Game Record

Steven Scher and Ryan Crabb

UCSC CMPS 290b, Fall 2007

ABSTRACT

We integrate in-person and on-line playing of board games such as Go. We allow a player to record an in-person game by placing their camera or cellphone on the table next to the game board. We record photos of the game and automatically transcribe the game. We automatically detect the board and playing pieces, using the game’s rules to accurately estimate long sequences of moves. We achieve excellent accuracy, with 9 out of every 10 games recorded without error. The game transcript is automatically uploaded in the standard SGF file format, and may be studied afterwards, shared with friends or coaches, or added to online compilations.

SECTION 1: INTRODUCTION

Many board games, such as Chess and Go, are played both in-person and online. After playing a game online, the record of the game is available so that the players may discuss particularly good and bad moves with each other and with friends and teachers. This commentary often adds significantly to the social experience and skill level improvement. Reviewing a game repeatedly until every move has been understood and memorized is a common study technique. Very good players can remember a game played in person, and review them afterwards. Other players don’t have as keen memory, but they do have a cellphone.

We allow a player to record their game simply by placing a cellphone camera on the table next to the board during a game. Our software on the phone automatically take pictures, finds the board and detect moves. A complete record of moves played is created, so that good and bad moves can be reexamined by those present and shared via email with others. A file in the standard SGF file format may be uploaded to the player’s online account, and online game databases can be queried to find professional games facing similar decisions. No other method has yet been devised to automatically record an entire game. Rather, in practice in order to transcribe a game, a written record is made as it is played.

Whether detecting a set of lines that constitute a go board, or a series of stone placements that constitute a go game the arrangement of the objects is highly specified. We first detect the individual objects (lines or stones) and then perform inference to identify their arrangement in space or time, recovering from both false positives and false negatives in detection. The inference methods constrain the arrangements to be consistent with its known properties; the board forms a regular grid, and stones alternative black and white.
When detecting the individual stones, or when detecting the lines on the board, a four-step process is used. First, an estimate of the orientation of the expected edges is made. Second, edges at this orientation, and of the correct order (e.g. step-edges as opposed to lines) are found. Third, a hough transform is used to approximately localize the object. Fourth, a maximum-likelihood estimate of the object’s position is made.

Furthermore, we perform inference over our detection probabilities to leverage our knowledge of the game’s rules, such as alternation of black and white stones. We achieve excellent error rates, with false negatives (failing to detect a stone present) only once in 10 games of 200 moves each, and false positives (erroneously detecting a stone) only once in 10 games. We contribute a merging of state of the art computer vision techniques to a novel application area, and a demonstration that such traditional techniques are robust and reliable.

Our contribution is a novel application of

SECTION 2: RELATED WORK

The task of recording go games requires four major elements. First, the board must be automatically identified. Second, each possible location of a stone must be classified as a black stone, white stone, or an empty intersection. Third, occlusion of the board by players’ hands must be accounted for when detecting stones. Fourth, the time series of the game must be coherently estimated. We now discuss prior work in each of these areas.

The go board is a 19x19 rectangular grid of black lines on a wooden board. While automatically detecting a chessboard pattern of black and white squares is now so reliable that it is the standard method of calibrating a camera, detecting a grid of thin lines has not yet reached this status. Detecting the grid may begin by detecting lines, for example with the Hough transform [10]. The Hough accumulator representation of the detected lines may be directly used to detect the grid, as in [1] and [9]. This method does not extend to the extreme perspective distortion present when the camera is placed next to the board at eye-level. The perspective distortion may be automatically corrected by identifying the two sets of perpendicular lines in the image and finding their vanishing points, as applied in [7] to images of city streets. We extend this method by also finding the grid, and optimizing the correction for our known grid size. [NOTE TO SELF: WE DON’T ACTUALLY USE THAT METHOD ANY MORE] An alternative method is applied in [1], where a set of SIFT features is detected and classified by a Support Vector Machine as corners, side intersections, interior intersections, or outliers. The side and corner intersections are grouped into four lines whose intersections define the board. This method reports 94% success on 9x9 boards with the camera close to overhead; it is unknown to the authors whether the system extends to 19x19 boards with the camera at an angle. Some systems omit automatic grid detection, requiring the user to manually specifying the grid corners [4, 5].

Classification of image patches into a small set of possible objects is a well studied problem [11]. In comparison to the more difficult task of finding e.g. faces or cars, detecting the stones against a known background represents a relatively easy classification problem: they are either white or black, and their appearance in the image is dominated by an ellipse. However, the elliptical approximation is
complicated by strong specular highlights, shadows, and occlusion by nearby stones [See figure XXX]. A cascade of three classifiers is applied in [1]. First, the a hough circle detector is used; ambiguous cases are resolved by an SVM operating on a SIFT descriptor; remaining ambiguities are resolved by a “Last Ditch” SVM operating on the mean and standard deviation of pixel brightnesses. The cascade achieves perfect accuracy on a partially filled 8x8 board (64 possible stone positions) in 73% of 39 test images. This is insufficiently reliable for an automatic whole-game recording system. Several simple heuristics may be effective for some cases. The neighborhood brightness is thresholded in [2] and [5]; success statistics are not available.

