Connect 4 as a Problem in Artificial Intelligence and Robotics

Brian L. Stuart
Department of Mathematics and Computer Science
Rhodes College
2000 N. Parkway
Memphis, TN 38112
stuart@mathcs.rhodes.edu

Abstract

This report presents work done by the Artificial Intelligence and Robotics class of Summer Scholars 1993 at Rhodes College. Summer Scholars provides college credit for intensive two-week courses taken by high school students. The goal of this course was to develop software to play the game Connect 4 and to control a robotic arm making the moves.

1 Connect 4

The game Connect 4 is played on a vertical grid of six rows and seven columns as illustrated in Figure 1. Players take turns dropping checkers into columns of the board. Each checker falls to the row immediately above the last which was played in that column. Initial drops in any column land in the bottom row. The object of the game is to create a board configuration in which four of a player's pieces are in a contiguous horizontal, vertical or diagonal ($\pm 45°$) line.

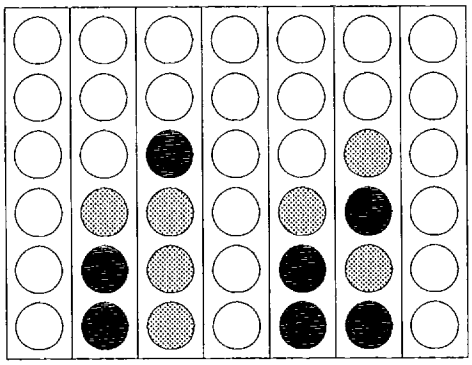

Figure 1: The Connect 4 Playing Board

2 The RoboArm

Each checker played by the computer is moved over the selected column and dropped by a robot arm (RoboArm) on loan to the college by Goldenrod Research Corp of Spalding, NE. The arm has five degrees of freedom: the base (Motor 1), the shoulder (Motor 2), the elbow (Motor 3), the wrist (Motor 4) and the hand (Motor 5) [1]. The base axis is oriented vertically, allowing 254° of left/right motion for the arm. The shoulder axis is oriented horizontally allowing the "upper arm" 95° of movement about the vertical. Approximately 30° of movement toward the rear of the unit and approximately 65° toward the front are allowed. The elbow axis is also oriented horizontally to allow the "forearm" to move up or down from the horizontal. Total movement allowed for the elbow is 85°. The wrist axis runs parallel to the forearm, allowing the hand to twist. This axis was not used in this project. The hand motor allows opening and closing of a gripper attachment. All motors in the RoboArm are free-running DC motors.

The robot provides an RS-232 connection for command messages. Each command message to the robot consists of one or more bytes, the first of which is the command byte. For commands which indicate movement (except open and close), a subsequent argument byte is sent representing the number of degrees to move.

Command sequences may be stored in a buffer for later execution. This feature was used for all

movements during game play. There is no indication of when a command has been completed, and if a second is sent before the first is finished, the first will be terminated. Consequently, all movement commands necessary to pick up a checker, move it over a column, drop it and then return the arm to the pick-up point are loaded into the buffer and executed as a sequence.

3 Software Overview

In order to apply the most computing power available to the task of determining a move, the load is distributed across seven Sun Microsystems workstations. In particular, each of the seven possible moves is examined by a separate machine which produces an evaluation of that move. When all seven systems have evaluated their moves, the best is selected and the robot is instructed to drop a piece into the game board. Each of the seven systems runs the min-max search algorithm to a depth of six plies to evaluate the move under consideration [2,3].

One program (called the "parent") initiates the operation of each of the seven "child" programs, collects their evaluations and controls the robot. It is also responsible for determining when the game has been won or lost. The parent program runs on the same computer as one of the child programs. The robot is connected to a serial port on this computer.

3.1 Parent-Child Protocol

In order to facilitate the communications between the parent process and the various child processes, a parent-child protocol is used. This subsection describes that protocol both from the parent's viewpoint and from the that of the child.

3.1.1 Protocol—Parent Side

The parent initiates the children and uses UNIX pipes to communicate with them. For each child, input and output file descriptors are kept. After each child process is created, but before the child program is executed, the file descriptors are mapped to the standard input and standard output descriptors. Streams are then opened for each of the child file descriptors.

