# An Application for the Game of Go: Automatic Live Go Recording and Searchable Go Database

Ananta Srisuphab, Piyanuch Silapachote
Tanapoj Chaivanichanan, Wiracha Ratanapairojkul, Wisallaya Porncharoensub
Faculty of Information and Communication Technology, Mahidol University
Nakhon Pathom, Thailand
{ananta.sri, piyanuch.sil}@mahidol.ac.th, {u5188018, u5188131, u5188115}@student.mahidol.ac.th

***Abstract*—The key to become a skillful Go player is to learn from experience, analyzing and reflecting on previously played games. Rich in strategy, the game of Go requires a player to recognize winning or losing configurations of game pieces on the board. Knowing how to respond is essential, and training to do so requires an extensive study of various different tactics used by a diverse group of players, especially professionals.**

**The benefit of recording Go games has undeniably been realized. Hand-recording has been attempted but it is neither efficient nor practical. Described in this work is an automatic Computer Vision system to record Go games - AutoGoRecorder. Incorporating advanced technologies, AutoGoRecorder automatically records a live Go game in real time. Additionally, it maintains a dynamic database of Go games that is easily accessible and searchable. It is an invaluable tool for any Go organization to record a tournament. Individuals can use it to review certain games or to familiarize themselves with strategies to respond to particular configurations of Go stones. Coaches and tutors can use it as an teaching aid to give suggestions or comments, reading the game to amateur players.**

## I. Introduction

The game of Go is a two-player board game most popular in China, Japan, and Korea. It has since gained worldwide attention including in Thailand. Go is a strategic game that involves mathematics, psychology, philosophy, and the art of negotiation. It is a territorial game where each player controls either white or black stones. Two players alternately place a stone on a vacant intersection position of a 19-by-19 game board. The goal is to protect and expand one's own territory and the player with the most territory wins the game.

Configurations of Go stones at a particular position on the Go board could reveal a winning or a losing preposition of a player. Making an appropriate sequence of moves would ensure territorial expansion; wrong moves may reverse the outcome. Accomplished players recognize and respond to these patterns ingeniously. Undoubtedly, learning to appreciate these patterns is crucial for improving a Go player's skill.

Since patterns repeatedly arise in many Go games, it becomes beneficial to keep a detailed record of the related Go game, which includes every move of both players and a complete state of the board at every turn. Players can then review their games, and advanced players can make recommendations to intermediate or amateur players.

Currently, Go games are recorded by hand which poses significant limitations. Because of the time and the human labor needed, the Go Association of Thailand only records professional games even though it organizes a number of local and national tournaments for all levels of players throughout the year. Additionally, analysis of hand-recorded Go games is cumbersome. The constraints lie not only in the difficulty of understanding complex hand-recorded notations and representations, but also the difficulty of accessing the collection of already recorded games.

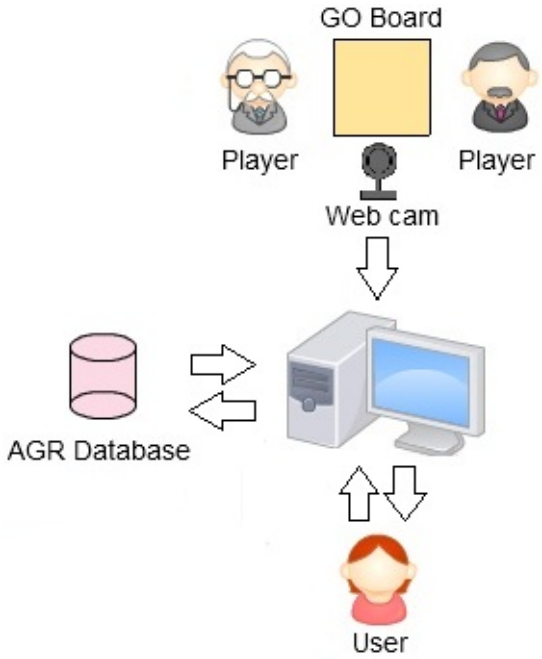


Figure 1. AutoGoRecorder maintains a database of recorded Go games and provides a friendly graphical user interface. Through a webcam, players can record live games. Other users access the database to review individual games or to study pros and cons of various tactics for the game of Go.

