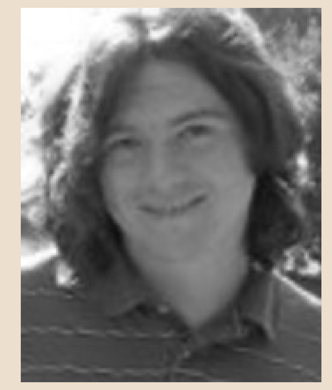
Making Real Games Virtual: Tracking Board Game Pieces

Steven Scher
sscher@soe.ucsc.edu
www.soe.ucsc.edu/~sscher



Ryan Crabb
rcrabb@soe.ucsc.edu

James Davis
davis@soe.ucsc.edu

Department of Computer Science, University of California, Santa Cruz, CA

Introduction

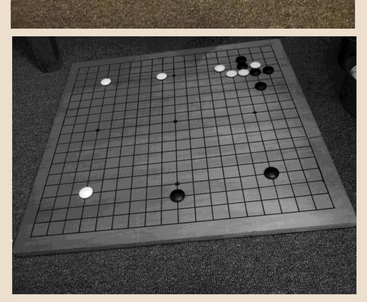
Many board games are played in person and online. We bring the real world into the virtual world.

A player may record an in-person game by placing their camera on the table next to the game board, taking photos of the game. We automatically find the likeliest legal move sequence

The game transcript may be studied afterwards, shared with friends and teachers, or added to online compilations, bringing the attendant benefits of online game play to an in-person game.

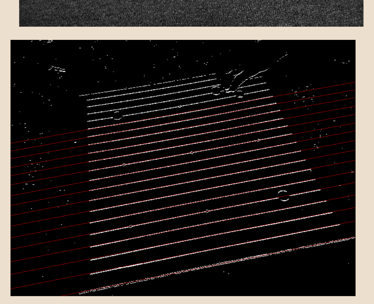
Detecting the Game Board

Original Photo



Foreground Removal

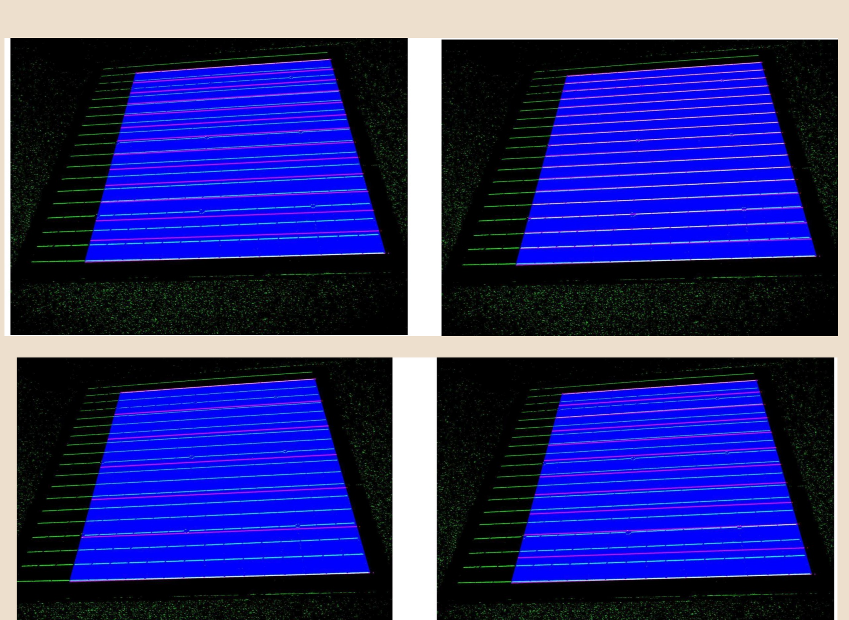
Hands and other moving objects create many false positive and false negative detections. A per-pixel median-filter removes most moving foreground objects. We compute the color analog of the median, minimizing L1-norm in HSV color space.



