Image Processing for Board Game Piece Recognition

Noam Eshed

5/3/18

1. Introduction

The purpose of this project is to use image processing techniques to detect, recognize, and locate board game pieces on their respective boards. This project focuses on finding checker pieces on a wide variety of checker boards and at the end includes notes on preliminary results of running a similar algorithm on backgammon boards. The goal is to detect all of the game pieces on the board, and return the board location (i.e. there are checker pieces located at A1, A3, A5, etc. in Figure 1).

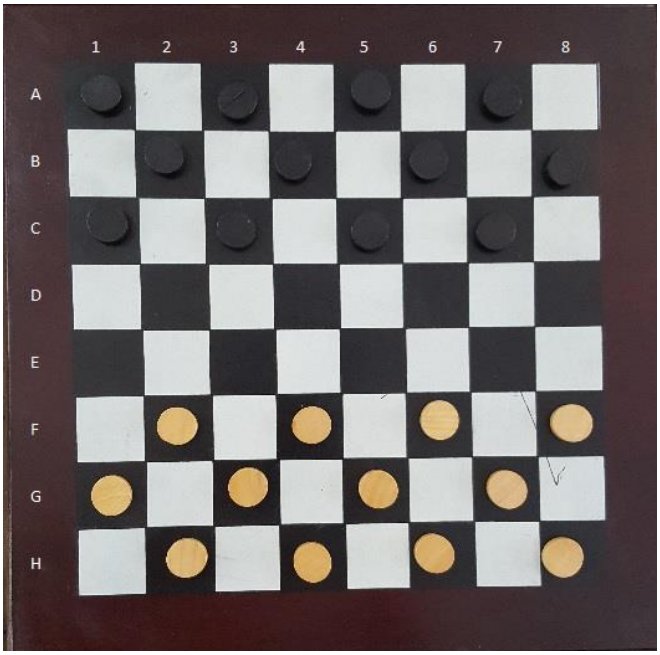


Figure 1: Checkerboard Labels

Object detection and location is relevant in many fields (surveillance and manufacturing to name a couple). It's important to be able to separate an object in an image from its background to allow for further processing and analysis. With the increasing interest in board game AI, such as AlphaGo and its recent victory over 18-time world title winner Lee Sedol [1], detection of board game pieces has applications as part of the framework for these AI algorithms. Game-playing robots are currently of high interest as well, and require a vision system to interpret the board and make decisions which the mechanical arms will carry out.

2. Related Work

This project makes use of image segmentation and edge detection. The outer edges of the board will be detected and aligned with the image edges. The inner edges of the board (those separating squares/triangles on the board, depending on the game) must be detected, as well as the circular pieces.

A team from Osaka International University in Hirakata, Japan published a paper on their work incorporating augmented reality to a popular Japanese board game, Sugoroku. Using a PC and webcam, they plan to detect "markers" on the board (denoted by a black letter or character on a white background enclosed within a black rectangle) and overlay them with 3D images. Their paper does not describe the marker detection methods [2]. Their work is similar to this project in the sense that they are also detecting objects within a simple polygon, albeit for AR applications.

A research group at Stanford University applied image processing techniques to identify a chessboard and the configuration of its pieces [3]. Their work was split between recognizing board squares and recognizing the chess pieces themselves. They noted that using traditionally-colored boards made it difficult for the algorithm to differentiate board edges, and so they substituted a red and green checkered board. The basic algorithm began with getting a binary image using Otsu's method and applying edge detection, followed by using the Hough transform to detect lines, using these lines to detect squares, extrapolating certain squares to identify those not previously found, and applying a transformation to the board. This is similar to the algorithm proposed here, but to simplify the process I propose applying the board transformation first and adding a stage of morphological operations to increase the accuracy of the Hough transform.

Another group from Gunadarma University in Indonesia considered how vision systems can recognize chess pieces and their movements and communicate with the game-playing robot as well. The team used contour detection, rather than edge detection, to map out the board edges, and assumed that the largest contour detected was the edge of the board. They used a Gaussian filter to reduce noise before applying a Canny edge detector to find the contours, then used morphological closing to strengthen the edge accuracy. Next, the Ramer-Douglas-Peucker algorithm was used to find the fewest points to describe the shapes on the board (the vertices). Rectangular contours were accepted as squares on the board. Approximations were made to account for slight lighting variations on the board, but for the most part the images were taken in constant lighting. Geometric transformation and pixel intensity interpolation were used to align the board with the image edges in the case of a tilted camera. Edges of the chess board squares were found by mathematical approximation (dividing the board width by 8) and by edge detection using the Hough Transformation on a binary

thresholded image. The rest of the research focused on detection the movement of pieces by a robot or human arm, and not on recognizing the piece itself, which was done by comparing boards at various stages of the game (via image subtraction) [4]. This research follows a very similar process to the one outlined in this project.

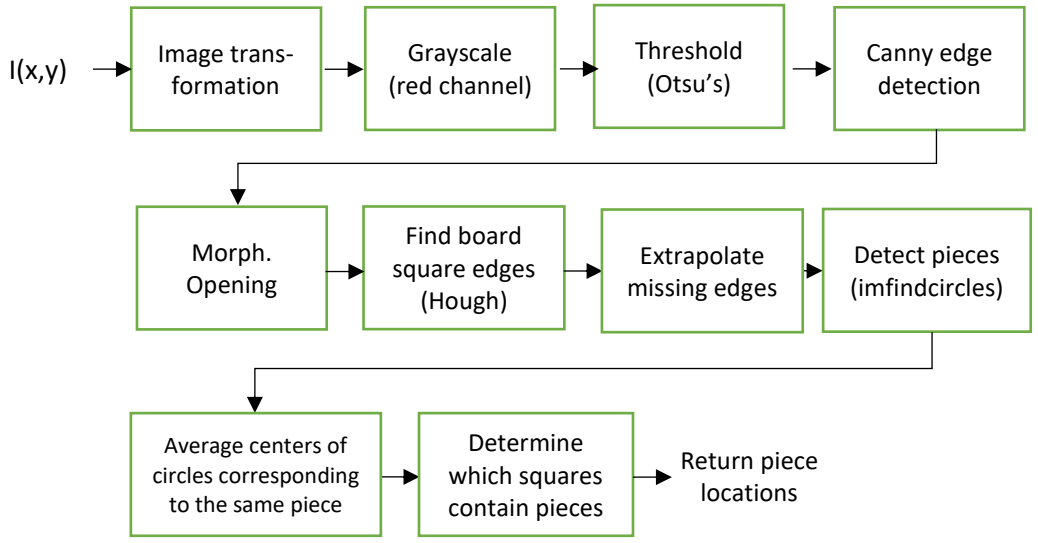
3. Data Collection

I collected images using a Samsung Galaxy S6 camera with 72 dpi resolution. Images are 5312 by 2988 pixels (or 2988 by 5312 pixels). I took 26 photos of checker and backgammon boards. Some photos were taken at different angles, with a variety of game piece configurations, and with different backgrounds (wood floor vs. carpet vs. table). Some were taken in constant lighting, while I placed others on the floor such that the light would fall onto the board in different ways. I collected additional images through many websites found on Google Images in order to get boards with different square colors and piece colors after seeing that it was difficult for my algorithm to separate the black pieces and black squares on my board. These all have different aspect ratios and sizes. In total, I used 50 images.

Figure 2: 3 Samples of Board Game Image Data

4. Technical Approach

Checkers Flow Diagram:



First, the image is shaped and cropped into a 1000x1000 image using a projective transformation (Matlab's fitgeotrans and imwarp) and user input with the locations of the four board corners. This helps straighten out the board to allow for processing of photos taken at an angle, and also avoids the issue of boards having borders with different widths (an example is shown in Figure 3 below).

