

A Tangible Interface to Computerized Chess

David J. Meppelink
dmeppeli@cs.uml.edu

Fred Martin
fredm@cs.uml.edu

University of Massachusetts Lowell
Department of Computer Science

May 12, 2003

Abstract

CHESTER is a tangible interface to computerized chess. With it, a person can use a physical chess board to play against a computer or an opponent connected through the Internet. This paper describes its hardware and software components and its most important design ideas.

1 Introduction

CHESTER is a tangible interface [7] for playing chess against a computer or a geographically remote opponent. It sees the moves its human opponent makes on a physical board and responds by moving its own pieces. Its hardware consists of a chess board and pieces, a robotic arm to move those pieces, a vision sensor to view the game board, and a personal computer. The PC controls the physical interface hardware and runs a chess engine that plans its game moves. The chess engine can also be replaced by a connection to a Internet Chess Server so that the user can play against a human or computer opponent anywhere in the world.

People have long held the desire to play chess against a machine. “The Turk” was unveiled in 1770 [10] and led a series of fraudulent chess automatons that continued through 1930 [15]. Of course, the technology of that time could not produce a machine that could actually play chess; the automatons were operated by a hidden human chess master. But digital computer technology has produced many commercial and research products that not only play chess, they play on par with chess masters [14]. The focus of most commercial chess products is on playing the game well, rather than on alternatives to a graphical user interface.

The idea of a “chess robot” is a popular topic for a broad range of researchers from computer science [2] [9] to hydraulics [1] to hobbyists [3]. This work parallels those efforts. Our goals

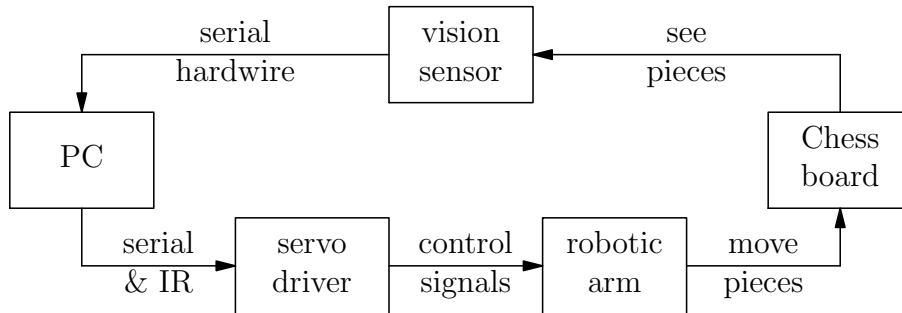


Figure 1: Hardware Components

in building CHESSTER were to gain introductory experience in fundamental robotic topics and to demonstrate the use of pre-existing hardware and software components to quickly develop a complete system.

This paper is an overview of CHESSTER’s hardware and software components and discusses some of its key design ideas.

2 Hardware Components

CHESSTER has the following hardware components, as shown in figure 1:

1. A **chess board** and 32 pieces. The board and pieces are fairly small¹ to accomodate the size of the robotic arm. We painted the top of each piece red so that the vision sensor could distinguish them from the board itself.
2. A Lynx 5 **robotic arm** [11]. This is an inexpensive kit with five degrees of freedom² driven by servo motors. We replaced the original gripper because it was unable to pick up the chess pieces. Our replacement used foam-covered flat tongs that could fit between the pieces on the closely spaced board.
3. A **servo controller** consisting of a Handy Cricket [6] and a servo controller bus device. The bus device is a pre-production prototype that can generate control signals for up to eight servos. The two are connected by the Handy Cricket Expansion Bus.
4. A CMUcam **vision sensor** [4] mounted twelve inches above the center of the chess board. At that distance, the board fills the camera’s field of view, leaving narrow margins at the sides.
5. A **personal computer** running Microsoft Windows XP™. The PC uses one serial port to communicate with the Cricket (via a Serial-IR interface) and a second serial port for the CMUcam (via a hardware connection).

¹The board is six inches by six inches and the pieces are $\frac{5}{8}$ to $\frac{7}{8}$ inch tall.

²The arm can move on five axes: base rotation, shoulder, elbow, wrist and gripper.

