**A Vision System for Automatic State Determination of Grid Based Board Games**

Michael Bryson

Computer Science and Engineering, University of South Carolina, 29208

**Abstract.** Numerous programs have been written to play, record, and analyze various board games such as checkers and chess. However, the use of these programs often requires the use of a computer interface that may be difficult to use and detracts from the overall experience of the game. Special boards could be constructed that detect and transmit information to the computer, but a more practical solution to this problem is to create a vision system using a commonly available camera.

The proposed system can determine the state of any game played on a grid, given a correctly oriented overhead image of the board. In addition the program is given an image of each unique game piece. The vision system then consists of two main steps. First the grid size and location are determined based on the vertical and horizontal responses to edge detection. Once the location of each grid block has been determined, images representing each game piece are compared across each region containing something. The minimum total difference is then used to determine which piece best represents the contents of each block.

1 **Introduction**

There are many uses for software that operates on information about grid based board games. This includes programs which simply record each move that is made, or that analyze each move. There are also the strategic applications that can play against a user, and even beat most. Extensions could even be made to automatically broadcast games over a network. However, these programs require the use of an often cumbersome interface. One example is GNU Chess, which is shown below in Figure 1.
1.1 Vision Approach

Programs such as GNU Chess often have separate graphical front ends, but even these can take away from the experience of actually playing on a real board. This scenario is depicted in Figure 2. Special boards could be made using various sensors to determine where pieces are, but this could prove costly and would have to be specially constructed. By approaching the problem through computer vision, a more general solution can be achieved.

![Standalone Approach](image1)

**Fig 2:** Standalone approach to interaction between a user and program.

![Vision Approach](image2)

**Fig 3:** Vision approach to interaction, where the user only interacts with the physical game.
By using the vision approach depicted in Figure 3, the user only has to interact with a physical game and not the program directly. One difficulty is that each direction of interaction requires a different solution; a vision system to transfer board state information to a program, and some form of robotics to transmit computational information back onto the board. Each of these solutions would probably not interact with the program directly, but would communicate through a more general interface program developed specifically for this purpose. The problem lies in the fact that robotic systems can be quite expensive and difficult to implement. However, if only one direction of interaction is desired, this system can be omitted leaving only vision. This vision system can be more easily implemented, using only a commonly available camera.

2 Method Overview

The method presented in this paper can determine the state of any general grid based board game from a correctly oriented overhead image. One of the goals is to be able to achieve this without any prior knowledge about the actual game, such as the number of grid blocks on the board, and the size of each block. In addition to the image showing the state of the game board, a separate image will be provided to the program for each unique game piece. This information will be needed to distinguish between different contents in each grid block.

The implemented algorithm can be broken down into three main stages, shown in Figure 4. The first step is to determine the size of the grid and the location of each grid block. This is done by finding where each grid line is on the board. With this information, the program can then determine which blocks are empty and which blocks have contents. The final step is to look at each grid block that has contents, and determine which game piece appears to be present. These three steps will be discussed further in subsequent sections. After the algorithm is complete, the program will have a data structure that represents the board as a matrix of grid locations, and a label for the piece that is located in each block, if any.
2.1 Determine Location of Grid Lines

The first step is to determine how big the grid is, and where each individual grid block is located. This is done by finding the position of each vertical and horizontal grid line. Each line can be found by first looking at the horizontal response to simple edge detection, shown in Figure 5.

![Image of checkered board and horizontal response](image)

**Fig 5:** An image of a blank board is shown on the left. On the right is the horizontal response to edge detection after applying a median filter and normalization.

Edge detection is achieved through the use of a gradient image. This image is generated by convolving the original with the following kernel:

```
1  1  1
0  0  0
-1 -1 -1
```

This kernel finds the vertical derivative (horizontal response) of pixels, and also has the effect of averaging the results. This averaging is caused by the width of the kernel being three instead of one. The resulting gradient image is then processed with a median filter, to remove any speckles that may cause difficulties later in the algorithm. The result of convolution with the above kernel does not preserve the range of values, however. To correct this, the result of the median filter finally normalized so that all values fall within the range 0-255.

This process is then repeated using the transpose of the above kernel, to give the vertical response shown in Figure 6.

![Image of vertical response](image)

**Fig 6:** Vertical response to edge detection after applying a median filter and normalization.
The location of the grid lines can be easily seen in Figures 5 and 6, but the actual pixel location of each is still not known. To determine these locations, the assumption is made that the board covers at least half of the image in both the horizontal and vertical directions. Given this assumption, a scan line through the middle of the image will cross each grid line, as shown in Figure 7.

![Fig 7: Bisecting the image with a scan line will cross each of the grid lines.](image)

By scanning across this line, each location can be marked where the value of the gradient image is above a given threshold. This will identify the bright locations evident in Figures 5 and 6. This process will identify more locations than there are grid lines, however. The next step is therefore to determine which identified locations actually represent a line. This is done by tracing each location up and down, by looking for connected pixels in the gradient image which are also above the threshold. This presents difficulties, however, as shown in Figure 8.

