Connect-451: A Real Time Connect 4 Board Processor

Amos Cao
EECS 451
University of Michigan
Ann Arbor, Michigan 48105
Email: amoscao@umich.edu

I. Introduction

This project outlines a method to capture the state of a "Connect-4" board game in realtime using a webcam and basic image segmentation methods. "Connect-4" is two player perfect information game, which was strongly solved in 1988 [1]. In this implementation, a human player will always play first as red. The program will then acquired board state using a webcam, and then calculate a "good" response for the yellow player. This is repeated until one player achieves victory by having four pieces in a row, either horizontally, vertically, or diagonally.

II. Image Acquisition

Images are acquired using a Logitech c270 webcam, which uses a USB interface to connect to a computer. MATLAB is then used to pull single frames from the webcam on demand. The image is then **downsampled to 640x480, a resolution that is selected at the image acquisition step. This is the first of our in-class DSP tools.** Because this program only needs to detect large featureless disks, downsampling decreases processing times while minimally affecting accuracy. Image preprocessing and segmentation steps are then performed on the acquired frame, in order to make a game decision. Figure 1 shows an example of a raw acquired frame.

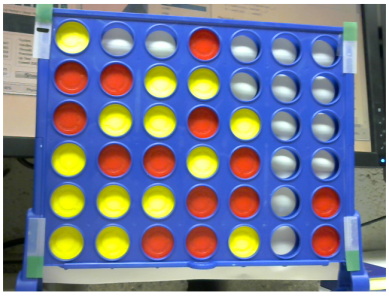


Fig. 1. 640x480 RGB Image Acquired Using MATLAB

The image in Figure 1 was acquired with the board intentionally tilted to the right. Because of this perspective, the rectangular shape of the board is warped, resulting in non-uniform piece holes.

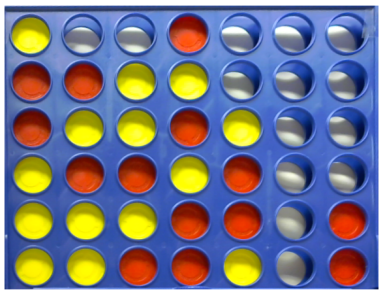


Fig. 2. Transformed RGB Board Image

III. Image Preprocessing

A. Corner Tracking

Undoing the effects of perspective will make segmentation easier by ensuring more uniform board piece size. This can be accomplished through an image transform, applied with an implementation by Michael Chan found on Matlab Central [2]. As we are only interested in the board surface, setting the input transformation coordinates to the corners of the board will also crop the image to the board, as well as transform it. In order to detect the corners of the board, green markers made of paper were taped to the board corners (Fig. 1).

An algorithm selects the four largest green areas from the image, removing noise using a **3x3 median filter, the second of our in-class DSP tools**. Centroids of each of those objects are then calculated using *regionprops*, part of the MATLAB Image Processing Toolbox. These centroids correspond with the corners of the board, and are then used as the input to the Chan *Perspective Control* function to projective transform the image into a rectangle. Figure 2 shows the image transformed board. As long as the camera's field of view is kept unobstructed, the board can be rapidly and accurately transformed, even when the camera is in motion or at an angle.

To test the range angles this system was functional at, the camera was horizontally and vertically rotated around board until the transform system stopped working. In the horizontal case, the image (Fig. 3) was able to be transformed up to an angle of $\pm 45^\circ$ (Fig. 4). The vertical transformation performed better (Fig. 5), with the transforms still occuring up to $+65^\circ$ (Fig. 6).

