

# Connect Four App for Perfect Play

David Zeng  
Stanford University  
davidyzeng@stanford.edu

## Abstract

*Connect Four is a strongly solved board game; the global optimal move for every game state can be determined. Connect Four is usually played with the physical board game rather than on a computer. This project bridges accessibility between the Connect Four solver and the physical board game by using traditional computer vision techniques to identify the game state for input to the solver via a mobile application. The application takes a picture of the game board and returns the next optimal move by displaying an augmented image.*

## 1. Introduction

Connect Four is a board game where two players alternately place pieces in a vertical grid. The objective is to create a game state where four pieces of the same color are contiguous horizontally, vertically, or diagonally. Connect Four is a strongly solved game and the global best move can be calculated from any game state [1]. A player going first can guarantee victory when playing perfectly. A player going second can guarantee victory if the other player makes at least one mistake.

These factors make Connect Four an attractive game to approach because we can leverage modern computational power to dynamically calculate the best move in near real-time.

However, Connect Four is usually played as the physical board game while these computational methods are all virtual. By creating a mobile application (app), we can bridge this disconnect and take advantage of the available computing power in the cloud and on mobile to augment reality. The basic idea is to use the app to take a picture of the game board in order to calculate and communicate the next optimal move to the user.

## 2. Related work

There are no existing computer vision publications on Connect Four. However, there are computer vision techniques that are closely related to this project for object recognition. Thus, we focus on techniques used in this

project as well as related technologies that were considered.

### 2.1. Detectors and Descriptors

A popular keypoint and descriptor technique is scale-invariant feature transform (SIFT) (Figure 1) [2]. To detect keypoints, SIFT approximates a Laplacian of Gaussian (LoG) edge detector with a difference of Gaussians for more efficient computation. To achieve multiscale performance, SIFT uses a Gaussian pyramid. Extrema and corners are then detected in this pyramid across each image level and across scales.

The SIFT descriptor uses a 16x16 neighborhood around the keypoint. Histograms of gradient orientations of 4x4 sub-blocks are created and concatenated. The values of the histograms are Gaussian weighted based on distance from the keypoint and a threshold is also applied to the histograms for robustness to illumination variations.

Inspired by the SIFT keypoint and descriptor, a faster method was created: Speeded Up Robust Features (SURF) [3]. Further approximating LoG, SURF uses box filters via implementation in integral images to quickly calculate edge images.

The SURF descriptor uses wavelet coefficients in a neighborhood of 20x20 with 4x4 sub-blocks. The sums and absolute sums of horizontal and vertical wavelet coefficients compose the descriptor. Analysis of SURF shows that it is significantly faster than SIFT but is less robust than SIFT when handling viewpoint variation and illumination variation.

There are alternative keypoint detectors and descriptors that trade robustness for speed but we prioritize robustness for this project and SIFT and SURF are sufficient.

Alternatively, instead of detecting keypoints, detecting blobs may be equally useful if the region of interest is largely uniform. Traditional blob detection has utilized multiscale implementations of LoG via image pyramids [4]. There is a peak when the scale of the kernel matches the scale of the blob. Similar to SIFT and SURF, LoG may be approximated by a difference of Gaussians or box filters via integral images for performance increases.

Once a blob has been detected, descriptors such as SIFT and SURF may be used to identify the blob.



Figure 1. Example of SIFT keypoints and descriptors [5].

## 2.2. Image Correspondence

If we know *a priori* that two images have the same or similar subjects, we can transform the viewpoint of one image to correspond to the other. A common method to calculate this correspondence is to extract keypoints from both images and find matches between the two. We can then use RANSAC [6] to filter out inconsistent matches (Figure 2). With these matches, we can solve a projective transform of one image to the other.

The corresponding point between images do not have to be from keypoints though; corresponding points may be derived via any method. For a projective transform, a minimum of four matching points are required.

## 2.3. Line Detection

Detecting lines in images is useful because many real objects are designed to be rectangular prisms and thus have well-defined edges. A common method for detecting lines is the Hough transform [7]. A Hough transform takes every point and projects it in a parameter space such that it spans all possible lines that it could belong to. When all points in the image have been projected, maxima arise from the intersection of many projections. These maxima correspond to likely lines in the image.