Line Detection:

Lines are initially detected with a circular Laplacian filter and hough transform, and the two dominant orientations are estimated with K-means.

Lines at each orientation are then found separately with an oriented 2nd-order Derivative-of-Gausian filter and Hough Transform.



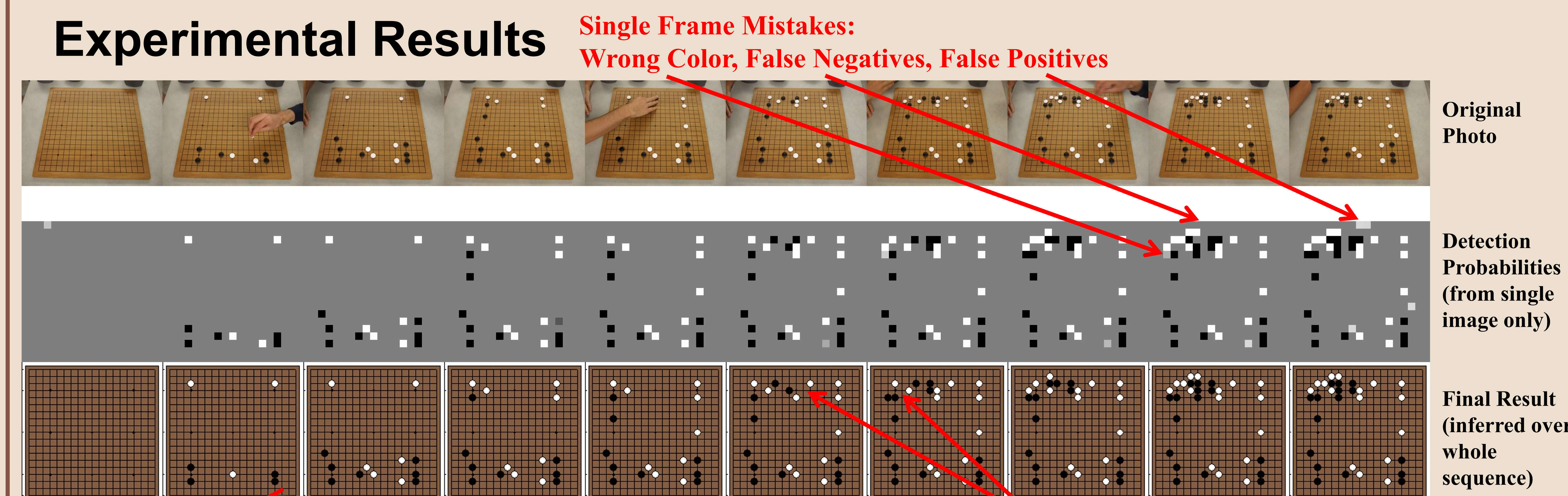
Finding a Rectangular Grid

We apply RANSAC to reliably find a rectangle among the detected lines. Two lines and a guess at the number of gridlines between them are selected from each orientation. The edges at each orientation predicted by this hypothesis are compared against the detected edges to score the hypothesis.

The highest scoring rectangle is iteratively grown in the best of the four possible directions to best match detected edges.

Once found, the board is rectified and cropped.