Therefore, the AutoGoRecorder (AGR) system, whose structure is shown in Figure 1, was developed. It automates the recording and maintains a database of Go games. It is a portable system that requires minimal setup. Video is captured through a webcam connected to a personal computer. Each game state is extracted and stored in a database using the Smart Game Format (SGF) [1]. The database contains live games recorded by the system as well as games or partially

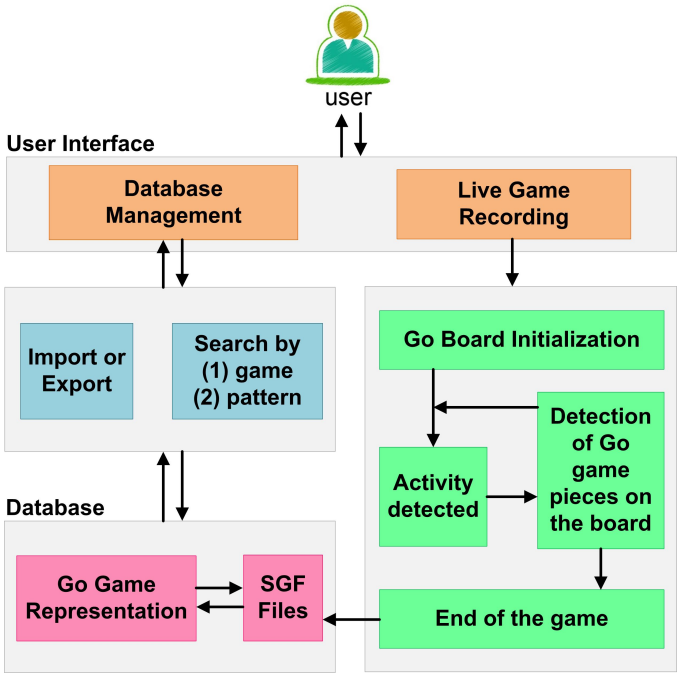


Figure 2. AutoGoRecorder system architecture.

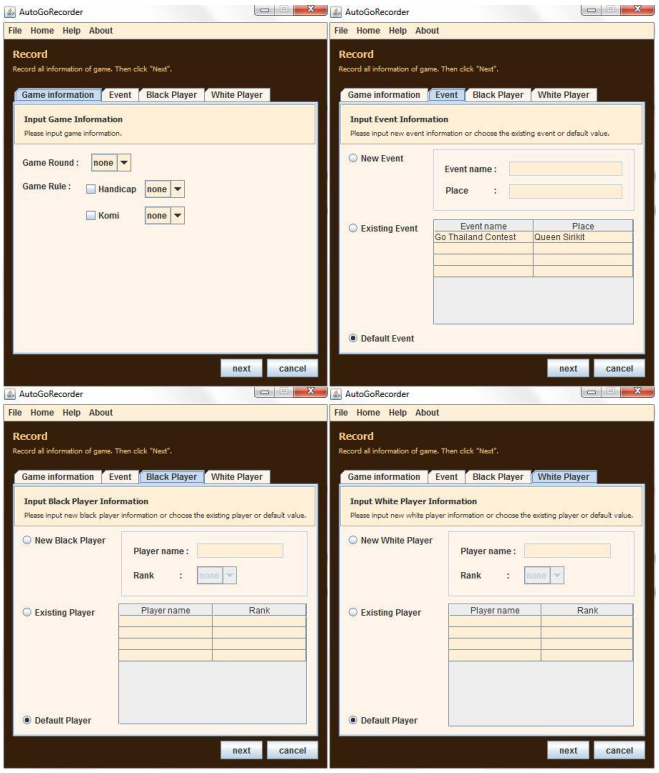


Figure 3. User interface of the AGR live game recorder. Players enter their names, their rankings, and specific information of the current game.

played games manually imported by a user. The search engine enables querying of games according to certain criteria, such as a specific tournament. When a user queries a particular configuration of a Go board, games containing such configuration are returned. This allows a user to compare and contrast the different moves from various different games leading up to and responding to a particular pattern.