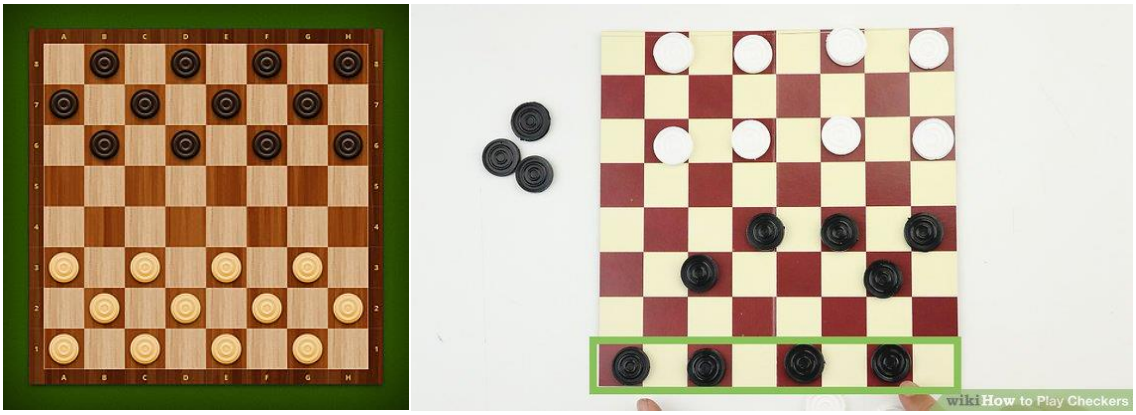


Figure 3: A board with (left) and without (right) a border

Next, the image is converted to binary using the equation $Binary\_im = im < 255 * T$ where T is the threshold between 0 and 1 found by Otsu's method. A Canny edge detector is applied to find the edges in the binary image.

Morphological opening is applied to the image using two length-4 line strels, one of which is horizontal and one of which is vertical. These strels are used to apply opening to the

binary image, and the two resulting images are added up. This helps strengthen both the vertical and horizontal edges separately. An example is shown below:

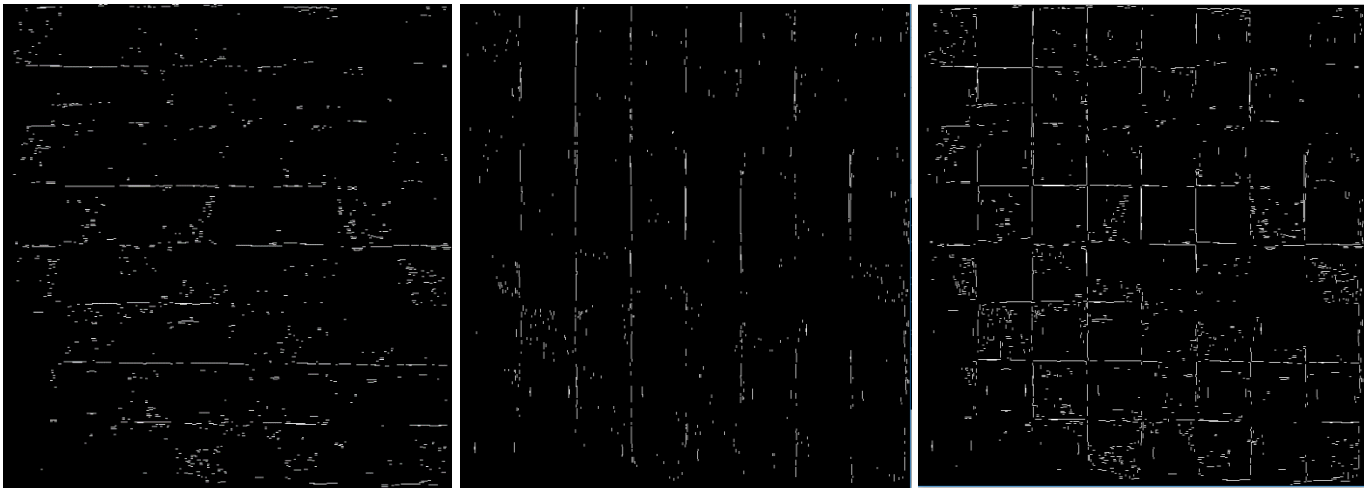


Figure 4: Opening with a) horizontal strel (left), b) vertical strel (center), and c) the sum of a) and b) (right)

A second round of morphological opening is done similarly with longer strels (10 pixels long) to further extend these edges.

Next, the Hough transform was applied and the top 30 peaks in the transform were used to find line segments. Ideally, 18 edges would be detected, since there are 18 lines on a checker board. However, this method often returns multiple segments on the same edge and may miss edges altogether, which is why there is the redundancy in searching for the top 30 peaks. Because of this, the next step is to get rid of edges corresponding to the same line using the following logic:

```
For each Hough edge segment:

   If the slope is near vertical:

      If we already detected a vertical line with the same slope and
      similar x-coordinates, ignore. Otherwise, extend the line to the
      image edges and add it to our list of vertical lines.

   If the slope is near horizontal:

      If we already detected a horizontal line with the same slope and
      similar y-coordinates, ignore. Otherwise, extend the line to the
      image edges and add it to the list of horizontal lines.
```

Often, Hough edges would not find all of the lines on the board. To extrapolate the missing lines, the following was done:

```
If any board edges haven't been found, add to list of lines
While < 9 horizontal lines:
   Find the distances between all neighboring lines
   Assume that the mode of these distances is the width of a square
   For each pair of neighboring lines too far apart:
      Number of lines we need to add in between is the distance divided
      by the square width
      Divide the distance between lines by the number of missing lines,
      and add a line at each of those points
```

The same process is followed to extrapolate the vertical lines.

Six thresholding methods were tested on their effectiveness for circle detection (e.g. how many of the pieces they can detect on the board). These were:

1) Original RGB image
2) Grayscale image (red channel)
3) Otsu global thresholding
4) Otsu local thresholding using 150x150 block structs
5) Simple thresholding (im < T) at T = 55
6) Canny edges on the output of (3)

Since the effectiveness of these methods varied greatly from image to image, and often one method would find pieces that others did not, I used the circles found from all methods to label pieces. In the example below in Figure 5, a combination of 3 methods (original, grayscale, and simple thresholding) would be required to detect all pieces.