A significant challenge in stone detection is the frequent presence of hands obscuring large parts of the board. The hand creates many depth-edges with the board and makes simple background-subtraction and edge-detection schemes unreliable. The hands must be detected and ignored in order to confidently classify the presence or absence of stones. If the photos are taken frequently enough, optical flow may be used to identify parts of the scene containing motion, as in [4]. However, the necessity of stopping one’s hand when placing a stone make this insufficient. The low light levels in typical game playing settings may also require significant shutter times, limiting frame rate. We use a low frame rate of 1 frame/second, and maintain a background model of the image in order to perform background subtraction, as well as using a skin-color detector.

A recording system must combine the classifications of many images of a video stream to coherently estimate the sequence of moves. Enforcing these game’s rules here is beneficial. The game of go has simple rules regarding stone placement. One stone is played at a time, with black and white stones alternating. Once played, a stone is not moved unless surrounded by opposing stones, when it is removed from the board. A relaxation of the rules were applied in [4], where each stone was assumed to remain in place unconditionally once played. The time-series of classifications of a single intersection was considered in separately from other intersections, and the most likely placement of the stone was found using a Hidden Markov Model. This method does not allow for the full rules of the game, where a stone is removed under deterministic circumstances; we allow this. We also enforce the addition of only one stone at a time to the board, and alternation of black and white stones.

An alternative to recording games by camera is to instrument the board itself with physical detectors, as performed by Alex [reference] for the game of go, and by the commercial product (???) for chess. This paper addresses the more universally-applicable use of a camera with a standard, uninstrumented board.

SECTION 3: THE GAME OF GO

The typical scene we consider is shown in Figure YY below. The go board itself is a 19x19 rectangular grid of black lines drawn on a rectangular wooden board. Two players, white and black, alternate turns placing a stone, one at a time, on an empty intersection. Once placed, the stone is never moved, with one exception: capture.

Two same-colored stones are considered connected if they are adjacent, that is, if they are connected by lines on the board (vertically or horizontally, not diagonally). A group of stones is a connected
component, as in figure YY. If no stone of a group of is adjacent to an empty intersection, having been surrounded by the other player’s stones, the entire surrounded group of stones is immediately removed from the board.

[NOTE TO REVIEWERS: I didn’t know where to put this so I gave it a separate section. Does this work? Any better idea of where it should go? --Steve]

SECTION 4: METHODS

This section describes the algorithms used to transcribe the game. The method used to deal with the presence of occluding hands is discussed first in section 4.1. The procedure used to detect lines on the board is discussed in section 4.2, followed by the RANSAC procedure used to reason about the geometric arrangement of detected lines in section 4.3. Finally, the procedure used to detect the stones is discussed in section 4.4, followed by the inference method used to reason about the time-series of detected stones in section 4.5.

We pause to note the similarity in detection of the board and the stones. The procedure for detecting the individual object (line or stone) is nearly identical, while the inference methods are different but similarly motivated. The inference methods are greedy approximations to an exploration of the full space of possible configurations of the objects.
SECTION 4.1: IGNORING HANDS

Note that a player’s hand will occlude some part of the board in most images, usually near where a new stone is being placed, the place we are most interested. Before looking for stones, we use smart algorithms to ignore the players’ hands. A naïve algorithm might form the image sequence into a spatio-temporal volume, and apply a moving-window median filter in the time-dimension. We use some cool thing that does better than that. [I need to ask Ryan what this is —Steve].

SECTION 4.2: DETECTING LINES ON THE BOARD

The board is found automatically by first detecting edges, then lines. For maximal robustness, the algorithm begins by finding edges at the center of the image with a radially-symmetric laplacian ("tophat") filter to detect edges at all orientations. A hough transform line detector finds lines, and these lines are clustered into 2 groups according to their orientations using K-means. The dominant orientation of each cluster of lines is then estimated.

The whole image is now considered. For each of the two dominant line orientations, separately, edges at that orientation are identified with an oriented 1D laplacian filter. Lines at the desired orientation are then found using a hough transform. Each line found is projected onto the image, edges near the line are identified, and a maximum-likelihood estimate of each line’s position is made.
SECTION 4.3: INFERING THE BOARD FROM LINES

Since some lines may be missed and some spurious lines may be found, RANSAC [reference?] is applied to find a confidently-identified subgrid of the whole grid. Two lines of each orientation are randomly chosen (biasing the selection to favor lines far from each other) to form a rectangle that defines a homography. For each of the 2 orientations, separately, several guesses are made at the number of gridlines between the two chosen lines. For each guess, the implied gridlines are projected onto the image, and the guess is scored by how well these lines match the detected edges.