Storing Go games in the Smart Game Format makes AGR compatible with many other computer-recorded two-player board games: SGF supports Chess, Backgammon, Reversi/Othello, to name a few. Understanding a commonly used format tends to be straightforward for a user. It is both simple and practical for importing, exporting, or exchanging recorded games with AGR. Nevertheless, it is not only inefficient but also insufficient to draw the state of a Go board from the SGF file alone. SGF accounts for every game piece that is placed on the board at every turn, but it fails to record game pieces that are removed from the board when they are captured. To overcome this shortcoming, a compact text-only representation of a Go game is proposed and stored alongside the SGF file. This new representation further enhances querying and searching the AGR database.

## II. Related Works

The intellectual complexity of the game of Go captivates not only Go players but also researchers in multiple disciplines. Neurobiologists and psychologists have studied brain activities of subjects engaging in a Go game using a functional MRI [5]. Mathematicians, computer scientists, and engineers have attempted to model, analyze, and predict Go moves [2], [6], [12], [13], [14]. Others have automatically extracted Go games from still images and video streams [9], [10], [11].

What distinguishes the AutoGoRecorder is its simplicity and portability. Different from other systems, AGR is a powerful application that integrates live recording of a Go game and a dynamic database that allows querying both by game and by Go patterns. A novel compact string representation of a Go game is proposed, enhancing the efficiency of the search.

## III. System Architecture

Essential elements that make AGR attractive include portability and a user-friendly interface. Setting up AGR involves a personal computer and a webcam that is positioned above a Go board. The top-level graphical user interface allows a user to choose one of the two operating modes: recording of live Go games and database use. Figure 2 outlines the architecture of the AGR system: its components and their interconnections.

In the recorder (shown in Figure 3), a user enters specific information to identify the current Go game, which will be saved into the SGF file along with the game being recorded. This includes names and Go rankings of both players, as well as tournament or event name. Players who play a game with handicap or komi may indicate the level of compensation they have agreed upon. After setup, a player starts the software that automatically records every time a movement is detected. The user can also pause or stop the recording. Section IV describes the technical details underlying this automatic recording phase. Related image processing techniques are implemented using the Open Source Computer Vision (OpenCV) library [3].

Figure 4. User interface of the AGR database. Searching by game is shown in the top left image, and searching by patterns in the top right image. A user specifies a search pattern in the virtual board (bottom left). AGR returns a list of every game containing a desired pattern (bottom right). A user may preview the state of a selected game (Figure 5).

Through the database (shown in Figure 4-5), a user can import or export selected Go games. There are two search options: by game and by pattern (particular configurations of Go stones on the board). To review a specific game, a user must specify information about the game, e.g. a tournament or an event name or information about either players. To study the tactics used for a particular pattern, a user must specify positions of black and white game pieces on the board. A simple user interface is provided where a user simply clicks on a virtual Go board to indicate the stone positions. Section V describes the AGR database, its components and its functionality. It is managed using the open source database MySQL [8].

## IV. Automatic Game Recording

After a user starts a recording, the first step is to initialize the board (explained in Section IV-A), mapping a location of every intersection point. This is a one-time process that identifies where a game piece will be placed during the game and thus a point where AGR will look for a Go stone during the recording. Next, the game is automatically recorded. Any activity above the board is continuously tracked as every turn is being recorded. This step is detailed in Section IV-B.

### A. Go Board Detection

The AGR software detects the 19x19 board and all 361 crossing positions using the steps described next and illustrated

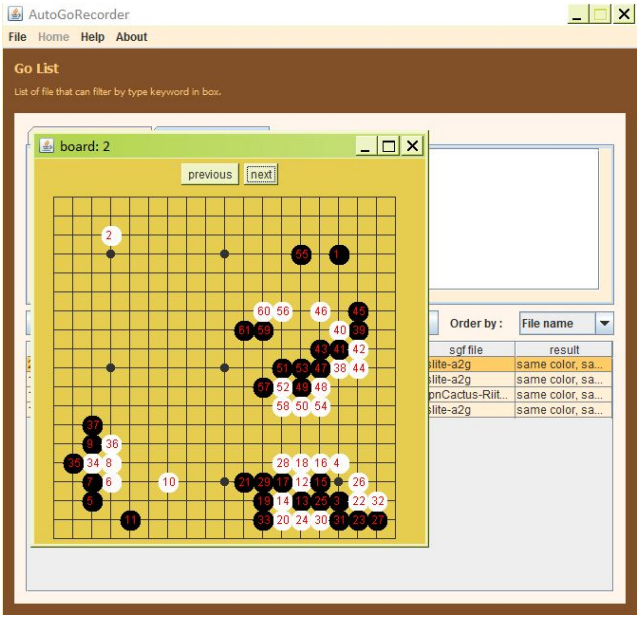


Figure 5. A preview of a selected game when searching by pattern.