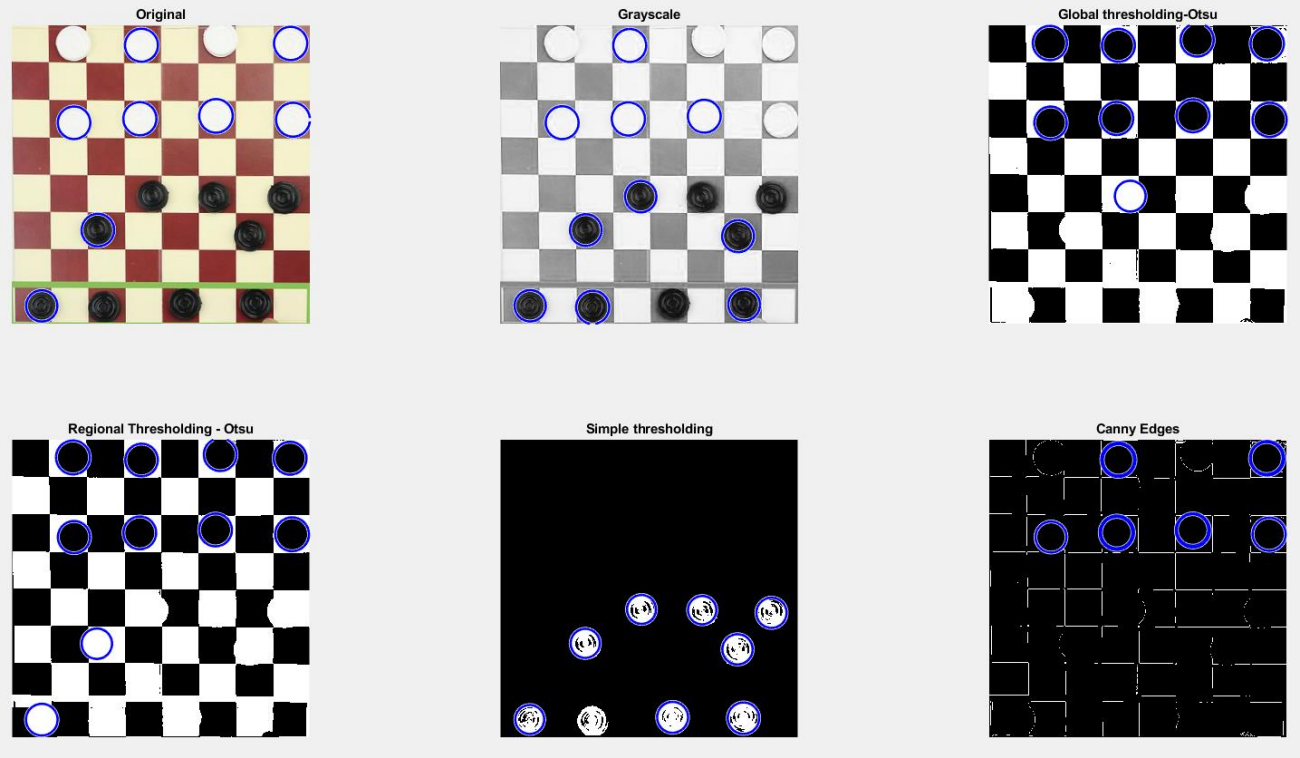


Figure 5: Six thresholding methods for circle detection

This algorithm finds many redundant circles which all correspond to the same piece. To fix this, all circles corresponding to the same piece were averaged to get a single circle as follows:

```
Sort the list of circles by center point

For each circle found (in sorted order):

   If its distance to the next circle on the list is small (<10
   pixels), average the center points and radii and keep that circle

   Otherwise, keep the circle dimensions with no modifications
```

The final step in the process is to compare the locations of circle centers with the locations of all vertical and horizontal lines to determine which squares contain checkers. The output is a list of piece locations (i.e. B3, F7) that follow the naming convention shown in Figure 1.

5. Intermediate Results

The first step was to transform the image into a square. I chose to make it a 1000x1000 square because this size was smaller than most of the images, and thus less data was lost. After the user selects the four corners, the figure is warped so that those points are at coordinates (0,0), (1000, 0), (0, 1000), and (1000,1000) respectively. The result is cropped so that the background is no longer in the image.

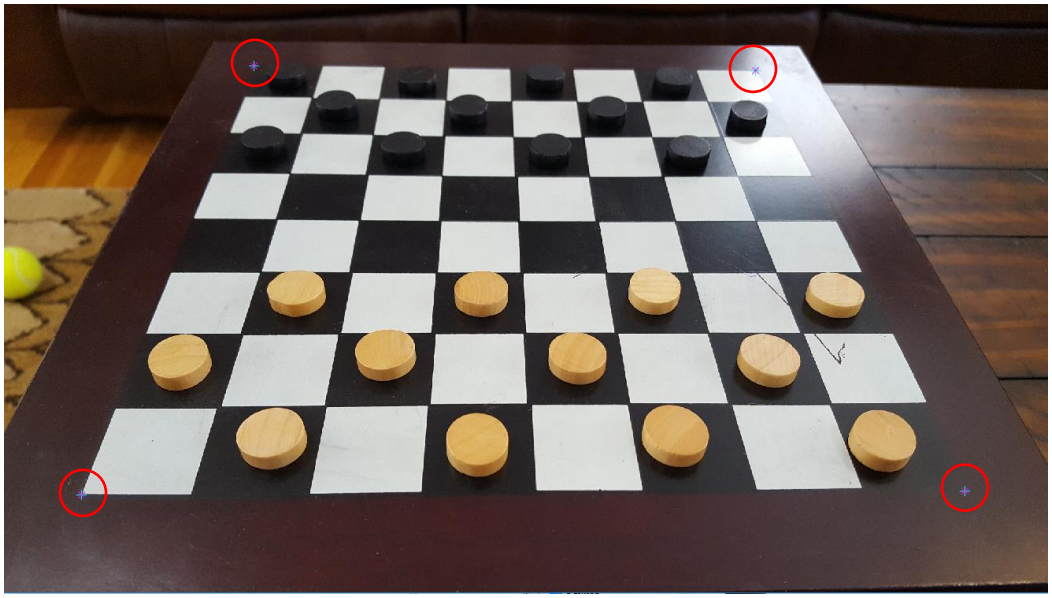


Figure 6: Point Selection (in red circles)