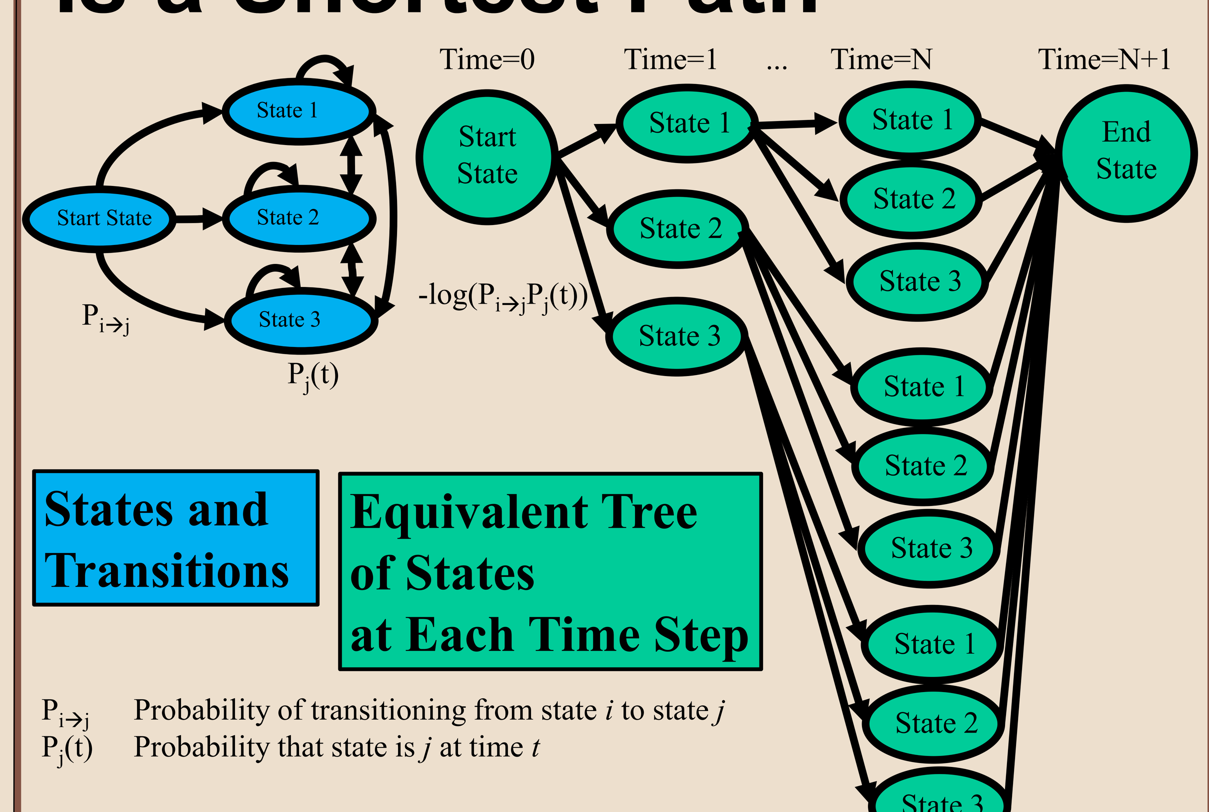
Experimental Results



Game	Mistakes in Simple classifier	Mistakes after A* correction	Number of moves in game
1	65	0	31
2	414	0	70
3	110	14	115
4	61	11	49

These images represent a time sequence of moves played. The top row shows the original image of the board. The middle row shows the likelihood of detected stones. This is found with a Support Vector Machine operating on each image separately, and may make similar mistakes at many time steps, possibly alternating between correct and incorrect detections. The bottom row shows the best move sequence decided by the inference algorithm. Inference qualitatively changes the kinds of errors incurred: many fewer mistakes occur, with the most common mistake being stones detected in the wrong order.

The Most Likely Path in a Hidden Markov Model is a Shortest Path



$P_{i\to j}$ Probability of transitioning from state *i* to state *j*
$P_j(t)$ Probability that state is *j* at time *t*

Constructing a Tree from a Graph

The graph of states and possible transitions is expanded into a tree with N discrete steps, including transition from a state to itself. All nodes at time step N transition to a new terminal node.