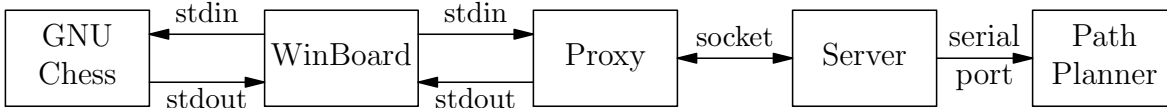


Figure 2: Software Components

3 Software Components

Figure 2 shows the software components:

1. **GNU Chess** [5] is an open source chess engine. It is called an “engine” because it focuses on the strategy of playing chess but has a primitive console interface. It requires that its opponent’s moves be expressed in coordinate algebraic notation and it responds with its own moves using the same notation.
2. **WinBoard** [13] is an open source graphical user interface for computerized chess. It renders the game graphically and permits either side of the game to be played by the computer user, a chess engine, or a remote player connected through an Internet Chess Server. WinBoard communicates with chess engines and remote players using the Chess Engine Communication Protocol [12], a simple protocol based on the GNU Chess console interface. We use the binary distribution of WinBoard, which includes the binary for GNU Chess.

The key feature of WinBoard for this application is its ability to coordinate the communication between two chess engines: GNU Chess on one side and the CHESSTER physical interface (via the Proxy, etc.) on the other side.

3. The **Proxy** passes data between Winboard and the Server. It does not perform interpretation of that data and it does not maintain any state. WinBoard starts this program as a separate process and communicates with it through operating system pipes. The Proxy communicates with the Server through a network socket. We wrote this component in Java to run on the PC.
4. The **Server** translates physical moves to and from the notation GNU Chess uses. This translation is handled through the semantic layers described in section 4. We wrote this component in Java to run on the PC. We used the Java Communications API [8] for serial port access.
5. The **Path Planner** converts the robotic arm positions produced by the Server into linear movements of the robotic arm. It coordinates the movement of the servos by simultaneously stepping each of them through an equal number of intermediate positions. Without this interpolation, the constant speed of the servos combined with disproportionate position changes caused the arm to move erratically.

4 Semantic Layering

4.1 From Chess Engine to Physical Board

We split the task of moving chess pieces into four semantic layers. Each layer is responsible for translating its model of the task into a representation that the next layer can process, until, at the lowest level, the arm moves the chess piece.

At the highest layer, GNU Chess has a model of all the pieces on the board and selects a move based on that model. It passes that move (through the Proxy) to the Server in coordinate algebraic notation. For example, it could start a game with the move “advance king’s pawn one space” by passing “e2e3” to the Server. Table 1 follows this example through each of the semantic layers.

The Server converts the move from coordinate algebraic notation into a sequence of arm positions that define the waypoints along the path of moving the chess piece. For example, the opening move “e2e3” would be translated into the sequence of positions shown in row 3 of table 1. The Server does not maintain the full game state; it only tracks which squares are occupied so that it can generate capture sequences. If the destination square had been occupied, the sequence would have been preceded by a sequence to remove the captured piece from the board.

The Server sends each arm position to the servo controller as a record containing an index value for each of the five arm axes: base, shoulder, elbow, wrist and grip³. Table 1 row 4 shows the records for the first three arm positions.

The Path Planner moves the arm along a linear path to the position given by the record. To do this, the planner needs to maintain only the current arm position; it does not need any information maintained by the higher layers, such as the game state or which squares are occupied.

4.2 From Physical Board to Chess Engine

We used a similar set of semantic layers to translate physical chess piece moves into coordinate algebraic notation.

At the lowest level, the CMUcam produces an 80×143 array of color pixels. The Server can load a single frame from the camera in 4.8 seconds⁴.

The Server converts the image pixels into an 8×8 occupancy array by examining 64 disjoint regions⁵. If more than 20% of the pixels in a region are predominately red, then the square is considered to be occupied. Each value in the array is a boolean value that is true only if

³We determined the index values for each of the arm positions empirically.

⁴A single frame is 34,402 bytes long, transmitted over a 115,200 baud serial connection.

⁵Each region is 6×14 pixels large and is centered on a board square.

Layer	Component	Move Representation
1	GNU Chess	advance king's pawn one space
2	GNU Chess and Server	e2e3
3	Server	above ^a e2, open gripper at ^b e2, close gripper above e2 above e3 at e3, open gripper above e3
4	Server and Path Planner	134 ^c , 193, 214, 51, 202 134, 171, 221, 75, 202 134, 171, 221, 75, 260 134, 193, 214, 51, 260 ...

