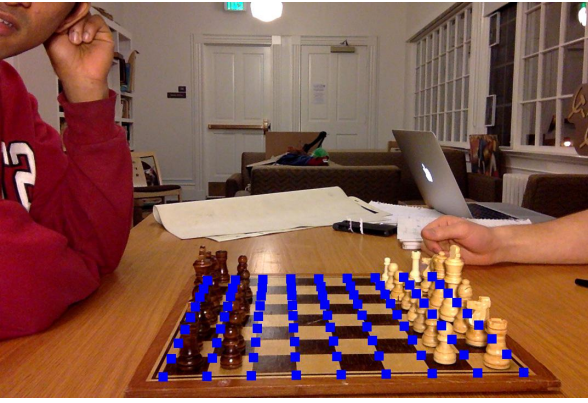
## II. Intended Approach and Technical Details

### Terminology

- Board coordinates:  $(2 \times 1)$  vectors that represent points on the chessboard;  $(0, 0)$  describes the top left corner of a8 and  $(8, 8)$  describes the bottom right corner of h1.
- Image coordinates:  $(2 \times 1)$  vectors that represent points in the image.
- Board-Image Homography: A  $(3 \times 3)$  projective matrix that maps homogenized board coordinates onto homogenized image coordinates.

The majority of existing research on computer vision applied to physical chess games either focuses solely on locating the board's gridlines (which many consider a 'solved' problem) or has been conducted by hobbyists and is not made available in the public domain. We have distilled this problem to three distinct modules: Detecting chessboard corners, finding a homography from board coordinates to image coordinates, then detecting whether a space is filled or not. An overview of our

approach and progress in each module is presented below.

	
<p><b>Figure 1:</b> Red points represent areas picked up by a Harris corner detector. We intend to train a logistic regression classifier on descriptors of each Harris corner (SIFT, location) in order to get higher precision in corner recognition.</p>	<p><b>Figure 2:</b> Given 5 (hand-marked) point correspondences between board coordinates and image coordinates, we computed the projective transformation relating them; this image displays the predicted location of all board corners.</p>

## A. Automatically Finding Point Correspondences

In order to find the plane of the board and subsequently analyze the content of board squares, we must first find a set of point correspondences between board coordinates and image coordinates. We intend to do this with the following algorithm: (1) find corner candidates with harris corner detection, (2) filter out clear false positives with a logistic regression classifier applied to descriptors of the Harris corners (SIFT, location), then (3) find the most likely set of remaining corner candidates comprising the center four rows of the chessboard using a moving parts model. The details of each step are described below:

### 1 Harris Corner Detection

The results of a Harris corner detected to one of our preliminary images is shown in (fig. 1). Clearly, this has a high recall in finding chessboard corners, as every corner in the center four rows is saliently marked. This provides a good population of candidates for more complicated classification procedures.

### 2 Filtering via LR on SIFT Descriptors

Intuitively, chessboard corners are very distinctive in their appearance. We therefore reason that SIFT descriptors of detected Harris corners, along with a function of their location (in order to capture the fact that chessboards are likely to reside near the bottom of the screen) will serve as good features for a logistic regression classifier trained to discern true chessboard corners from false positives. Furthermore, the output of this classifier, the probability that a given Harris corner is a true board corner given it's surrounding pixels and location, provides a convenient factor to use in

the deformable parts model PGM described below in section (3).

### 3 Finding Correspondences via Deformable Parts Model

Once we have obtained a reduced population of candidates for chessboard corners, each with a confidence score obtained from the logistic regression classifier described above in (2), we intend to find the set of points that comprise the vertices of the central four rows of chessboard squares using a deformable parts model. In particular, we will construct and parametrize a bayesian network as follows:

- Step 1: find all subsets of remaining Harris corners of size N, where N is the number of square corners in the central four rows of the chessboard
- Step 2: for each set, put them in a matrix M such that  $M_{ij}[x] < M_{kl}[x]$  if and only if  $j < l$ , and similarly for the y coordinate and i, k.
- Step 3: for each set, find the full joint probability that all are in fact square corners, as follows:

$$P(\text{all square corners} \mid \text{locations, descriptors}) = \prod_i P(p_i = \text{square corner} \mid p_i \text{ location, descriptor}) \\ * \prod_{i, j \text{ neighbors}} P_{ij}(p_i, p_j = \text{square corners} \mid \text{relative locations})$$

Here,  $P_{ij}$  is a distribution for each pair of neighboring elements in the matrix over their relative locations - that is, it will capture the fact that near the bottom of the board, two neighboring corner points are likely to be further away from each other than those at the top of the board.

- Step 4: take the set of points that maximizes this full joint probability and return them, along with their corresponding board coordinates, as point correspondences for the algorithm described in part (B).