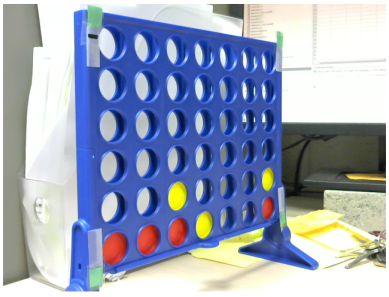


Fig. 3. Horizontal 45° Rotation

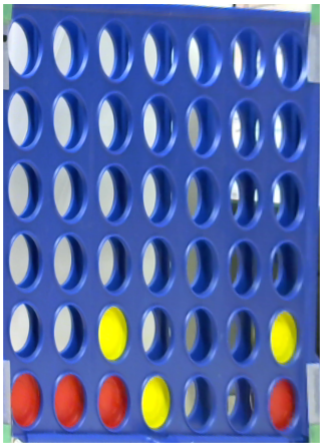


Fig. 4. Horizontal 45° Transformation

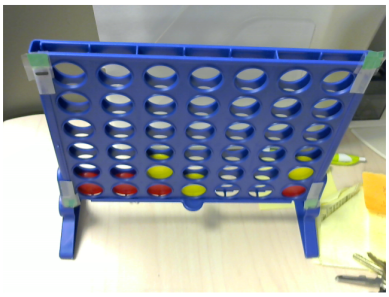


Fig. 5. Vertical 65° Rotation

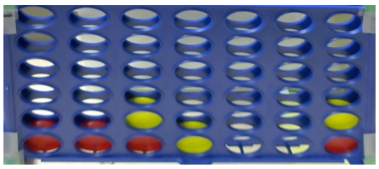


Fig. 6. Vertical 65° Transformation

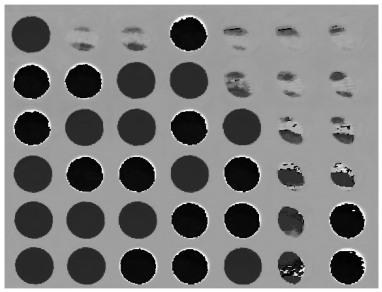


Fig. 7. Hue Channel after HSV Conversion

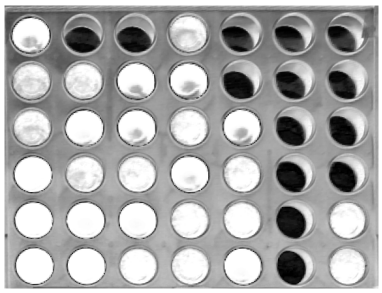


Fig. 8. Saturation Channel after HSV Conversion

B. Corner Tracking Issues

The Logitech camera used employs some sort of auto-brightness correction, which occurs before the image acquisition step in MATLAB. Because of this, hues can become distorted under different lighting conditions. The thresholds for detecting the green corners are set to a very narrow range, in order to exclude pixels which do not belong to the markers. This often results in the markers not being detected at all under certain lighting conditions. To improve on this method, a pattern based tracking method could be employed, similar to the corners of a QR code which have alternating white and black concentric squares.

C. HSV Transformation

Next, the RGB color model of the acquired image is non-ideal for detecting and segmenting colored pieces. **The image is transformed into the hue-saturation-value (HSV) coordinate system, making segmentation based on color much simpler by minimizing the effects of lighting. This step uses a DSP tool not mentioned in class.** This was implemented using the MATLAB command *rgb2hsv*. The hue (Fig. 7) and saturation (Fig. 8) channels show clear differences in the piece locations and are good candidates for segmentation, while the value channel (Fig. 9) is ignored.

IV. Image Segmentation

Segmentation intends to decide whether each pixel in the image belongs to:

1) A red game piece
2) A yellow game piece

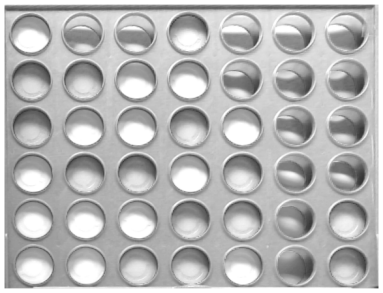


Fig. 9. Value Channel after HSV Conversion

3) The blue board

The MATLAB *rgb2hsv* command creates a hue channel with values ranging from 0 to 1. By looking at this channel, ranges for each color were determined manually: $0 < x < 0.07$ for red, $0.14 < x < 0.18$ for yellow, and $0.59 < x < 0.65$ for blue.