in Figure 6. First, a Canny edge detector [4] is applied on an image captured by the camera. Then, a Hough transform [7] is used to connect and to refine those pixels identified as being part of some edge, forming well-defined lines. Any pixel where these lines cross are hypothesized as an intersection of the Go board grid, and will be referred to as a point.

This process detects most of the 361 intersection points on the Go board, however some of the hypothesized points are not be accurate, especially those along the borders. To fill in missing points and to fix erroneous points, the hypothesized points are mapped and adjusted against a virtually generated grid of a game board. AGR first determines a circumscribed rectangle (the smallest bounding box that contains all hypothesized points), then the four corners and the center of the Go board. Correspondingly, a virtual 19x19 board game is drawn. This virtual grid serves as a guide and will match well with most of the hypothesized points. Missing points are created by interpolation and adjustments are made to misaligned points.

The above rectification step is necessary since it is very important to get accurate detections of the 361 intersections of the grid of the Go board. This is because the Go board is detected only once and all subsequent stone detections rely on these points.

### B. Go Gameplay Tracking and Go Stone Detection

The AutoGoRecorder continuously monitors the Go board. Using gradient differences of motion history images, it detects any activity across the board. Activity is not limited to a turn made by a player when a stone is placed on a board. There are also other changes such as a removal of a dead stone or even a false alarm when the board is unchanged but a player passes his hand over the board.

Every detected motion triggers a check for any alteration of stone configuration on the board. If a change has been