## 3. Methods

There are two primary components for this project. The first is processing the image to extract the current game state for input to the Connect Four solver. The second is the implementation of the iOS app to utilize this project on mobile.

### 3.1. Game state extraction

The input to the system is the raw RGB image. The first step is to generate a silhouette of the game board. To accomplish this, we first convert the image from RGB to

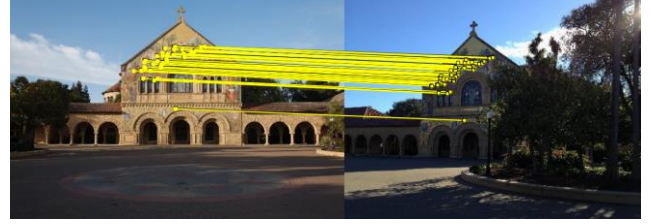


Figure 2. Example of keypoint correspondences after RANSAC [5].

HSV. This serves to build robustness to illumination variation. We determine the hue of the blue game board and then create a L1 distance threshold to detect the board. The silhouette can broadly detect the game board. By using image hole closing techniques, we can clean up the silhouette to be usable (Figure 3(b)).

Next we extract the edges of the silhouette and use a Hough transform to acquire the primary lines of the outline of the board. We merge collinear line segments and place a threshold on the length of the found lines to increase robustness. With the lines found, we then determine the points of intersection of the found lines. Sometimes we may find more lines than the four edges of the game board. In this case, we choose the four lines which intersect in the smallest quadrangle.

The heuristic behind this choice is that the interior of the game board should not produce any lines as it has no straight edges since it is entirely circles. There is also a minimum area to reduce noise. The results are shown in Figure 3(c).

With the four corners of the game board identified, we now have sufficient points to calculate a homography of the input image to a rectified canonical image (Figure 3(d)).

With the rectified game board and known corners, identifying the state of the board becomes straightforward. We first identify the locations of pieces of each color. To identify the red pieces, we first pixel-wise normalize the RGB image. We then extract the red color channel from the normalized image and threshold (Figure 3(e)). To identify the yellow pieces, we convert the image to HSV and threshold based upon the hue L1 distance (Figure 3(f)). Both methods are philosophically the same but RGB has an advantage to HSV in the case of red because red wraps around the hue channel in HSV. This makes extracting red more complicated whereas in RGB, an entire channel is already dedicated to red. With both image masks, we close holes by two dilations in the image to remove noise. We then label the image to identify discrete pieces.

We now divide the game board into a grid. The grid and identification of the pieces may not be perfect. Accounting for this, we iterate through each labelled piece and choose the cell with the most pixels of the silhouette as the cell to which the piece belongs (Figure 3(g)). The process can be interpreted as a generalized Hough transform. This results in a fully determined game state.

With the game state determined, we now submit it to a Connect Four solver [8] whose API we have slightly altered to fit our system. Briefly, the solver explores all possible future game states via a minimax tree. By taking advantage of alpha-beta pruning, the tree can be efficiently searched.

The solver returns the location of the optimal next move and we draw the next piece in the appropriate position as well as display the column number of the optimal move in the top left corner (Figure 3(h)). Column 1 is the left-most column.

### 3.2. Mobile app implementation

The app takes a picture using the camera and sends it via FTP to a server. The server is triggered on receipt of the file to run the MATLAB implementation of the game state extraction and create the final image with the next move. This image is then pushed back to the mobile device and displayed.

## 4. Results and Discussion

The processing pipeline results are shown in Figure 3. An actual playthrough of the game is shown in Figure 4. The app worked surprisingly well. On moves (iii), (ix), and (x), the images had to be retaken because game board detection did not recognize all four edges of the game board correctly. Although it does not work on the first try every time, it is encouraging that by taking new pictures, the method can work on subsequent attempts.

The initial thought is that silhouette detection does not have to be robust because we are looking for edges. We only have to detect fragments of each edge to extract line segments. The line segments do not have to intersect on the image because we can extrapolate the true line to calculate points of intersection.