Once all of the children have been created with the pipes in place, the parent enters into the following protocol:

1. If the computer is to be playing first, send the character 'y' to all the children; otherwise, send the character 'n'. (This and all other protocol messages are sent as ASCII characters and are terminated by a newline character.)
2. If the computer is not moving first, send the column number (0–6) of the opponent's first move to all the children.
3. Mark all children active.
4. Let each child be numbered 0 to 6. Send each active child its own number. This is the move number for that child to consider.
5. For each active child, read the signed integer evaluation of its move.
6. Send to each active child the number of the move selected.
7. If that move resulted in a win or the opponent's response resulted in a loss, then send the character 'n' to all active children, indicating that the game is over and that they should all exit.
8. Otherwise, for each active child whose column is full, send the character 'n' and mark the child as inactive and for those whose columns are not full, send the character 'y'.
9. Send the opponent's move to each of the active children.
10. Go to step 4.

Each time a move is selected, the parent program looks up the coordinates for the drop position over the selected column. The appropriate movement angles are computed as described in Section 5 and are encoded into a message which is sent to the robot arm.

3.1.2 Protocol—Child Side

Each of the seven child processes speaks the converse of the parent side of the protocol. All reads are made from the standard input stream and all

sends are done to the standard output stream. The only exceptions are fatal error messages, which go to the standard error stream. The details of this protocol are as follows:

1. Read an indicator of who's going first.
2. If the first character of that message is 'n' or 'N', read a message containing the opponent's first move.
3. Read a message containing a suggested move for evaluation.
4. Send a message containing the integer evaluation value for that move.
5. Read a message containing the move selected for the computer.
6. Read a message indicating whether or not to continue.
7. If the first character of that message is 'n', then exit.
8. Otherwise, read a message containing the opponent's next move.
9. Go to step 3.

Each child maintains a copy of the current board on which it bases its evaluation of the suggested move.

4 Play Strategy

Before codifying a strategy for playing the game, the students in the class played a number of games, paying particular attention to identifying winning strategies. One such strategy is to set up two winning lines each with one missing piece. The two lines are arranged so that the two missing pieces are positioned at empty board positions one atop the other. Unless the opponent forces a similar win first, such a configuration will eventually lead to a win by playing to the column containing the missing pieces. If the opponent does not block the winning move for the lower line, the player who set up the trap will win. If the opponent does block the lower line, then a piece may be played to that column again causing a win in the upper line. An example of this strategy is depicted in Figure 1 where both the second and third positions from the bottom in the middle column would result in wins for the grey pieces.

The students also determined that in the absence of clear strategic direction of play, preference should be given to playing pieces that could build diagonal lines in the lower middle portion of the board.

The evaluation function used in the min-max search returns a large positive value for imminent wins and for winning situations as described above. Similarly it returns a large negative number for the same situations where the opponent would win. Otherwise, the evaluation function returns the weighted sum of occupied piece positions where the positions in the lower middle portion of the board are weighed higher than those around the edges. Positions occupied by friendly pieces are added to the sum and positions occupied by opposing pieces are subtracted. The greatest positive and negative weighted sums are of a smaller magnitude than the large numbers used to indicate imminent wins and losses.

5 Robot Control

All destinations for the robotic hand are given in cylindrical coordinates, (r, θ, z), with $(0, 0, 0)$ being at the point on the table intersected by the vertical rotational axis of the robot base.

5.1 Translating (r, θ, z) to (r', θ', z')

On the RoboArm not all axes intersect at any single point. Therefore, a translation in necessary from the measured coordinates (r, θ, z) to those relative to the intersection of the shoulder axis with the forearm plane, (r', θ', z'). This point is not fixed in space relative to $(0, 0, 0)$, but instead moves as the arm is rotated about its vertical base axis. The translation is needed because the forearm plane will define the plane in which the necessary angles for the shoulder and elbow are computed.

First, consider the vertical axis, the z coordinate. The shoulder is raised above the table by some distance, l_5, giving

$$z' = z - l_5$$

for the desired height of the hand above the shoulder axis.

Next, consider r and θ together. Figure 2 represents the RoboArm viewed from above. In it, the shoulder axis is offset behind that for the base by a distance l_3. Similarly, the vertical plane which passes through the forearm is offset to the right of the base axis by a distance l_4.