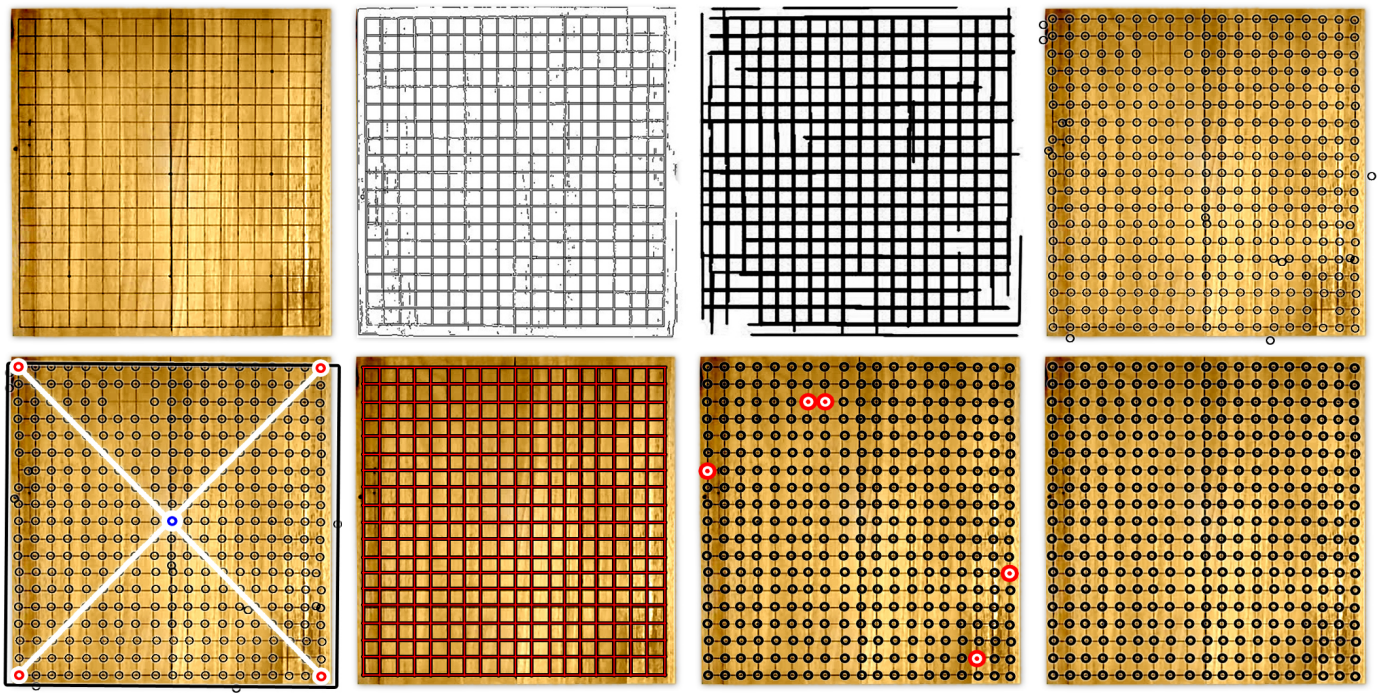

Figure 6. Steps of the Go board detection (top row from left to right, followed by bottom row left to right). (1) The board is captured by a webcam. (2) Result of Canny edge operator. (3) After applying a Hough transform. (4) Hypothesized intersection points. (5) Circumscribed rectangle and locations of the four corners and the center. (6) Virtual 19x19 grid. (7) Rectification of hypothesized intersection points. (8) Final result.

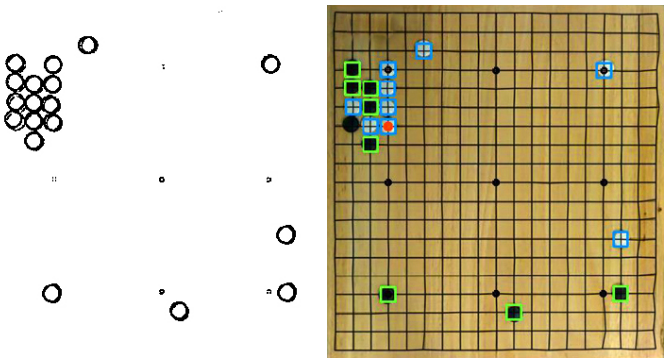

Figure 7. Example detection of Go stones and their configuration.

made, then the corresponding event is added to the game and subsequently to the SGF file.

The detection of Go stones and their configurations, depicted in Figure 7, involves creating a circular boundary the size of a stone (a game piece) around every intersection point of the detected 19x19 grid on the Go game board. The average color value around every point inside the designated circular boundary is calculated. This value is used to determine the state of a point: it may be empty, have a black stone on it, or have a white stone on it. To model colors, AGR employs the HSV color model. HSV is chosen because it enables good distinction between shades of black and white.

At the end of the game, or when the user stops the recording, AGR generates a SGF file of the game and imports it into the Go game database.

## V. Database Management

There are two modes of operation involving the database: search for specific games and search for specific patterns.

### A. Game Search

To search for a specific game, the user enters at least one of the following information: an event or a tournament name, when and where a game was played, a player name and ranking, and if any additional rules were applied. The database returns an SGF file of all the games that satisfy the criteria.

### B. Pattern Search

A pattern refers to a particular configuration of Go stones, either black or white or both, positioned on a Go board. Many of these configurations carry significance with the players. Making a flawless successive move increases the chance of winning: defending a player's own territory or attacking to secure more territory. Making incongruous moves allows the opponent to surround and to capture stones, losing control of an area. Therefore studying stone configurations, corresponding subsequent moves, and game outcomes is very important to mastery of the game.

Once a search is initiated, a user clicks to position black and white game pieces on a virtual board provided. This user-specified pattern is matched against the database (see an example in Figure 8) and all the games during which the pattern had been detected are returned in SGF files. Afterwards the user can examine the sequence played in each game, analyze it, adapt successful strategies, recognize unfavorable tactics, and consequently improve one's skill level.