The likelihood of sequence S in the original graph is $\Pi_{t=1\ldots N} P_{S(t)}(t)P_{S(t-1)\to S(t)}$

The length of a path from the start to end node in the equivalent tree is:

$\Sigma_{t=1\ldots N} -\log\,[P_{S(t-1)\to S(t)}\, P_{S(t)}(t)]$, so that the shortest path in the new graph is the likeliest sequence. In the original graph

Finding The Most Likely Move Sequence In an Exponentially Large HMM Efficiently with A*

Constructing a Tree

For a Game of Go

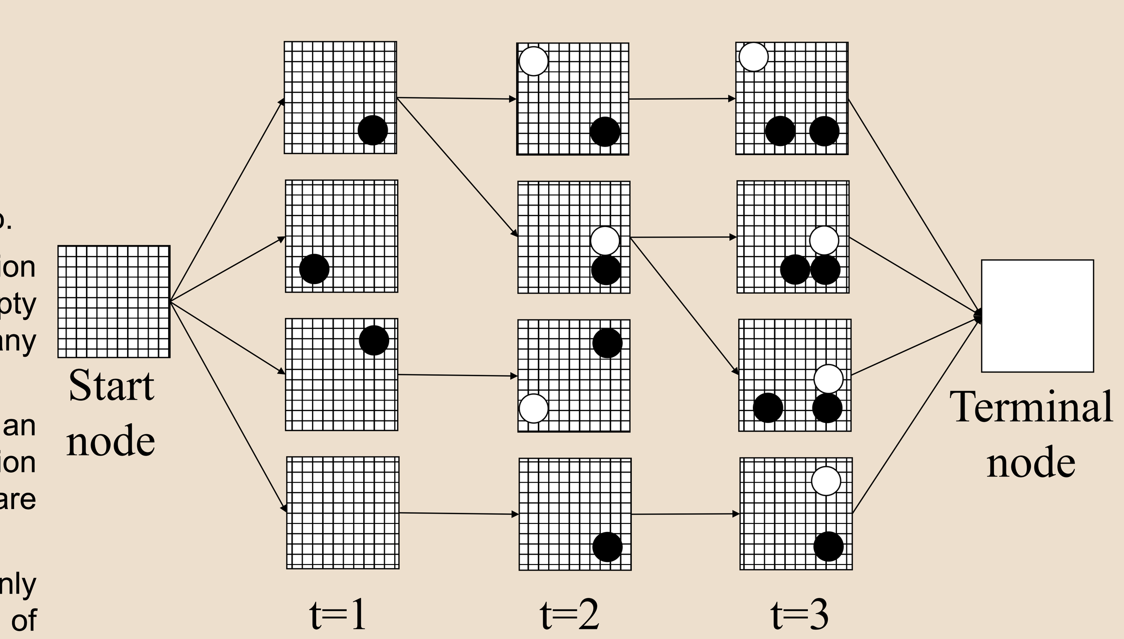
The Start node is the empty board.

Each time step corresponds to a photo.

Possible nodes (nonzero transition probability) at time step 1 are the empty board, or a single black stone in any position.

Possible nodes at any time step are an unchanged board, or the legal addition of a single stone. All legal transitions are equally likely.

Following stone capture, the only possible transitions are removal of possible stones.



Calculating $P_j(t)$

The Go board is a 19x19 grid of 361 possible stone locations. Each location is either empty, or occupied by a black stone or white stone. A Support Vector Machine is trained to estimate probabilities for the three possible values, and is applied independently to each pixel in each photo.

The board state j of the whole board assigns a value (black, white, empty) to each of the 361 locations. The likelihood of a state is the product of the likelihood of each assignment.

A Good Heuristic for A*

All graphs that calculate $P_j(t)$ as we do admit a very good heuristic for A*. The state that assigns the most-likely value to each location (independently and without regard to history or game rules) gives an upper bound on $P_j(t)$. In practice this bound is quite tight, since the stone detector typically makes no more than a few mistakes in any single photo.

A tight bound allows A* to trim large portions of the search tree. Since the tree is constructed on the fly, and only those nodes in the fringe are kept in memory, a tree with $o(10^{20})$ nodes may be searched with only $o(10^4)$ nodes actually evaluated, and only $o(10^3)$ in memory at once.