When we are choosing which lines to use, the assumption that the smallest quadrangle is the true game board usually holds. It has held that the game board does not generate edges within it but at large oblique angles, the legs of the game board begin to be detected as edges and this can result in a pathology in the method.

At the same time, silhouette detection cannot catastrophically fail. Figure 5 (e), (f), (i), (j), and (m) are examples when silhouette detection failed. Diagnosing these cases, we see that the most common mode of failure is when the background color is too similar to blue and detected as an extension of the game board. There are many natural colors with hues similar to blue including the sky and this may lead to incorrect results.

Even knowing this pathology, the solutions are not immediately apparent. If the blue color of the game board is an unreliable descriptor, then the only other unique aspect is its shape. Detecting a particular shape is more difficult, especially if detection must be projectively invariant. One potential idea is to use blob detectors and if enough blobs are arranged in projective grid then we can assume that the

game board must contain these blobs. However, even at face value this method is already more easily proposed than implemented.

If we can detect the four corners correctly, the homography is robust and the remaining methods can perform independently of previous processing. This is based on a strong assumption that the four corners detected correspond to the true four corners. The strength of this assumption leads to more robust techniques down the pipeline but also requires that previous techniques are robust. This requirement is not necessarily met so there are many times in which the proposed method for this project fails. In Figure 5(l), we see what can happen when this assumption fails. The game board is incorrectly detected and the homography results in an incorrect grid for piece detection.

Even when the corners are properly detected, piece detection can be challenging. Figure 5 (b) and (h) are examples of when red is incorrectly detected. In both of these cases, the background is similar to red. These pathologies are not discouraging because the method is working exactly as intended. The pieces are identified almost exclusively on color and in this case, the technique is achieving few false negatives. Furthermore, even to a human from a distance, it would be unreasonable for them to quickly determine which cells had red pieces and which cells were empty with a red background. This suggests that future work needs to somehow differentiate between red pieces and red backgrounds of exactly the same color. Perhaps the texture of the pieces or the matte reflectance could be identified.

On the other hand, yellow piece detection does not perform as well. Figure 5 (a), (d), (g), (n), (q), and (r) show examples of incorrect yellow piece detection. Figure 5(d) can be explained by the argument above for red piece detection. Figure 5 (a), (n), and (r) show false positives for yellow piece detection. In these cases, the background colors are being detected as yellow and this not an uncommon situation since lighting usually has a yellow tint. One way to combat this is to check for board consistency. Physically, pieces cannot float in the game board without pieces below it so this could be one check. Another check could be to count the instances of each piece such that the number of red and yellow pieces are within one since players alternate turns. When checking for number of pieces, if an incorrect of pieces is detected, it would still be ambiguous to determine which pieces are false positives.

Yellow piece false negatives are shown in Figure 5 (g) and (q). In these cases, it seems like the background has changed the white balance of the images and the hue of the pieces has been shifted outside the threshold. Detecting yellow is again difficult because yellow is particularly sensitive to white balance changes and this affects the fundamental color assumption in our method.

White balance is also interesting because even in the

same real world lighting conditions, adjusting white balance effectively adjusts illumination. Cursorily looking over the images in Figure 5, we can see that the game board has varying degrees of apparent illumination even though only the background is changing. Thus, illumination robustness is especially important due to processing by the image acquisition stack.

Even with so many failure cases, there are several instances in which the method performs correctly. In addition to the complete game played in Figure 4, Figure 5 (c), (k), (o), and (p) correctly detect the state in very different situations.

Figure 5 (c) and (k) correctly detect the state when the background is a benign color relative to the colors being detected. Figure 5 (o) and (p) demonstrate the rotation invariance of the homography and Figure 5(o) is particularly impressive in that its background is cluttered.

#### 4.1. Alternative methods

The original plan for game board identification was to use SIFT keypoints and descriptors in order to calculate a homography with RANSAC with a canonical rectified game board. This method did not work well because the game board is all circles. The circles do not generate any SIFT keypoints. This led to trying blob detectors with SIFT descriptors. The circles of the game board are detected but the keypoint matching does not work well because we are then using the background to match keypoints. However, the background of the canonical reference image is not going to match with an arbitrary background. Even just using only the blob locations without descriptors is not robust because not all of the game cells are necessarily detected as blobs. This would require further work to determine the true extent of the game board. If the keypoints are not matched correctly then RANSAC will not work. The keypoints could be heuristically matched but even then the method is not guaranteed to be robust.