The best-scoring set of six items (for both orientations, two lines and an estimate of the number of gridlines between them) is considered to be a subgrid of the full 19x19 grid. The grid is greedily grown until reaching the full 19x19 size. At each step, a grid larger by 1 line in each of the 4 possible direction is hypothesized, projected onto the image, and scored, and the best of the 4 is accepted.

SECTION 4.4: DETECTING STONES

At this point, a synthetic video of the game has been created with hands ignored, and where the 4 corners of the board have been found. The homography defined by the board specifies the 361 locations in the image where a stone might appear, and further specifies the size and eccentricity of each expected ellipse. So armed, we apply the stone detector described below to each expected location.

Though the expected center of the stone is known, the stone may be placed off-center. We assume that the stone will be placed sufficiently close to the center that it covers the center. That is, we assume the offset is less than half the width of a stone. Under this assumption, all gradients formed by the edge of the stone will have orientations approximately matching the direction pointing outward from the expected center. We find gradient magnitudes only at these orientations, ignoring spurious edges from nearby stones.
We threshold the gradient magnitude to create a sparse set of detected edges. A hough transform ellipse detector is applied to the detected edges. The detected edge pixels supporting this ellipse are identified, and their gradient magnitudes summed to give a score that we interpret as a probability. The color of the area within the ellipse is thresholded to determine the possible stone’s color.

[NOTE TO REVIEWERS: Is any figure needed for the detection section?]

SECTION 4.5: INFERING THE SEQUENCE OF MOVES FROM DETECTED STONES

Three layers of the algorithm have now been described. Hands are ignored, the board is found, and in every image, each possible stone location is classified as empty or occupied. The classification accuracy is only XXX%, so that on average, the average length of play expected to be perfectly transcribed without error is [half?] of a single game. If the players are known to be playing a game of go, then the rules of the game greatly limit the sequence of legal plays. Enforcing the rules of the game allows many games to be played with no transcription errors. We enforce the rules by performing a depth-limited-search of the tree of all legal move sequences, in order to consider the evidence over a large time period when choosing the first move. The first move is greedily chosen, and a new depth-limited search is conducted to choose the next move.

This algorithm enforces the placement of only one stone at a time on the board, alternation of the placement of black and white stones, and capture (removal from the board) of stones when surrounded by the opponent’s stones. This comprises the full rules of go, except for ko, a rule which requires that a move may not recreate any board state previously seen in the game. Though ignoring the ko rule would be disastrous to a player’s strategy, it does not significantly limit transcription of games from video, and could be easily added to this technique if desired.  [NOTE TO REVIEWERS: is mentioning ko necessary?]  

A simple way to enforce these rules is to enumerate all legal move sequences, and choose the sequence that best matches the predictions of the classifier. This may be done by creating a tree where each node represents a configuration of stones on the board (a 19x19 array whose elements are empty, black, or white), and the children of a node are all boards resulting from legal moves from the parent board, plus a copy of the parent node. The root node of the tree represents the empty board; the first layer of the tree consists of any board with one black stone, or the empty board; the second layer consists of every board with one black stone and one white stone, or one black stone, or one white stone, or the empty board.

A node at a particular depth D represents the state of the board in the D\textsuperscript{th} time frame of the video recorded of the game. A path through the tree from the root to some particular depth D represents the state of the board at each of the first D time frames of the video. At every time frame of the video, either a legal move is made, or (most of the time) the board does not change. At most points in the game, the only legal moves are the addition of a stone by the appropriate player (white or black). However, when a move captures some of the opponents stones by surrounding them, then the only legal moves are to remove one of the captured stones, until they have all been removed.
The likelihood of a hypothesized state of the board at some particular depth \( D \) may be evaluated by multiplying the likelihoods that each empty intersection is truly empty and each predicted stone is truly empty. This likelihood may be obtained from the classifier described in the previous section. The likelihood of an entire sequence of \( D \) moves (a path from the root to a node on the \( D \)th layer) may then be evaluated by multiplying the likelihoods of every state of the board along that path.

Enumerating the full tree to depth \( D_{\text{max}} \) would require an enormous amount of memory to construct, order(\( 2^{D_{\text{max}}} \)). The tree may be searched implicitly in a depth-first-search with little memory required, only order(\( D_{\text{max}} \)). The time to search the entire tree would nevertheless be prohibitive, order (\( 2^{D_{\text{max}}} \)), but large portions of the tree may be pruned from the search, which tremendously decreases the search time.