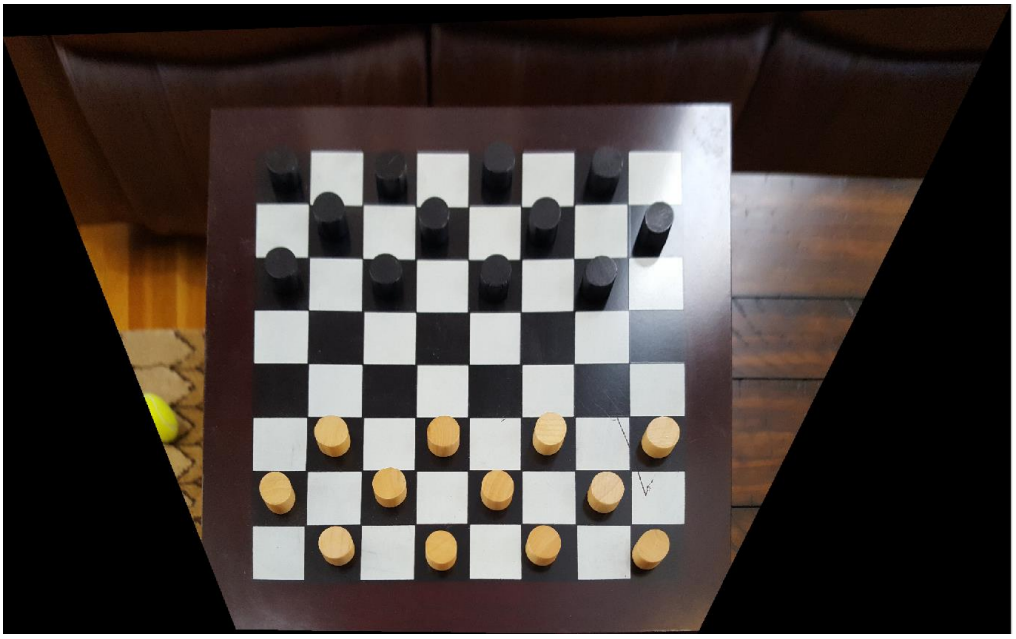


Figure 7: Warped Image

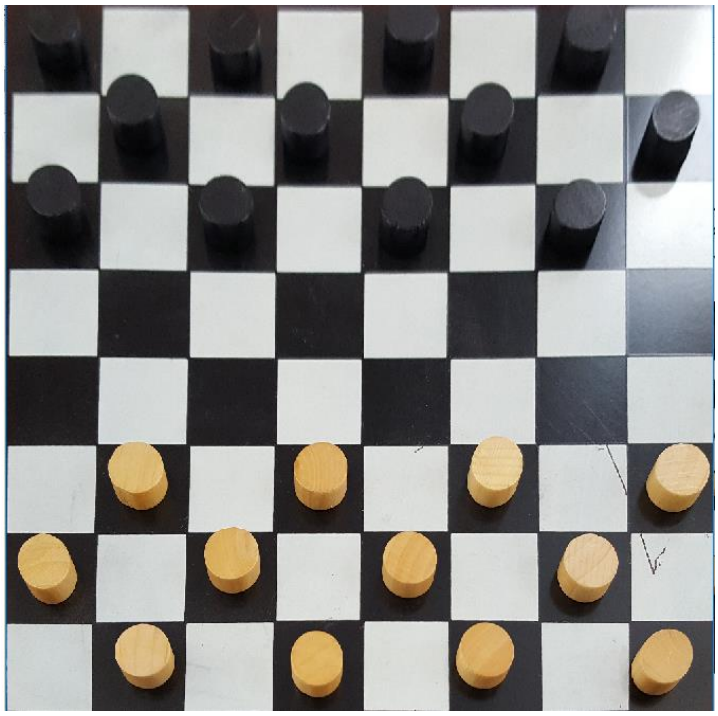


Figure 8: Final 1000x1000 square image

It's clear in this image that when a photo is taken at a low angle, the pieces end up looking tall and warped after the transformation, which makes it difficult for the algorithm to detect them later on.

The two rounds of morphological opening operations serve to clean up noise and strengthen the edges of the board (Figure 9).

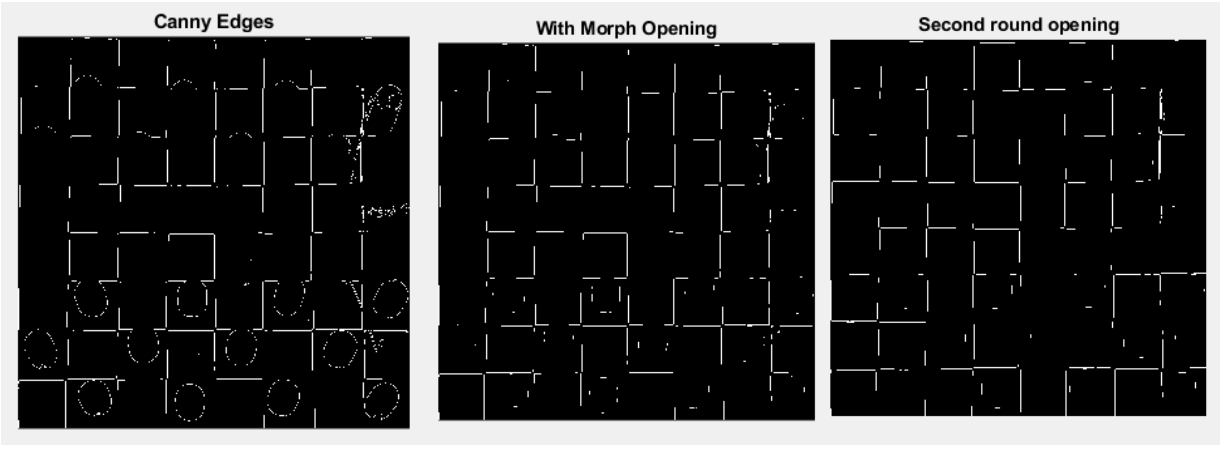


Figure 9: Morphological Opening

The Hough edges found are displayed as green line segments as shown below:

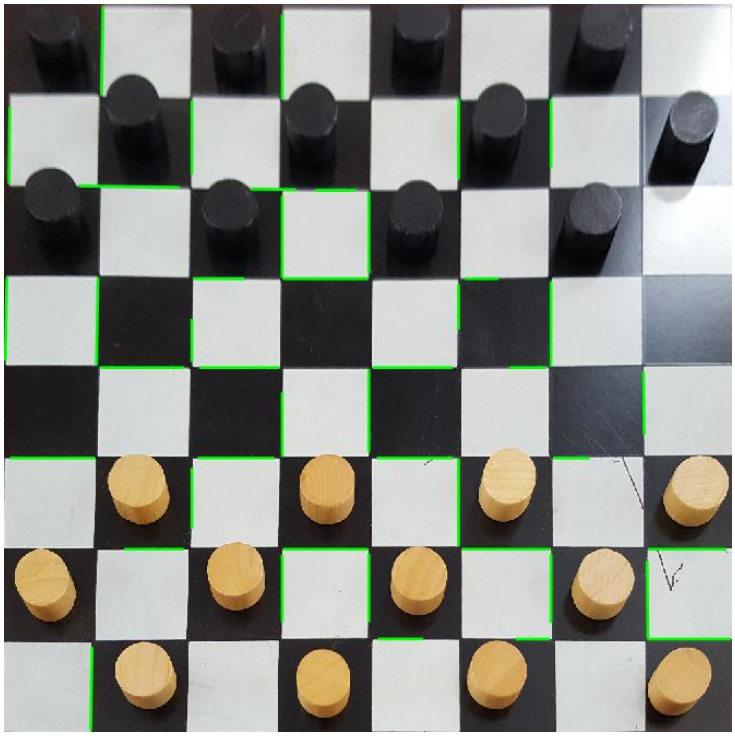


Figure 10: Hough line segments

After combining the line segments which correspond to the same line, all lines are plotted as below. In this example, the algorithm missed all four edges of the board, as well as the top horizontal edge.

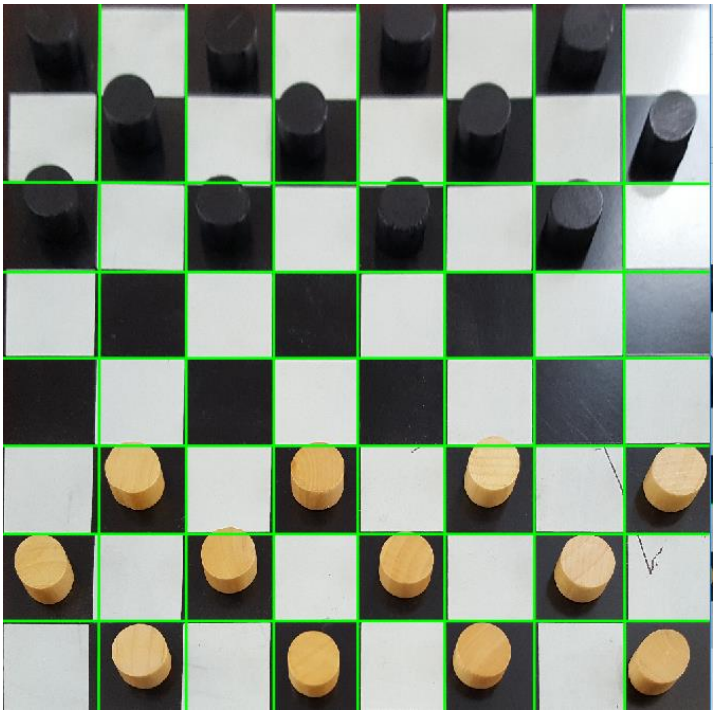


Figure 11: After combining Hough line segments and extending the lines to board edges