While this method fully captured the pieces, some noise is introduced in the empty board spaces. Looking at the saturation channel however (Fig. 8), the empty board positions are close to black as they contain no bright color. A basic intensity threshold applied to the saturation map results in a mask that contains target areas (bright colored pieces) while neglecting empty spaces. The threshold value for this mask was experimentally set at 0.3 (Fig. 10).

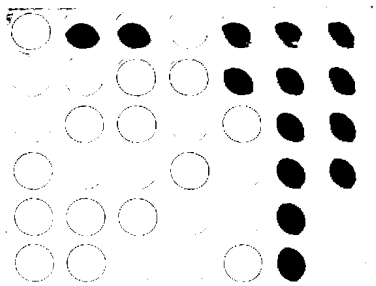


Fig. 10. Saturation Mask: Thresholded at 0.3

An AND operation is used to combine the previous color masks with the saturation mask, which then undergoes a hole-filling operation as the pieces are known to be solid. While this captures the general shape of the pieces, some of the edges are noisy due to the conservative thresholds used (Fig. 11). **A 10-pixel wide pillbox kernel is convolved with the image to smooth out the piece edges. This represents an additional out of class DSP tool.** This results in much cleaner piece segmentation (Fig. 12) for the red, yellow, and blue areas as shown in Figures 13, 14, and 15, respectively.

Each mask is then divided into 6x7 cells, with each cell corresponding to each board piece space. Each cell is then quantified for what percentage of its area is white; a threshold of 10% was experimentally set to determine if a cell contains a piece. The board state can then be quantified as a 6x7 array (Figure 16), with each coordinate representing if a board cell

Fig. 11. Pre Pillbox Filtered Red Piece Mask (Zoomed)

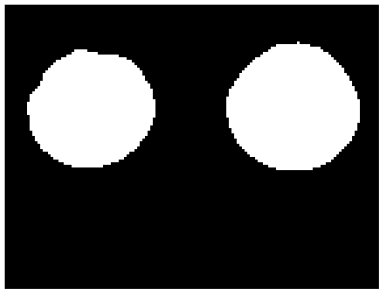


Fig. 12. Post Pillbox Filtered Red Piece Mask (Zoomed)

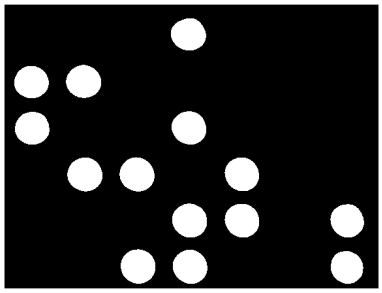


Fig. 13. Hue+Saturation Based Red Piece Mask

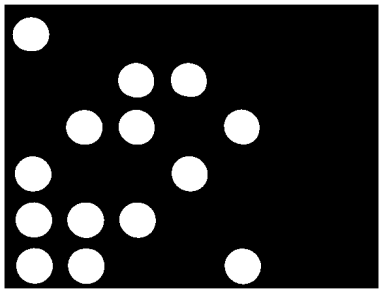


Fig. 14. Hue+Saturation Based Yellow Piece Mask

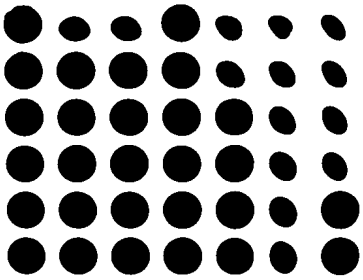
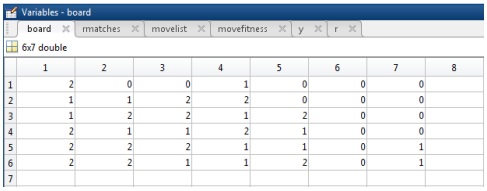


Fig. 15. Hue+Saturation Based Blue Board Mask



	1	2	3	4	5	6	7	8
1	2	0	0	1	0	0	0	
2	1	1	2	2	0	0	0	
3	1	2	2	1	2	0	0	
4	2	1	1	2	1	0	0	
5	2	2	2	1	1	0	1	
6	2	2	1	1	2	0	1	
7								

Fig. 16. MATLAB Array Showing Simplified Board State

is red, yellow, or empty. A value of '1' has been chosen to represent red, '2', is yellow, and '0' is empty.