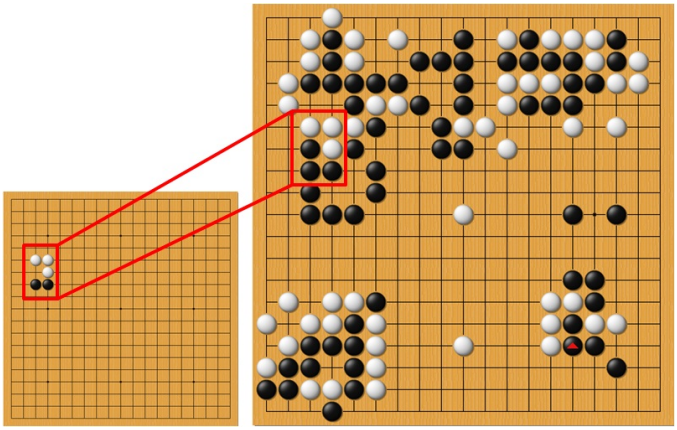

Figure 8. An example of a pattern search. A user inputs the pattern on the left. It is matched with the stored games and one matching board (on the right) is found. The SGF file of the game to which this board belongs is returned.

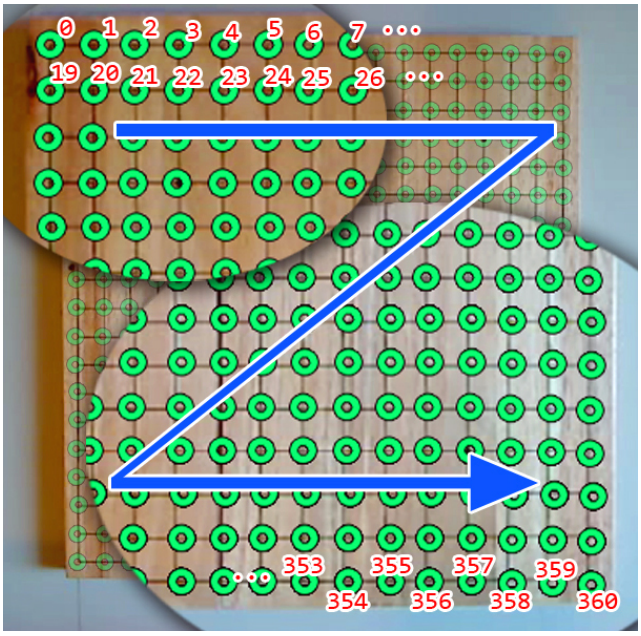


Figure 9. Indexing of every point, i.e. intersection position, on a Go board.

### *A String Representation of the Game of Go*

In order to facilitate fast search for Go game patterns, a compact representation of a Go game is developed. When a game is saved or imported, both its SGF file and a corresponding compact representation of the game are added to the database.

A Go game is represented in compact form by three strings that each have 361 elements. Each element represents one point, that is one intersection position of the Go board on which a Go stone is placed. The indexing of the 361 elements follows from the upper left corner of the 19x19 grid to the upper right corner and from the top to the bottom row, illustrated in Figure 9.

One of the three strings encodes the state of each point. For each point, the corresponding value is zero if a point remains empty throughout the game. It is one or two if at any time during the game it is occupied by a black or a white stone, respectively. Otherwise, it has a value of three indicating that it has been occupied by a black stone at some time during the game and by a white stone at some other time during the same game, this includes the possibility of multiple switchovers.

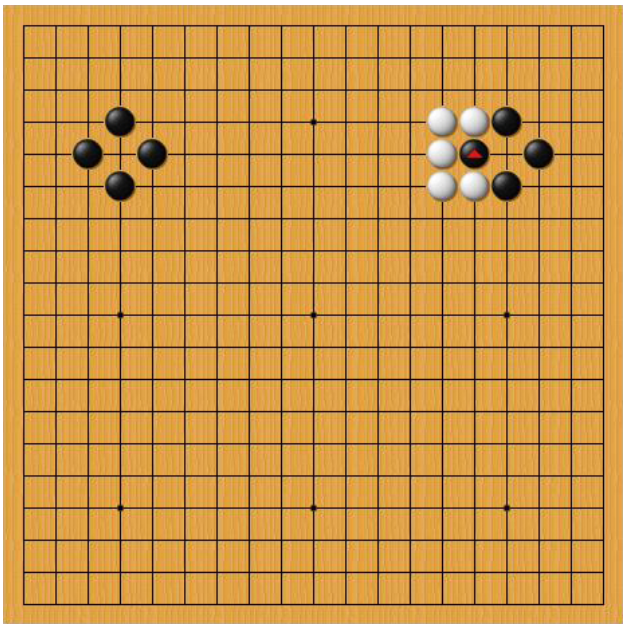

Figure 10. An example of a Go game that has been played for 17 turns; its corresponding three-string representation is shown in Figure 11.