Once the extrapolation is complete, all of the edges are accounted for:

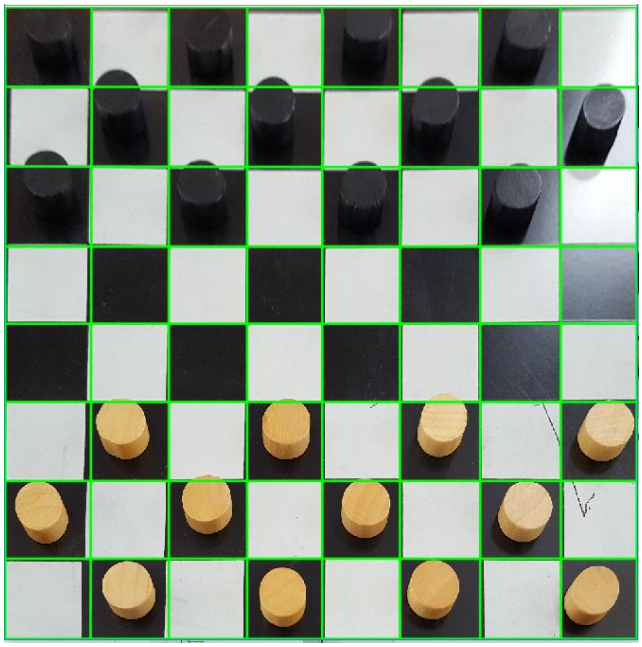


Figure 12: After extrapolation

In the example in Figure 13, Matlab's imfindcircles, the circular version of the Hough transform, was able to detect ten of the white pieces on black squares, but only one of the black pieces on black squares. This is an issue in many of the images where the color of the pieces too closely matches the square color.

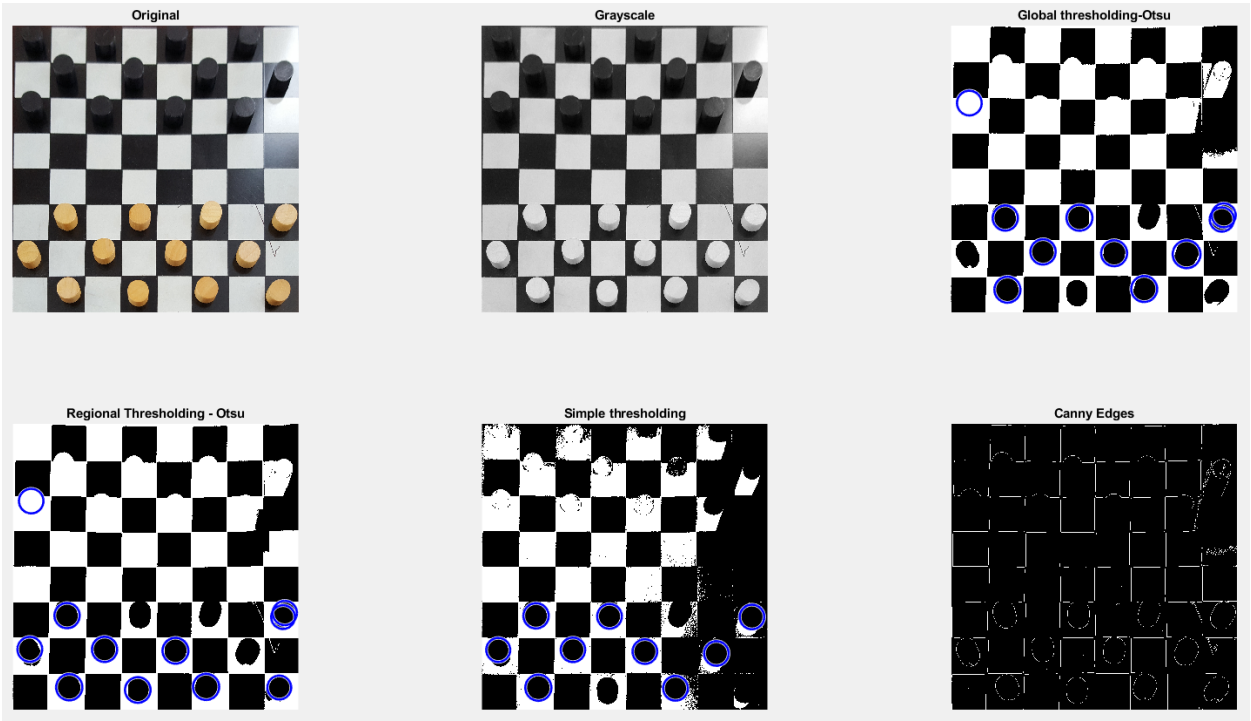


Figure 13: Piece detection on 6 input images

After combining the redundant circles, we can visualize the results of the algorithm. The edges found are shown in green, and the pieces found are circled in blue.

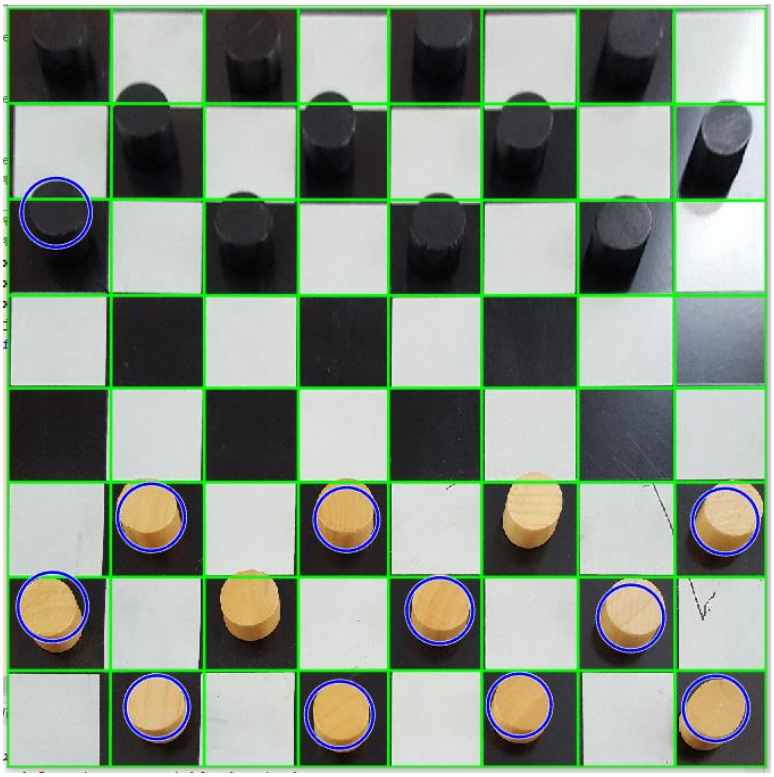


Figure 14: Found edges and pieces

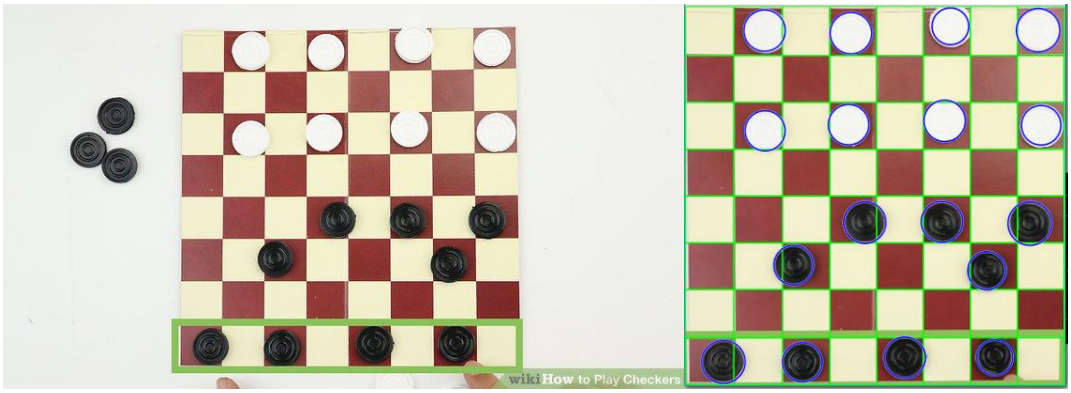
Finally, the code outputs the listing of piece locations:

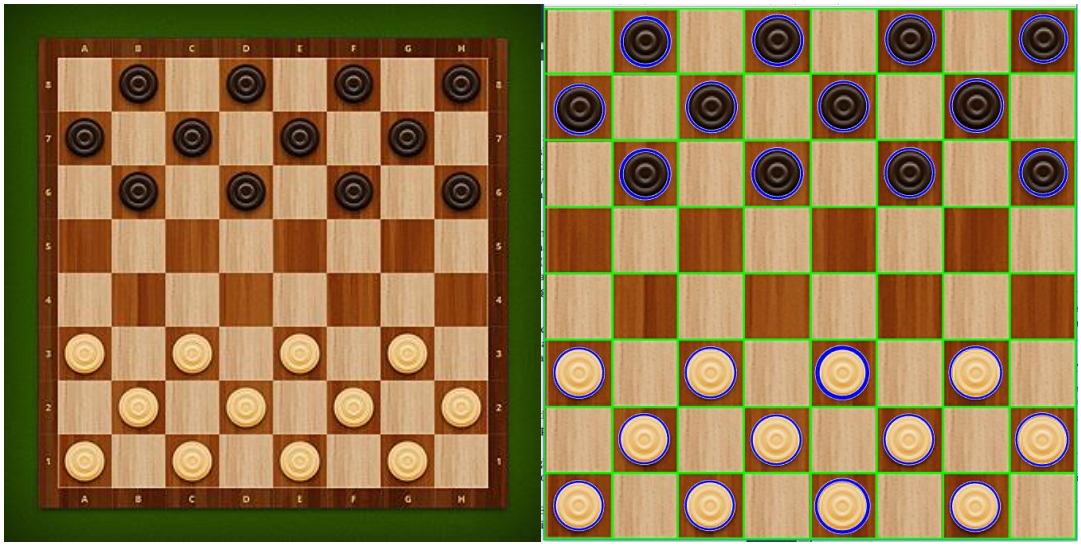
```
There are checker pieces at locations:
C1
F2
F4
F8
G1
G5
G7
H2
H4
H6
H8
```

6. Final Results

This algorithm was strong in detecting board edges and calculating the missing edges' locations through extrapolation, even when the board was angled in the original photo. All edges are always marked, due to the extrapolation phase, and are generally close to the actual edges. Occasionally, when the original image was taken at a low angle and so the transformed image was very warped, lines would be detected in the wrong places.

The accuracy of piece detection depended largely on the image quality. In terms of accuracy, the photos taken directly from above or at a small angle, with a board that had different colors from the pieces, and without light shining on it, did the best. Using computer-generated board images also yielded highly accuracy results. The photos taken directly from above with these ideal conditions found on average 79.0% of the pieces, missing only the black pieces on black squares. Below are three examples of boards which were analyzed with 100% accuracy, so all of the edges and all of the pieces were detected. The original image is shown on the left, and the processed image on the right.





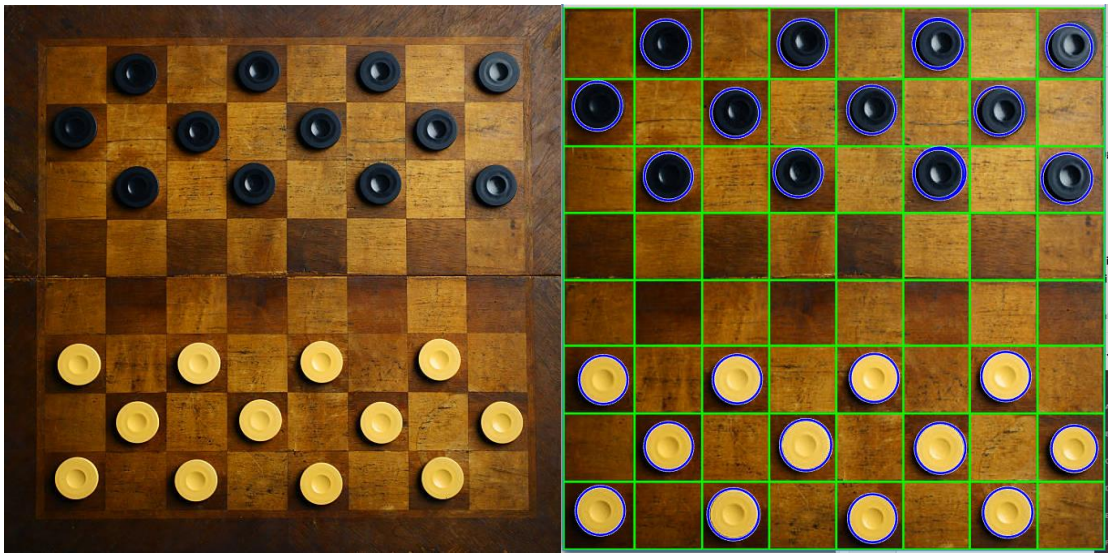


Figure 15: Examples of 100% accuracy. Original image on left and processed image on right

The algorithm struggles with finding circles when the pieces are the same color as the square they are on, and when the image is so warped during the transformation that the pieces are no longer circular enough. For images taken at an angle, the average percentage of pieces found dropped to 32.8%. Some of these examples are shown in the following images.