This matrix is then plotted as circles to display the game state to the player (Figure 17).

V. Computing a Response

A. The Minimax Algorithm

As Connect-4 is a perfect information zero-sum game similar to chess and checkers, the same approaches can be used to compute move responses [3]. The most basic algorithm is known as "minimax", which seeks to "**mini**mize **max**imum loss" while choosing a move. In a basic minimax implementation, the computer plays all valid moves it can make. For each of these moves, the computer then plays all possible human responses. The computer can then play its response to each of those human responses, and so on, forming a "tree" of future game states. Ideally the computer would be able to compute

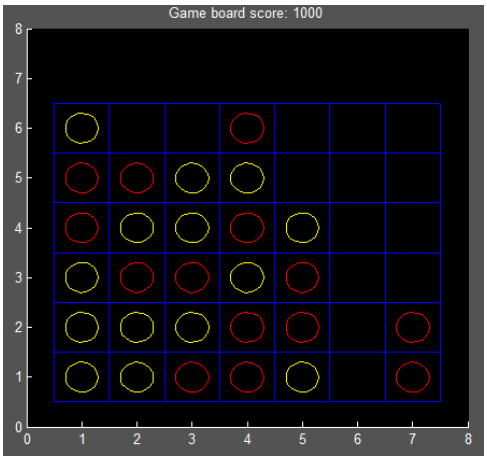


Fig. 17. Drawn Representation of Game State Matrix

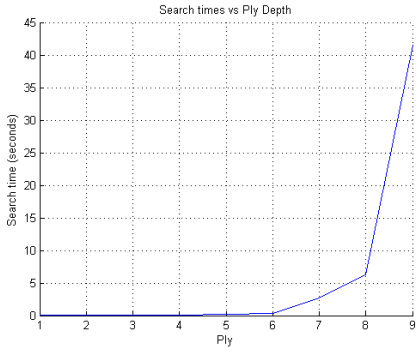


Fig. 18. Time to Compute An Arbitrary Search Depth

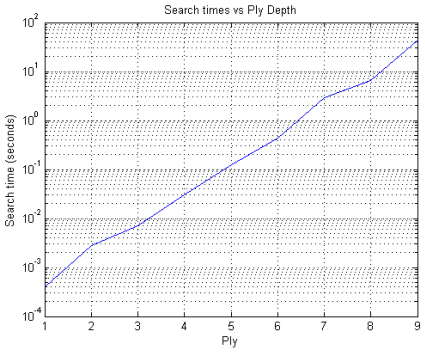


Fig. 19. Time to Compute An Arbitrary Search Depth (Log Scale)

all game possibilities, however this is infeasible in Connect-4, as the number of possible game states on a 6x7 board is $4,531,985,219,092$.

Thus, the computer can only "look forward" a certain amount of moves in a game tree, then evaluate the terminal nodes for favorability. This number is referred to as "search depth", and is measured in "plies," where one ply is one player placing a piece. Assuming all columns of the board are open to drop a piece, the number of possibilities evaluated is

$$O(b^d),$$

where b is the branching factor (number of open columns, usually 7) and d is the search depth.

The number of possibilities quickly becomes unwieldy: looking 5-plies ahead means searching 16807 possiblities, 6-plies is 117649 possiblities, and 7-plies is 823,543. Figures 18 and 19 show the effects of this exponential complexity on execution times.

B. Scoring The Game States

Because most terminal nodes of a search tree do not contain wins, the minimax algorithm must know which trees are "better" outcomes for the computer. This is done by a scoring algorithm, which uses simple heuristics to determine which player is winning for any given board sate.