In the game of Go, once a stone is captured, called a dead stone, it may be removed from the board. The point becomes empty and later in the game another game piece may be replaced here. A history of how game pieces occupy each point on the board is encoded in the other two of the three strings; one for black stones and the other for white stones. More specifically, a string for black stones lists for each point a range of turn numbers during which a point is occupied by a black stone. It uses a comma to separate neighboring points and uses a semicolon to separate ranges of turn numbers (the case when a black stone is placed at a point where there had been black stone that was captured). The same principle is used to create the string for white stones.

An example of a Go game that has been played for 17 turns is shown in Figure 10 and its corresponding three-string representation in Figure 11.

### *Database Query and Pattern Matching*

When querying a user-specified pattern, its compact representation is generated. The first string encoding the status of all points is compared against the status strings in the database. As described before, this string indicates an accumulated status throughout the game. That is, once a black game piece is placed on a point, the status of this point changes to one. It may change to three if a white game piece is later placed here, but it never changes back to zero (even though a stone may be captured and the point becomes vacant later).

| index | 0 | ... | 60 | ... | 70 | 71 | 72 | ... | 78 | 79 | 80 | ... | 89 | 90 | 91 | 92 | ... | 98 | ... | 108 | 109 | 110 | ... | 360 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| status | 0 | | 1 | | 2 | 2 | 1 | | 1 | 2 | 1 | | 2 | 1 | 2 | 1 | | 1 | | 2 | 2 | 1 | | 0 |
| black | | | 3-17; | | | | 1-17; | | 15-17; | | 11-17; | | | 5-13;17-17; | | 7-17; | | 13-17; | | | | 9-17; | | |
| white | | | | | 16-17; | 12-17; | | | | 8-14; | | | 6-17; | | 2-8;14-16; | | | | | 4-17; | 10-17; | | | |

Figure 11. The proposed compact representation of a game of Go. The top row is an index shared by all three strings; the second to the fourth row are point status, black stone, and white stone strings, respectively. Taking point number 90 as an example, status is 1 indicating that at some time during the game it is occupied by a black game piece. The string of black stones further clarifies that this happened at turns 5 through 13 and again at turn 17. The gap between turns 13 and 17 reveals a capture and a re-placement of a black stone.