^a“above” means position the gripper directly above a piece that occupies the given square.

^b“at” means position the gripper so that surrounds a piece that occupies the square.

^cValues are in the range of 0 to 318, as dictated by the servo controller.

Table 1: An example of semantic layering

the corresponding square is occupied.

To translate the occupancy array into a chess move, the Server compares the current occupancy to the previous occupancy. The square that changes from occupied to unoccupied is the source; the one that changes from unoccupied to occupied is the target. Given that information, it is trivial to create the move in the proper notation.

Captures require three occupancy snapshots: the original (before the move starts), the intermediate (with the captured piece off the board), and the final (with the capturing piece in its final position). Although the Server could recognize that a capture took place with only two snapshots, it cannot determine the target of the capture without the intermediate snapshot. Once the Server has the source and target squares, it can produce the move using the same notation as for regular moves.

When GNU Chess receives the move from the Server (through the Proxy), it converts it to the higher level semantic of the game play and calculates its next move.

5 User Interactions

The Server needs a way to quickly tell when the user completes a move. It cannot poll the image until the occupancy array stabilizes because of the CMUcam's slow frame rate. Polling requires at least two identical image frames, which would result in an unacceptable latency of 10 to 15 seconds. Instead, the Server uses a special feature⁶ of the CMUcam to monitor the average color in a small area (2×1 pixels) of the image. The user moves a yellow brick into that area when he has completed a move. When the Server sees a significant color change in the average color, it proceeds to look for the move (see section 4.2).

During demonstrations, people often found the yellow brick distracting or confusing. Most preferred it to using the computer keyboard. Future development should focus on a more natural interaction.

6 Conclusion

We have accomplished both of our goals for this project: We were introduced to robotics topics such as microcontrollers, arm path planning, image processing, inter-machine communication, and tangible interfaces. And we completed a sophisticated prototype in approximately one hundred hours (spread over five weeks). That was possible only because we used pre-existing components to implement large portions of the system and were able to focus on the elements that made CHESSTER unique, such as its semantic layering for path planning and image processing.

References

- [1] V. Buldyzhov and K. Brezhnev.
Hydraulic chess playing manipulator.
<http://www.vb.mksat.net/cr/manipulator.htm>.
- [2] Nasron Cheong, Peter Cooney, Stuart Day, Evelyn Melnichouk, and Andrea Sommer.
Dynatutor 5 axis arm robot with chess playing software.
<http://members.lycos.co.uk/nasroncheong/dynatutor/>.
- [3] Andy Clapham.
Lego mindstorms chess.
<http://www.artilect.co.uk/lego/default.asp?page=Chess>.
- [4] Cmuacam vision sensor home page.
<http://www-2.cs.cmu.edu/~cmucam/>.
- [5] GNU Chess home page.
<http://www.gnu.org/software/chess/>.

⁶Specifically, a combination of the “set window” and “get mean” commands.

- [6] Handy cricket home page.
<http://www.handyboard.com/cricket/>.
- [7] Hiroshi Ishii and Brygg Ullmer.
Tangible bits: Towards seamless interfaces between people, bits and atoms.
In *CHI*, pages 234-241, 1997.
- [8] Java communications API home page.
<http://java.sun.com/products/javacomm>.
- [9] Michael Jenkin.
Student chess-playing robot project at York University.
<http://www.cs.yorku.ca/~vgrlab/>.
- [10] Gerald M. Levitt.
The Turk, Chess Automaton.
McFarland & Company, October 2000.
- [11] Lynxmotion, inc. home page.
<http://www.lynxmotion.com/>.
- [12] Tim Mann.
Chess engine communication protocol.
<http://www.tim-mann.org/xboard/engine-intf.html>.
- [13] Tim Mann.
Winboard graphical user interface for chess.
<http://www.tim-mann.org/xboard.html>.
- [14] Monty Newborn.
Kasparov Versus Deep Blue: Computer Chess Comes of Age.
Springer Verlag, January 1997.
- [15] Bill Wall.
The chess automatons.
<http://www.geocities.com/siliconvalley/lab/7378/automat.htm>.