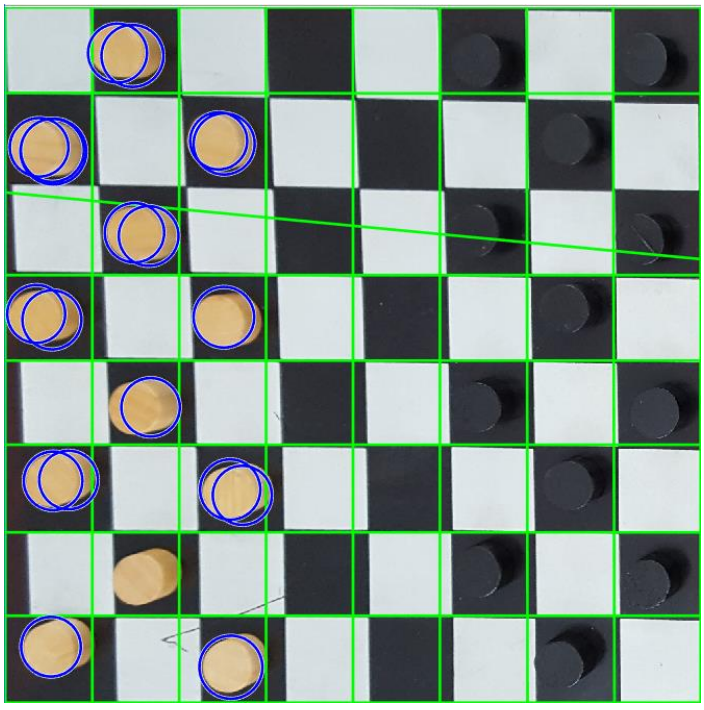


Figure 16: Multiple circles per piece; angled line

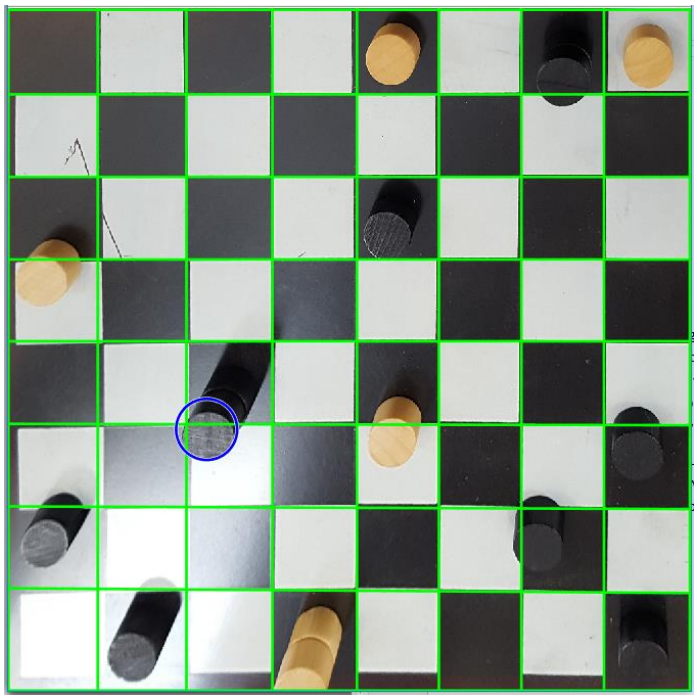


Figure 17: Only one piece found

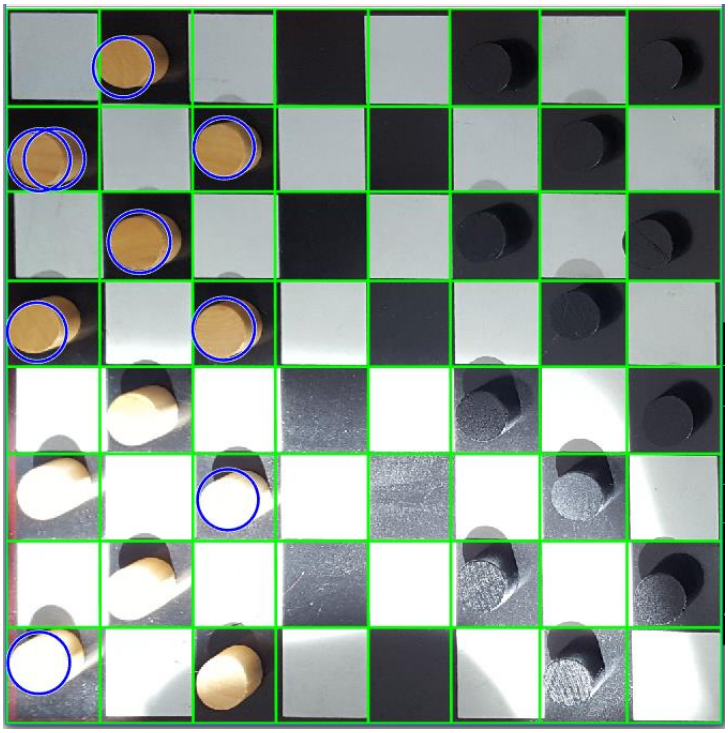


Figure 18: Missed pieces in the sunlight and black pieces on black squares

Some preliminary tests were done using a similar algorithm on backgammon boards. The main differences in the algorithm were:

1) The image is transformed into a 1000x1500 image to keep the aspect ratio of the board
2) The strels used in the morphological opening are angled at 95 and -95 degrees and are 10 pixels long

Morphological opening did not have as positive of a result on the backgammon boards as it did on the checker boards, even though the strels were at the angles of the triangle edges. As a result of this, finding Hough lines in the following step was done on the direct output of finding canny edges.

Figure 19: Backgammon board, transformed to be 1000x1500

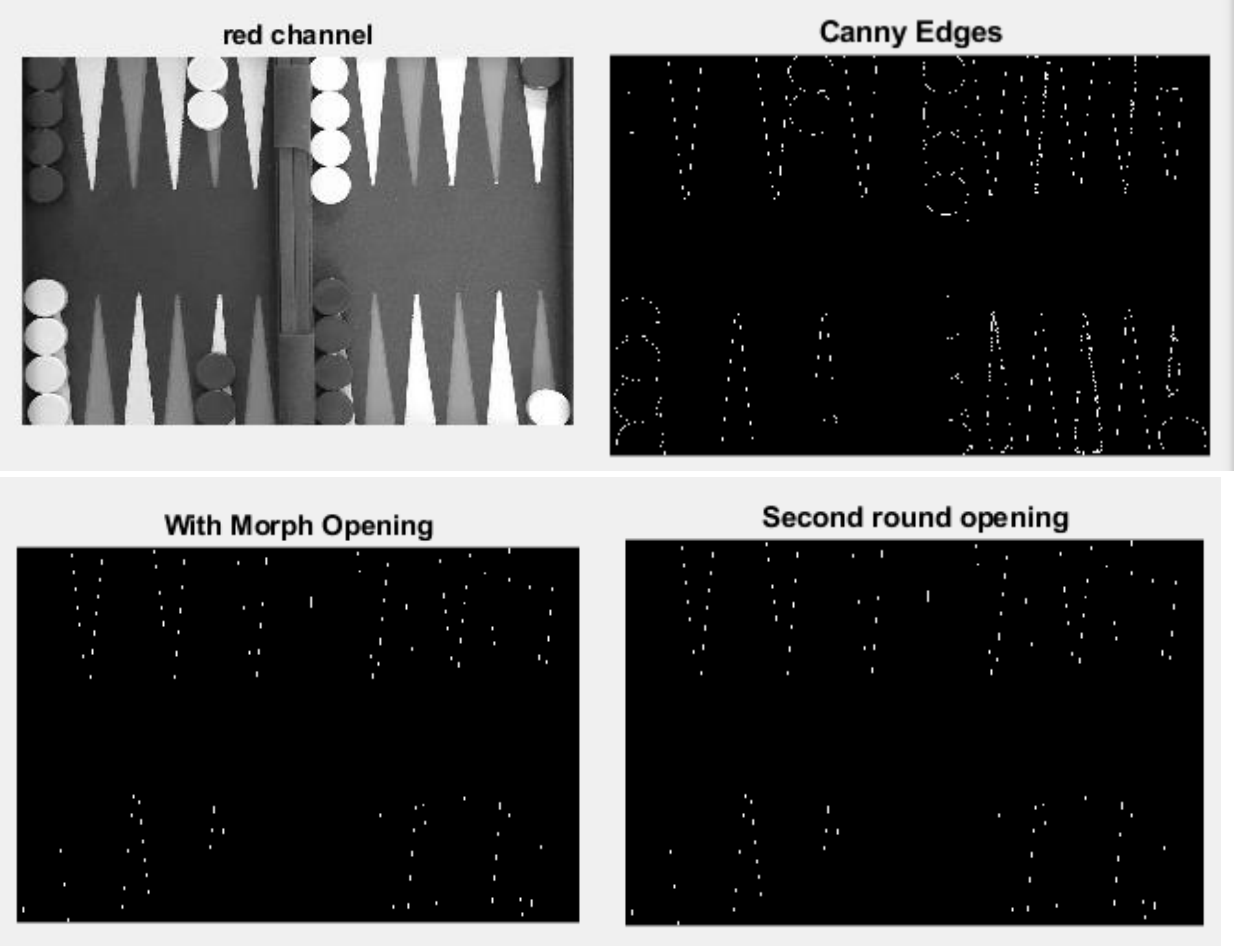
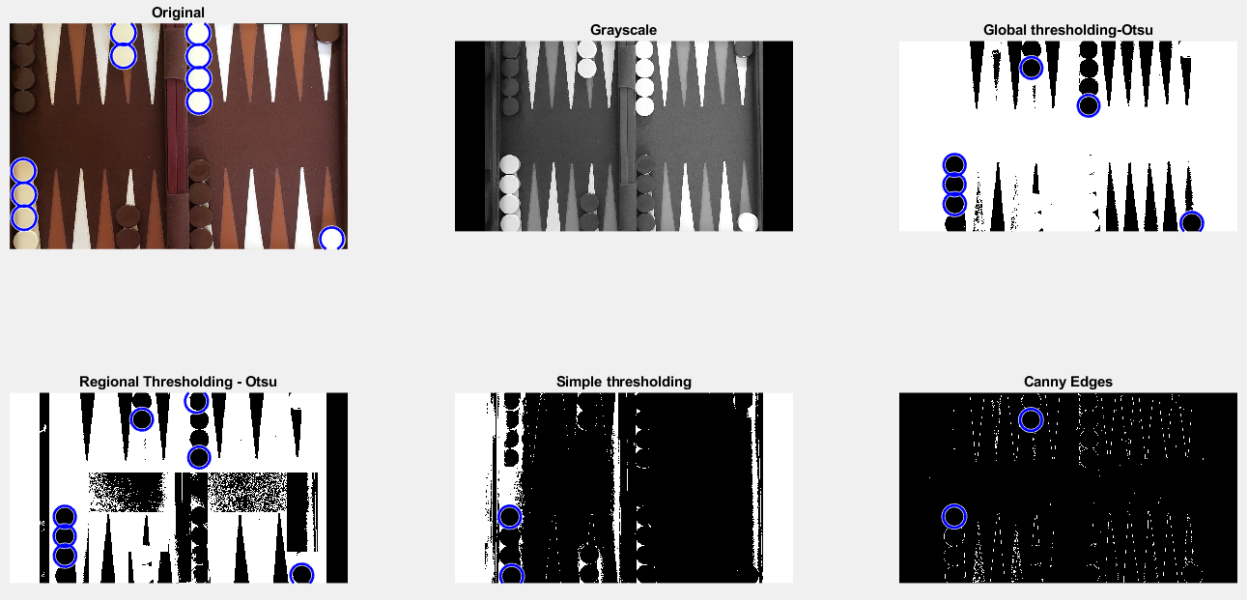