### B. Finding the Board-Image Homography

Only a subset of board corners are detectable with corner detectors due to occlusion. (i.e. the top left corner isn't even visible) We are guaranteed, however, that the chessboard consists of an evenly-spaced grid lying on a plane; there is therefore a projective transformation relating points on the board (in board coordinates) with points in the image, which can be computed from a small set of known point correspondences between the image and board. This homography allows one to find the image coordinates of any point on the board, therefore enabling one to select image regions corresponding to certain squares. Here we outline our approach to finding this homography.

let  $P_i, P'_i = \text{corresponding points in board/image coordinates, respectively, for } i = 1 \dots n$   
 $\Rightarrow \exists H \in R^{3 \times 3} \text{ s.t. } P'_i = HP_i$

$H$  can be determined by the following overdetermined system of equations :

$$Ph = 0, \text{ where } P =$$

$$\begin{bmatrix} p_{1x} & p_{1y} & 1 & 0 & 0 & 0 & -u_1 p_{1x} & -u_1 p_{1y} & -u_1 \\ 0 & 0 & 0 & p_{1x} & p_{1y} & 1 & -v_1 p_{1x} & -v_1 p_{1y} & -v_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ p_{nx} & p_{ny} & 1 & 0 & 0 & 0 & -u_n p_{nx} & -u_n p_{ny} & -u_n \\ 0 & 0 & 0 & p_{nx} & p_{ny} & 1 & -v_n p_{nx} & -v_n p_{ny} & -v_n \end{bmatrix}$$

and  $h$  is a columnized representation of  $H$ .

In order to solve this system of equations, we apply SVD to  $P$  and construct a matrix from the last row of the resulting matrix  $V$ , as discussed in lecture. Our implementation of this algorithm can be found in the file `CVAnalyzer.py` on our github repository; (fig. 2) provides a visualization of this algorithm applied to five hand-marked point correspondences.

### C. Detecting Chess Piece Locations

Once we have the plane of the board and the image locations of every vertex, as outlined in the steps above, we will be capable of extracting the image regions that correspond to every given square. We intend to focus our efforts on detecting the *presence* of a pieces within squares and, subsequently, use the constraints on piece movement, as well as the initial board configuration, to infer the identities of pieces in each space. We are currently considering two alternative methods for piece detection and plan to experiment with them both (potentially in tandem) throughout our development process. They are as follows:

#### 1 Detection via Square Color Histogram

It seems intuitive that a histogram of the pixel color values within a square should be fairly indicative of the square's content. This is because an empty square will likely be of uniform color, while an occupied one will have two large, distinct constituent color components. Therefore, we will experiment with training a predictor on some function of the square's pixel color (or intensity) histogram in order to predict square occupation.

#### 2 Detection via Square Color Gradients

One issue with the above is that various lighting conditions can cause an empty board square to have varying color throughout. (i.e. the when a particularly bright light reflects off of a shimmery chessboard surface.) Our intuition is that one way to differentiate between color changes from light and color changes from chess piece occlusion is to look at the gradient of pixels within the square, or perhaps the number of 'edge' pixels within it. While lights are likely to introduce smooth and slow gradients of color across the square, a piece will have a sharply defined boundary - therefore, it could be a very informative feature in predicting square occupation.

## III. Milestones

(1) Given correspondences for a few board points and image points, generate the positions of all 64 squares and 81 vertices on a chessboard, independent of the angle. (Complete)

- (2) Recognize the visible vertices on a chessboard with pieces on the image. (Complete)
- (3) Achieve automatic generation of board point to image point correspondences. (In progress)
- (4) Given an image region corresponding to a board square, correctly predict its occupation with high accuracy. (2/28)
- (5) Recognize when a move is played in a video of a chess game. (2/28)
- (6) Automate all of the above processes for live games of chess. (3/1)
- (7) Integrate program output with existing chess engines. (3/10)

## **IV. Resources**

<http://martincmartin.files.wordpress.com/2006/06/finding-a-chessboard.ppt>

<http://codebazaar.blogspot.com/2011/08/chess-board-recognition-project-part-1.html>

[http://www.comp.nus.edu.sg/~cs4243/showcase/chess\\_vision/chess\\_vision.html](http://www.comp.nus.edu.sg/~cs4243/showcase/chess_vision/chess_vision.html)

<http://arxiv.org/abs/1301.5491>

<http://www.curiousattemptbunny.com/2011/07/chess-board-computer-vision.html>

[sdiwc.net/digital-library/web-admin/upload-pdf/00000296.pdf](http://sdiwc.net/digital-library/web-admin/upload-pdf/00000296.pdf)

[http://www.hallway.us/~tyson/downloads/mwscas\\_2010.pdf](http://www.hallway.us/~tyson/downloads/mwscas_2010.pdf)