This reveals the actual intent of rectifying the image. We are not truly after image identification since we *a priori* assume that the game board must be in the picture but we are actually interested in localization of the game board. Keypoints are the traditional approach but in this case cannot define the game board well enough. Interestingly, edge detection via silhouette and Hough transform are not the first thoughts that arise when approaching this problem but this method provides important exact localization information.

There are many heuristic-based methods in this implementation. Many of these work well in controlled environments but they may fail quickly in arbitrary environments. Improvements to these methods could come from more advanced methods such as convolutional neural networks (CNNs). Every step of this project could actually be implemented in a CNN. In fact, it would not be unthinkable that a CNN could perform the entire game state

detection pipeline.

#### 5. Future Work

The original idea for this project was to create an augmented reality app for solving Connect Four. The augmented reality portion proved to be too ambitious for this project. There are object tracking libraries available for object tracking on mobile but many of them are not free or do not work well. As noted above, the game board does not generate keypoints and almost all traditional object tracking methods rely on keypoints. These algorithms could be modified to use blob detection but then again descriptors are a challenge because the game board occludes the background.

Another idea would to track the game board would be to identify an area devoid of keypoints but this would then significantly alter the object tracking algorithmically.

Fundamentally, while the idea of augmented reality in the app may sound novel and exciting, there is actually little utility added. The minimum requirements for this app to function is to display a number to the user to indicate in which column the next piece should be placed. The picture is to create a more intuitive user experience. Augmented reality would further enhance user experience but is an expensive computation.

Currently all of the processing is performed in the cloud. There are two factors for this decision. The first is that all of the algorithms have already been implemented in MATLAB and the Connect Four solver is written in Python. Admittedly, the MATLAB portions could be converted to native code using analogous functions in libraries such as OpenCV [9]. It is a significant resource commitment to run a Python script natively on iOS and the alpha-beta pruning algorithm would probably have to be rewritten.

Furthermore, the Connect Four solver requires significant computation. On a server, the solver can at times use up to 30 seconds of computation to solve its alpha-beta pruning algorithm. Mobile devices are generally less computationally powerful than a server and this could result in significant power consumption by the app as well as unacceptably long waits for the result.

Many concerns of the game state detection method could be addressed by approaching the problem using newer methods such as CNNs. Object detection and localization have been proven to be effectively tackled by CNNs so further development in this direction would not be completely novel [10]. Furthermore, once trained, a CNN could run efficiently on mobile devices as every modern device now contains a GPU.

#### Data and Code

<https://onedrive.live.com/redir?resid=5C6FE3F3DBCF3D94!110477&authkey=!AC1MNLtk5SG22yA&ithint=file%2czip>

## References

- [1] Allis, V. A Knowledge-based Approach of Connect-Four, Thesis, Vrije Universiteit, 1988.
- [2] Lowe, D. Object recognition from local scale-invariant features, ICCV 1999.
- [3] Bay, H. SURF: Speeded Up Robust Features, CVIU 2008.
- [4] Lindeberg, T. Detecting Salient Blob-like Image Structures and Their Scales with a Scale-Space Primal Sketch: A Method for Focus-of-Attention, IJCV 1993.
- [5] Girod, B. EE 368 Course Notes, Stanford University, <http://web.stanford.edu/class/ee368/Handouts/Lectures/>
- [6] Fischler, M.A. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography, Comm. ACM 1981.
- [7] Hough, P.V.C. Method and means for recognizing complex patterns. U.S. Patent 3,069,654, 1962.
- [8] Saveski, M. AI Agent for Connect Four, 2009, <https://github.com/msaveski/connect-four>
- [9] Bradski, G. opencv\_library, Dr. Dobbs's Journal of Software Tools, 2008.
- [10] Redmon, J. You Only Look Once: Unified, Real-time Object Detection, CVPR 2016.