Figure 20: Morphological opening with angled strels

The Hough lines method was able to find some of the triangle edges, but certainly not the majority. More edges were found on triangles whose color was very different from the background. In the image below, for example, the white triangles have a lot more associated line segments than the brown triangles.

Figure 21: Hough lines on backgammon board

The pieces which had a similar color the background were also rarely found, just as in checkers.



Some more immediate challenges were found with using a backgammon board as opposed to the checkerboard. Because there is more variation from one backgammon board to another in terms of distances between triangles and the board dimensions, it is much more difficult to extrapolate the triangles not found with Hough line segments. The triangles are also

difficult to locate since the pieces occlude them. Backgammon pieces are placed on the board touching each other, which makes it more difficult for imfindcircles to differentiate them.

7. Discussion and Future Work

This method was highly accurate in locating board edges, but required specific image properties to succeed in finding all of the checker pieces. Ideally, such an algorithm would not be so sensitive to its input image. That being said, this sort of algorithm would likely be used to process images from a controlled environment, where the camera could be placed at an optimal angle, the board and piece colors could be chosen to make image processing more accurate, and there would not be sunlight streaking across the image.

One of the main issues in detection was the warping of pieces as a result of the image transformation. Switching the order of the algorithm by doing the transformation after the edges and pieces are detected, as suggested by the Stanford paper [3], could prevent this issue. This would prevent lack of recognition due to the pieces and image lines being warped, and reduce the effects of inconsistent user input for the four corner points.

Another aspect of the algorithm which could use modification is the use of multiple thresholding types to find the circles in the images. This was the part of the algorithm that takes the longest to run, and is a little clunky. Since simple thresholding generally allowed imfindcircles to find the most pieces, perhaps the algorithm could use only that method, but over a range of thresholds. A range of thresholds would help account for the difference in intensities between the two different colored pieces and the square colors.

Handling the lighting across images is a problem not touched upon in this report. The idea behind using regional Otsu thresholding was to reveal edges hidden by light streaks across the game board, but this turned out not to work well. That being said, this is an edge case. It is highly unlikely that in a controlled environment where such an algorithm would be used, there would be this much light streaming onto the board.

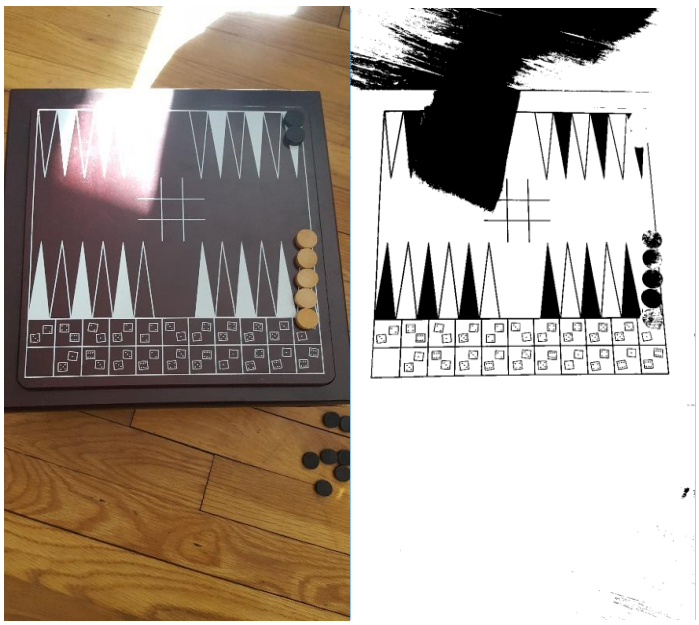


Figure 22: Regional Thresholding - Otsu's method

Although it is also an edge case, finding double stacked checker pieces would be a nice feature to add to this algorithm, since knowing the location of checker kings would be useful in deciding which move to make next.

After these improvements, this algorithm could be extended to other games. The backgammon algorithm can be modified to find the triangles on the board and the circular pieces. This analysis can also be adjusted to work for other board games, such as chess and Chinese checkers.

7. References

[1] DeepMind Technologies Limited, "The Story of AlphaGo So Far," 2018. [Online]. Available: https://deepmind.com/research/alphago/. [Accessed 8 April 2018].

[2] H. H. Makio Fukuda, "Improvement in the fun of the board game by A.R. introduction," in *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)*, Tokyo, 2013.

[3] M. K. Cheryl Danner, "Visual Chess Recognition," Stanford, 2015.

[4] T. M. K. P. M. Dennis Aprilla Christie, "Chess piece movement detection and tracking, a vision system framework for autonomous chess playing robot," in *2017 Second International Conference on Informatics and Computing (ICIC)*, Jayapura, Indonesia, 2017.