Three search optimizations are made. Firstly, backtracking is used to prune subtrees known to be less likely than the current most-likely path. Second, searching children of a node in priority order, as in a best-first-search, increases the amount of the tree that is pruned from search by backtracking. Third, the branching factor may be limited to match the observed probability distribution of errors in the classifier described in the previous section. In practice, these three optimizations limit our search to a time complexity of order (\( ??? \)).

Since the likelihood of every node in the tree is in (0,1), and the likelihood of a path is the product of the likelihood of its constituent nodes, the likelihood of any path through the tree is lower that the likelihood of any subpath (a path containing only the first \( K \) nodes of the original path). Thus, when conducting a search of the tree, if the path to a node \( N \) is less likely than the current most-likely path, then the path continuing to every node in the subtree rooted at \( N \) is also less likely than the current most-likely path, and the entire subtree may be pruned from the search without changing the result of the search. This pruning is known as “backtracking.”

Since subtrees are pruned when a path is found to have a lower likelihood than the current best most-likely path, it is fruitful to find high-likelihood paths early in the search, in order to prune large amounts of the tree. This may be accomplished by performing some extra work when choosing which child of a node to consider next. Each child may be evaluated, and the search may continue along each child in descending order of likelihood. The cost of performing these extra checks is (\( ??? \)) while the time saved through increased backtracking is (\( ??? \)).

The classifier is quite accurate in practice, and can be reasonably expected to have no more than a few, e.g. ten (?), false positives or false negatives in any given time frame. Thus, only a few locations will be classified differently in adjacent time frames. Thus, the branching may be limited to consider only some maximum number of branches from any node.

Although this search is quite fast in comparison to a search of a full tree, searching to a depth equal to the length of the video is prohibitive. We achieve most of the benefit of the full-depth tree search by using a modification of depth-limited search with iterative deepening. A depth-limited search (with the depth limit \( D_{\text{max}} \) equal to 30 seconds of play) is first conducted with a root node as the empty board, corresponding to the first frame of video. The most likely path up to depth \( D_{\text{max}} \) is found, and the first
node of this path $N$ is greedily accepted true. All subtrees not containing this node $N$ are pruned, and the search is deepened one additional ply, $(D_{\text{max}} + 1)$. Since there is now only one child $N$ of the root node, this is equivalent to performing a depth-limited search of depth limit $D_{\text{max}}$ rooted at the node $N$. This process is repeated to choose each node as first node in the most likely path from the previous node to a depth limit $D_{\text{max}}$.

With many object detection algorithms, a trade off must be made between minimizing two different types of mistakes: false positives (mistakenly reporting nonexistent stones) and false negatives (mistakenly failing to report true stones). Often, the error rate in one type of mistake can be reduced by allowing the other to rise. The tree search copes well with large number of false positives, provided that the rate of false negatives is reasonably low.

[NOTE TO REVIEWERS: the tree-search section seems long. What should I cut? What parts should be explained with figures?]

RESULTS

We are very accurate and robust. This will be useful to people in practice. Many games can be accurately recorded without error. We used our algorithm on a test set with excellent results.
Figure Caption: Our method accurately creates a transcription of a Go game (top row) from a video of the game (bottom row). Inference over possible legal move sequences allows mistakes by the classifier (middle row) to be ignored in the final result.

Table Caption: Accuracy in Detecting Board

<table>
<thead>
<tr>
<th>Method</th>
<th>Expected # Games without Manual Intervention</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detecting subgrid with</td>
<td>10</td>
</tr>
<tr>
<td>Method</td>
<td>Expected # False Positives Per Game</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>Our Full Method</td>
<td>Small</td>
</tr>
<tr>
<td>Our Raw Detector</td>
<td>Not as Small</td>
</tr>
<tr>
<td>Other Method?</td>
<td>Large</td>
</tr>
</tbody>
</table>

**DISCUSSION**

Providing the benefits of online game play to in-person play enhances both game play environments. Automating transcription of games from an image sequence is useful to individual players who want to review their games to improve their skills, as well as to tournament organizers who want to record the result of many games for display to the public.

This project is important, difficult, and creatively solved. You are very impressed and want to hire us.

**REFERENCES**

(Recording a Go Game)


[4] Recording a Game of Go: Hidden Markov Model Improves a Weak Classifier, Steven Scher and Manfred Warmuth, University of California Santa Cruz (My project from last year)


[8] A New Sub-Pixel Detector for Grid Target Points in Camera Calibration, Chen D., Wang Y. and Zhang G., Beijing University of Aeronautics and Astronautics

[9] Houghing the Hough: Peak Collection for Detection of Corners, Junctions, and Line Intersections, Barrett and Petersen, Brigham Young University (Detecting Stones)

[10] Hough Transform