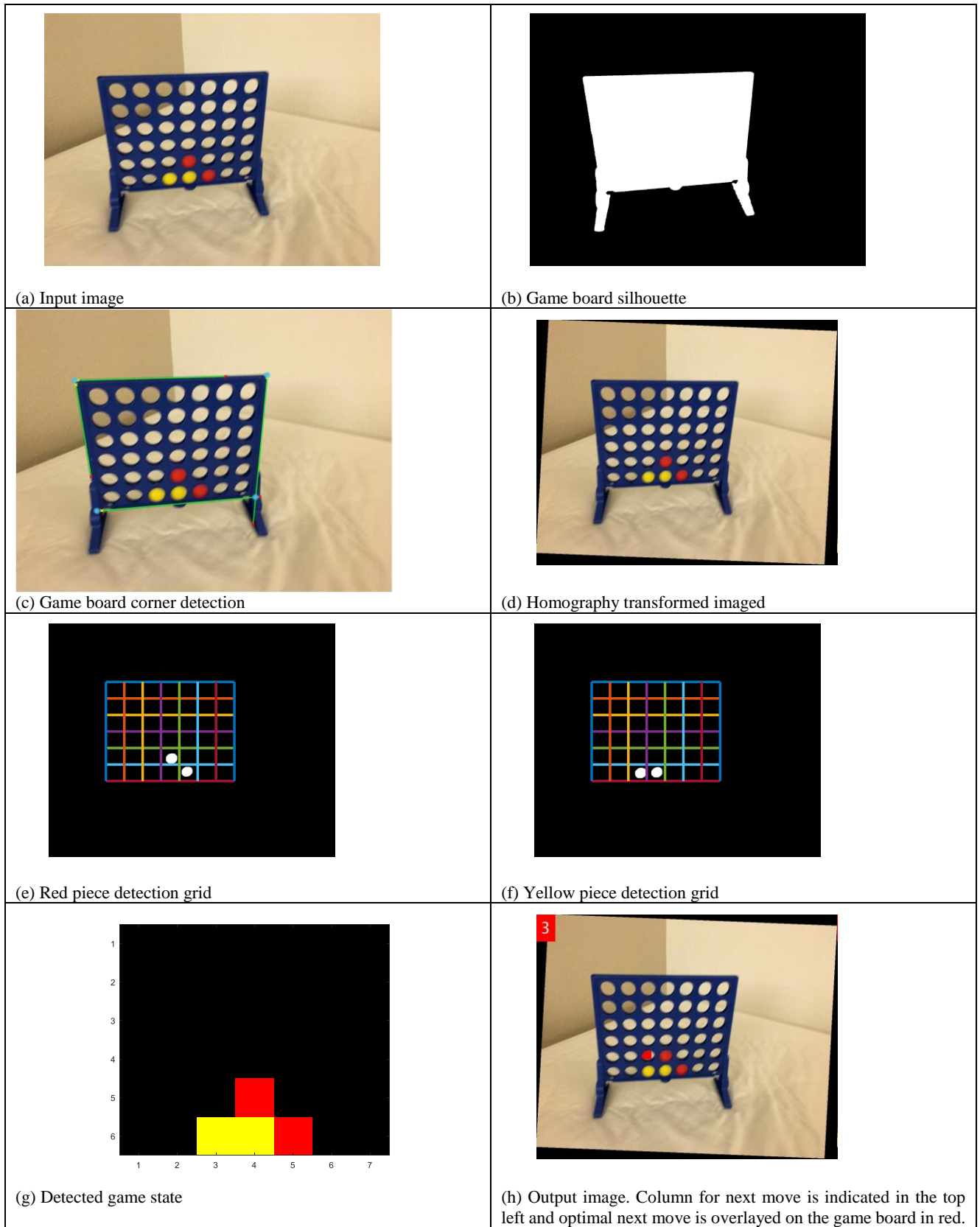


Figure 3. Game state detection pipeline and final image output.