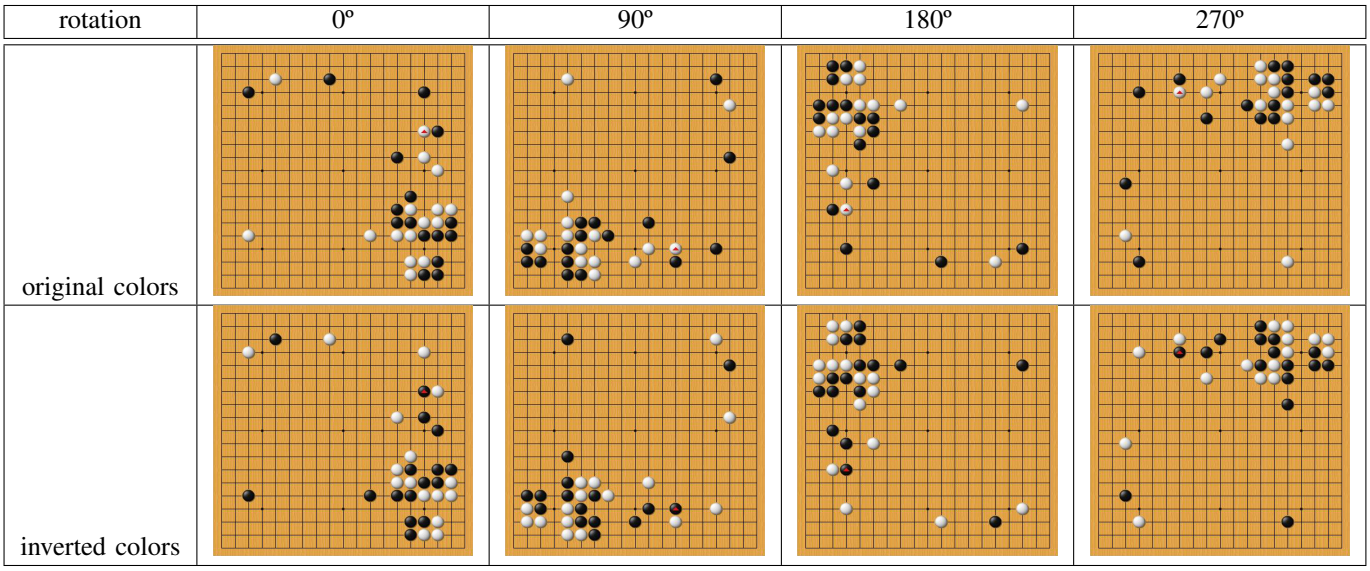


Figure 12. Equivalent patterns of Go stone configurations causes by a rotation of the Go board and the interchanging of black and white.

Matching these strings does not necessarily mean an exact pattern match, but it narrows down the search space. Refining the search is accomplished using the other two strings of black and white stones. The turn numbers are checked to make sure that all the game pieces occupy the specified locations at the specified time.

### *Equivalent and Symmetric Patterns*

When matching a pattern, not only are the exact colors and positions of a user-specified pattern matched, but also the equivalent patterns when the board is rotated at 90, 180, and 270 degrees. Additionally the colors of the stones can be swapped between black and white. The user interface allows specifying these search options. Figure 12 lists eight instances of equivalent Go patterns.

## VI. Conclusion

The AutoGoRecorder has been developed and successfully tested. It automates the recording of live Go games without distracting the players. With a minimum requirement of a personal laptop and a small webcam, it is a portable system that can be used at home, at a local event, or at a national tournament. AutoGoRecorder manages a database of Go games. It keeps a record of both live and manually imported games. The standard Smart Game Format supports an easy external file exchange. Using a newly developed compact representation of a Go game results in efficient search. The software provides easy access to recorded games containing desired stone configurations, a functionality that enables players to improve their skills and ultimately advance their Go rankings.

## References


[1] "Sensei's library." [Online]. Available: http://senseis.xmp.net/

[2] N. Araki, K. Yoshida, Y. Tsuruoka, and J. Tsujii, "Move prediction in Go with the maximum entropy method," in *Machine Learning*, 2007.

[3] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*. Cambridge, MA: O'Reilly, 2008.

[4] J. F. Canny, "A computational approach to edge detection," *IEEE Transaction on PAMI*, vol. 8, no. 6, pp. 679–698, November 1986.

[5] X. Chen and et. al., "A functional MRI study of high-level cognition: II. the game of Go," *Cognitive Brain Research*, vol. 16, no. 1, 2003.

[6] R. Coulom, "Computing Elo ratings of move patterns in the game of go," in *ICGA Computer Games Workshop*, June 2007.

[7] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Comm. of the ACM*, vol. 15, no. 1, 1972.

[8] S. Hinz and et. al., "MySQL 5.5 reference manual," 2011. [Online]. Available: http://dev.mysql.com/doc/refman/5.5/en/

[9] T. Hirsimaki, "Extracting go game positions from photographs," Helsinki University of Technology, Tech. Rep., 2005.

[10] S. Scher, R. Crabb, and J. Davis, "Making real games virtual: Tracking board game pieces," in *ICPR*, 2008.

[11] A. K. Seewald, "Automatic extraction of go game positions from images: a multi-strategical approach to constrained multi-object recognition," *Appl. Artif. Intell.*, vol. 24, no. 3, pp. 233–252, March 2010.

[12] D. Stern, T. Graepel, and D. J. C. Mackay, "Modelling uncertainty in the game of Go," in *NIPS*, 2004.

[13] D. Stern, R. Herbrich, and T. Graepel, "Bayesian pattern ranking for move prediction in the game of Go," in *ICML*, 2006.

[14] S.-J. Yen, T.-N. Yang, J.-C. Chen, and S.-C. Hsu, "Pattern matching in Go game records," in *ICICIC*, 2007.