![Fig 8: Closeup of a small section of two grid lines.](image)

Each of the grid lines is not perfectly straight, and each may contain small disconnects. To compensate for these anomalies, a small shift is allowed for each iteration of tracing the lines. This means that when tracing up, the highest value (brightest) pixel will be found within a few pixels of the previous one. Small disconnects are overcome by continuing to scan even if no appropriate pixel can be immediately found. If one is found after a few steps, then the tracing continues. However, if numerous steps fail to pick the line back up, then there is probably not actually a line at the current location (it could have mistaken the edge of a game piece as a grid line, for example).

Now that the program has identified which high points in the gradient image correspond to actual grid lines, one final processing step is required since too many lines may have been found. For example, a particularly wide grid line may produce two very close lines with this method. To correct this error, the distribution of lines is considered. If two lines are much closer together then the rest of the lines, they are merged together. The final results of determining the location of grid lines is shown in Figure 9.
Fig 9: Various results of the grid line finding algorithm are shown. The far left image shows the result of the blank board used in previous figures. The middle image shows the same board with pieces present. The far right image shows the result of using a different board. All three results correctly identify all grid lines.

2.2 Finding Grid Blocks with Contents

Now that the location of each individual grid block is known, which blocks actually have a game piece in them must be determined. This is done by using the combined horizontal and vertical gradient images, which were generated in the previous step. This gradient image of the board with pieces is shown as Figure 10.

Fig 10: Combined horizontal and vertical gradient images of game board which contains pieces.

By looking at Figure 10 and noting that brighter pixels have a higher value, it is evident that blocks containing a game piece have a higher overall sum of gradient values than the homogeneous empty blocks. The distribution of gradient sums is plotted in Figure 11.

Fig 11: Plot of the sorted gradient sums of all grid blocks.

From this plot is can be seen that there is a distinct jump in values at a total sum of around 200000. This divides the distribution into two main groups, those below the
jump and those above it. Knowing that a block with contents has a higher gradient sum than an empty one, we can conclude that all blocks with sum greater than 200000 (in this case) contain an object, while those below are empty.

2.2 Calculate Best Fitting Game Piece

Knowing the location of each block that actually contains a game piece, the program can complete the final step of determining which piece each actually is. This is done by using the provided images as shown in Figure 12.

Fig 12: Two provided images showing the distinct game pieces. The left image is labeled ‘r’ while the right image is labeled ‘b’.

The assumption is made that each grid block is larger that the game piece images. Therefore each game piece image is iterated over each possible location in each block containing a piece. During each iteration, the absolute difference is calculated between the game piece image and the current position in the grid block. This is done for the red, green, and blue channels, and the minimum total absolute difference is chosen for each game piece image in each block. This determines the location in each block where each game piece image best fits. Then the minimum of these values (one for each game piece image) determines which of the actual pieces best fits.

3 Results

Two sample outputs from this method are shown below in Figure 13. This shows a text representation of the internal data structure obtained from each image, however output could easily be formatted differently to communicate with the interface program originally shown in Figure 3. In each case, this method correctly determined the size of each grid, which grid blocks were empty, and which piece is located in each block.
3 Issues and Difficulties

While the example game states shown above were correctly determined, this method still has some limitations. One limitation is that each game piece must be distinctly different from the background of each grid block. For example, consider if a black game piece was positioned on a black grid block in the examples above. It would be almost impossible to see it. Two other difficulties lie in specular surfaces and the assumption that the overhead image was taken under orthographic projection.

Fig 14: Image demonstrating difficulties with reflections and projection.
Figure 14 shows these two issues. The reflection in the middle can hide some of the grid lines, which would not show up in the gradient image. This could be addressed by scanning across the image in multiple locations, instead of just across the middle. By doing this, the possibility of finding each line would be increased, even if not all lines were found by each scan.

This method also assumes that the overhead image was captured under orthographic projection. This is not the case, however, as demonstrated by the tall chess pieces in Figure 14. Instead of appearing only in on particular grid block, they also reach into neighboring blocks, which could present problems in the first two steps of the algorithm. The appearance of each game piece also changes depending on where is it located on the board, which could cause difficulties in the final step of determining the best fitting game piece.

4 Conclusion

The method presented can accurately determine the state of a grid based board game in simple cases, and with improvements could be effective in more general cases. This would allow for some interesting extensions. One possibility would be to correct for rotated board images, using the grid as a guide. Edge gradient kernels could be applied at different rotations to find the two with strongest response. These strongest directions (presumably 90 degrees apart) could then be used to rotate the image to the orientation assumed by the rest of the algorithm.

This approach could also be applied to a constant video stream. It would be undesirable to run the algorithm for every frame of the video, so instead it could again use the grid to determine when a change has been made. It would wait for an occlusion to appear, which would correlate with a hand entering the scene to move a piece. When the hand leaves the scene the grid would become unobstructed, and the new state of the board could be calculated.