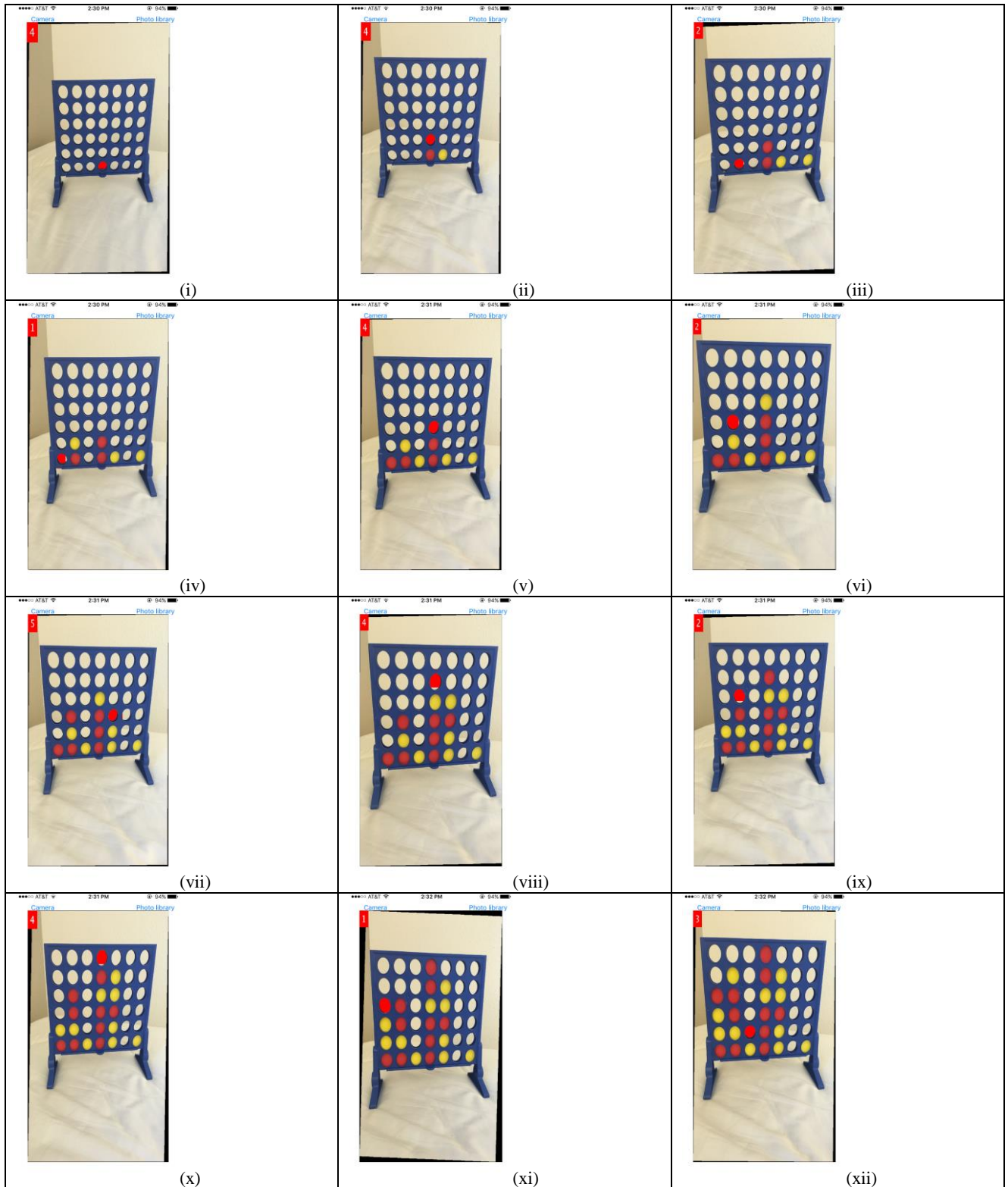
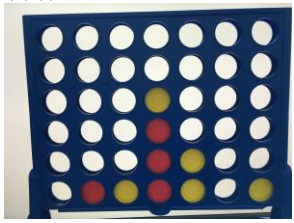
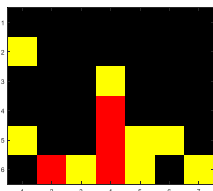
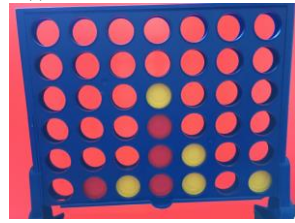
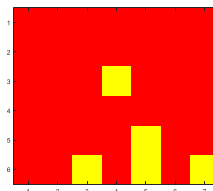
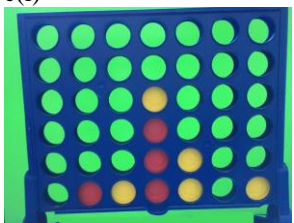
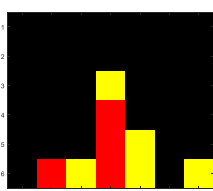
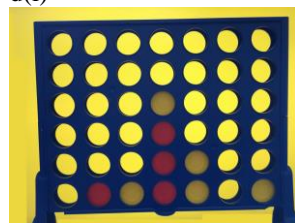
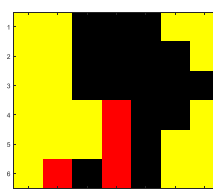
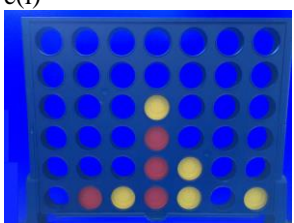