In this implementation, a negative score means the human is thought to be favored, and a positive score means the

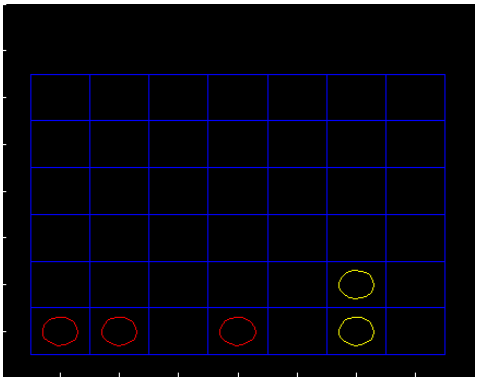


Fig. 20. Example Board with Score -11

computer is thought to be favored. The scoring is based on several rules, of which are assigned points. The points are then summed to form the final board score. The rules are as follows:

1) Four adjacent spaces contain 3 pieces of the same color, and one empty space: 10 points to the controlling player.
2) Three adjacent spaces contain 2 pieces of the same color, and one empty space: 5 points to the controlling player.
3) A corner space is controlled: 1 point to the controlling player.
4) One of the two central spaces is controlled: 3 points to the controlling player.

An example board is shown in Figure 20, which has a score of -11. This is due to condition 1 being met for red (-10), condition 2 being met for both red and yellow $(+5, -5)$, and condition 3 being met for red (-1).

C. Alpha-Beta Pruning

A naive minimax implementation searches all terminal nodes at a specified depth in order to return the highest scoring move. In practice, an exhaustive search is unneeded, as sometimes a branch being searched can be determined to be worse a previously searched branch, and therefor cannot influence the final outcome. When this happens, the branch is "pruned" by stopping the search, proceeding to the next branch as usual. This strategy is known as "alpha-beta pruning", and aims to decrease computational complexity while returning an identical output as a normal minimax search. An ideal alpha-beta implementation will search best moves first, yielding an complexity improvement of $O(\sqrt{b^d})$, allowing the search to run twice as deep in the same amount of time. At worst, the complexity reverts back to standard minimax: $O(b^d)$ [4].

In implementation, the minimax performance was boosted from searching 5-ply in ˜7 seconds to searching 8-ply in the same amount of time (approx. 350 times faster).

The alpha-beta pruning also has a very interesting side effect. If the game extends past turn 30, the board often has very limited moves: either many columns will be completely full, or playing a column will result in an instant loss for a player. Because of this, most trees are immediately pruned, allowing the algorithm to easily search to depths of 15 or higher. This results in the computer playing a stronger endgame, forcing a win from many moves in advance.

VI. Testing and Outcomes

In over 60 played games against 12 humans, the computer has lost three times. Average move time is approximately 8 seconds. The realtime board image capture is resistant to bumps and occasional loss of corner tracking due to an arm or hand being in the field of vision, however is temperamental in different lighting conditions. The implemented AI is extremely strong against ~~human players of any skill level~~ graduate students. Future improvements to the system could include:

- Not using MATLAB for speed boosts, increasing search depth
- Implementing a pattern based corner tracking system
- Use machine-learning based methods and datasets of Connect-4 games to weight the scoring heuristics

Overall, I was happy to learn new variations on image segmentation techniques, and I was able to program my first game AI. It was a lot of fun!

References

[1] V. Allis, "A Knowledge-based Approach of Connect-Four", Vrije Universiteit, Amsterdam, The Netherlands, 1988.

[2] M. Chan, "File Exchange", MATLAB Central, 09 March 2012. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/35531-perspective-control–correction. [Accessed 7 December 2014].

[3] Cornell Department of Computer Science, "Assignment A4: Connect 4 AI," 2014. [Online]. Available: http://www.cs.cornell.edu/Courses/cs2110/2014sp/assignments/a4/A4ConnectFour.pdf [Accessed 7 December 2014].

[4] S. J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 2003.