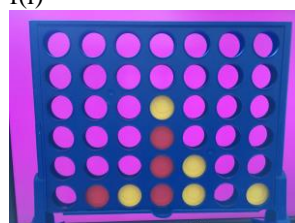
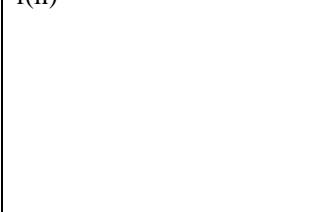
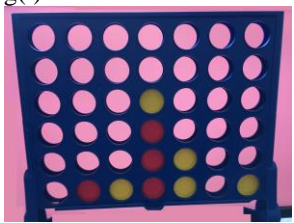
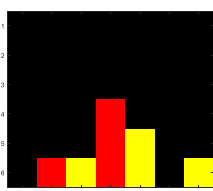
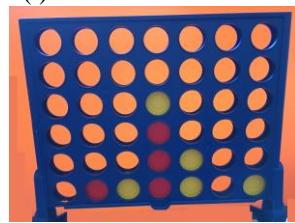
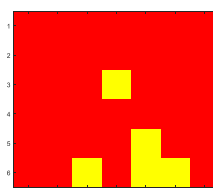


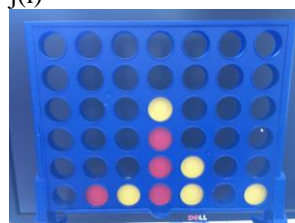
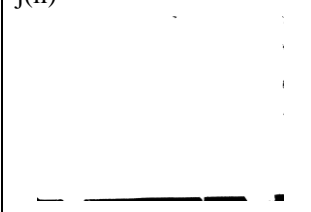

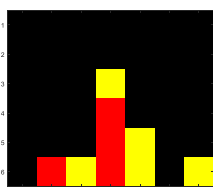

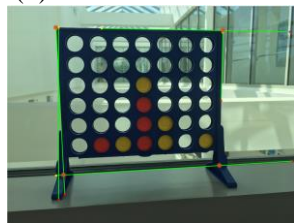


Figure 4. Playthrough sequence of a Connect Four game with the iOS app. The app is making move suggestions for red player against the adversarial yellow player. The next optimal move is overlaid on the game board in red and the column for the move is displayed in the top left corner. The left-most column is column 1. After 12 moves for red player, we see that red has won.

(a)(i) 	(a)(ii)  <p>Yellow incorrectly detected</p>	b(i) 	b(ii)  <p>Red incorrectly detected</p>
c(i) 	c(ii)  <p>Correct state</p>	d(i) 	d(ii)  <p>Yellow incorrectly detected</p>
e(i) 	e(ii)  <p>Unusable silhouette</p>	f(i) 	f(ii)  <p>Unusable silhouette</p>
g(i) 	g(ii)  <p>Yellow incorrectly detected</p>	h(i) 	h(ii)  <p>Red incorrectly detected</p>
i(i) 	i(ii)  <p>Unusable silhouette</p>	j(i) 	j(ii)  <p>Unusable silhouette</p>
k(i) 	k(ii)  <p>Correct state</p>	l(i) 	l(ii)  <p>Incorrect corners</p>



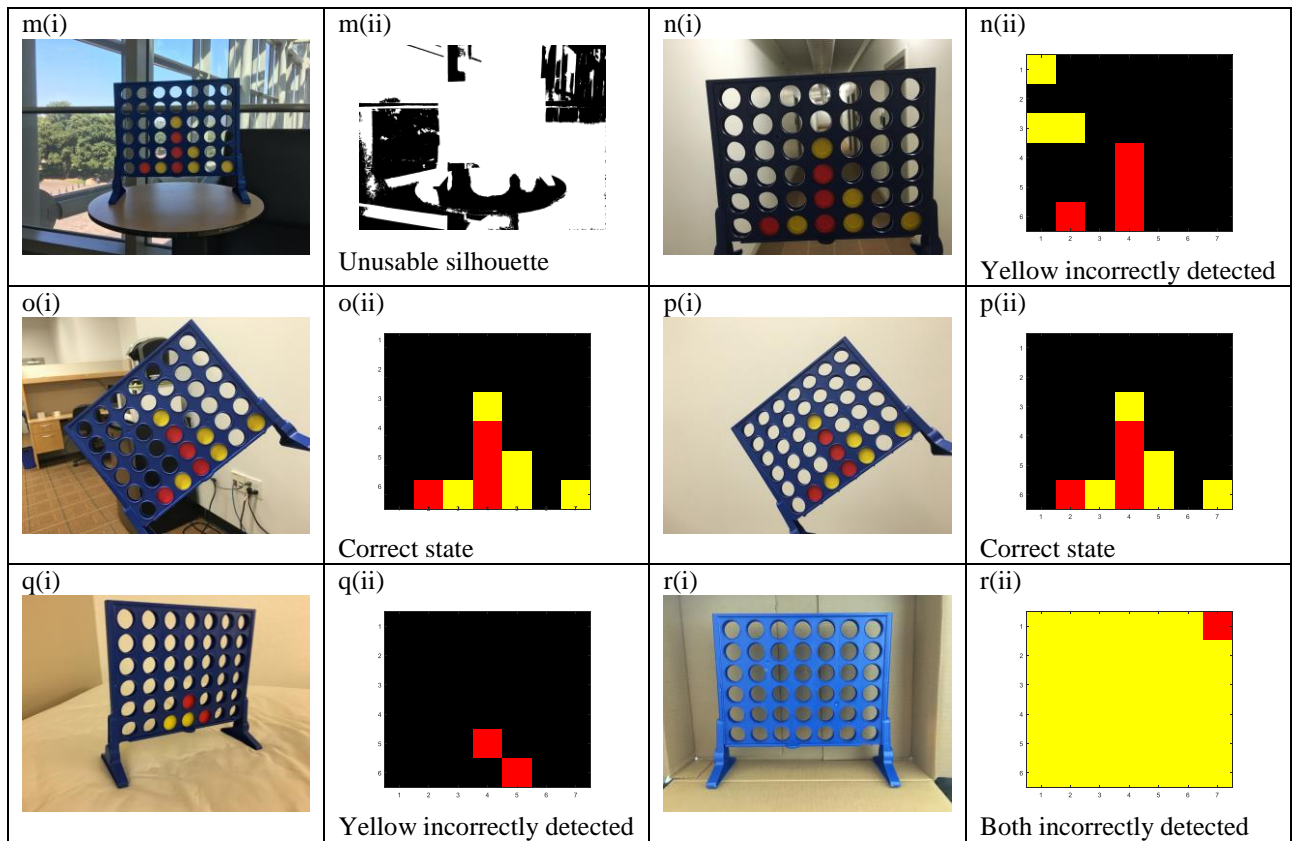


Figure 5. Results of multiple images. **a-k** are the game board held in front of a monitor with different background colors. **l-r** are the game board in various real world locations. The (i) images are the input image taken by the camera. The (ii) images are the first location where pathologies may be observed. From these results, we see that the key pathologies are in proper silhouette formation and detection of the pieces. An unusable silhouette means that game board edge detection fails. Detecting pieces incorrectly usually relates to incorrect color identification.