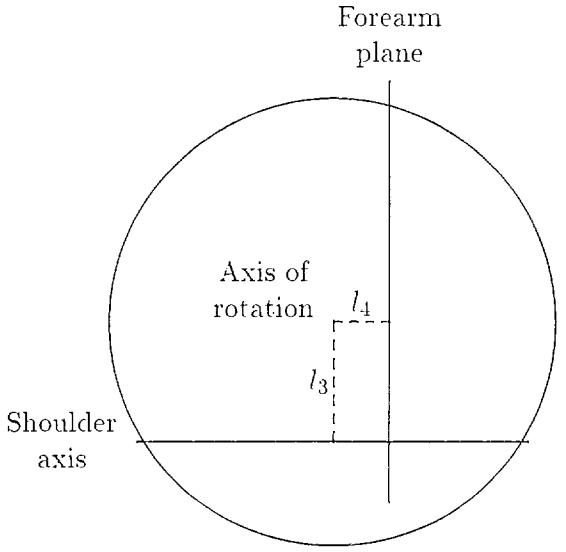

Figure 2: Top View of the RoboArm

What is needed is an r' and θ' relative to the intersection of the shoulder axis and the forearm plane. Figure 3 shows the situation. The origin of the solid axes is the point from which measurements are taken. The origin (point C) of the dashed axes is the point of intersection between the shoulder axis and the forearm plane. Point A is the desired position of the hand. To avoid the risk of ambiguity and clutter in the figure, the following dimensions are defined here:

r'': distance between A and B
r': distance between A and C
l_3: distance between B and C
l_4: distance between O and B

Now, because $\triangle AOB$ is a right triangle,

$$r'' = \sqrt{r^2 - l_4^2}.$$

and because of the definitions of the distances

$$r' = r'' + l_3.$$

The value of α also comes from the fact that $\triangle AOB$ is right and from the relationship of $\cos(\cdot)$ to the sides in a right triangle. So,

$$\alpha = \cos^{-1}\left(\frac{r''}{r}\right).$$

$$\beta = 180^\circ - \alpha - \theta,$$

and

$$\theta' = 180^\circ - \beta = \theta + \cos^{-1}\left(\frac{r''}{r}\right).$$

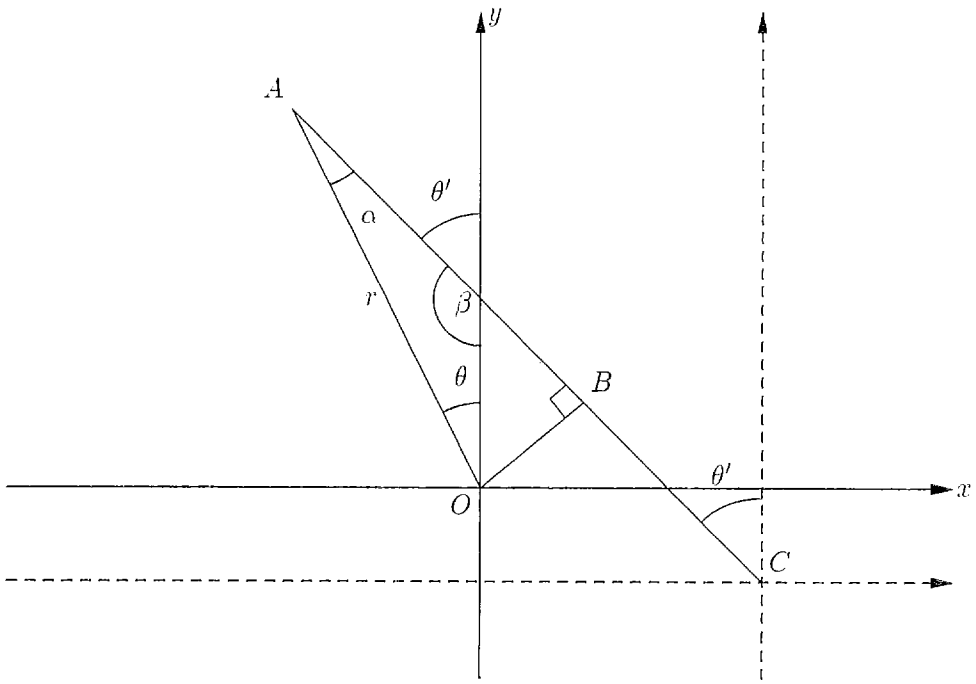

Figure 3: Calculation of r' and θ'

5.2 Finding the Shoulder and Elbow Angles

Next the shoulder and elbow angles must be computed. These are computed from the perspective looking at the forearm plane along the shoulder axis. This is illustrated in Figure 4.

In this figure, the length of the RoboArm "upper" arm is l_1 and the length of the forearm is l_2. Consider two circles, one of radius l_1 around the origin and one of radius l_2 around the point (r', z'). The equations for these two circles are

$$\begin{array}{ll} c_1: & u^2 + v^2 = l_1^2, \text{ and} \\ c_2: & (u - r')^2 + (v - z')^2 = l_2^2. \end{array}$$

Solving c_1 for u

$$u = \pm\sqrt{l_1^2 - v^2}.$$

and multiplying out c_2

$$u^2 - 2ur' + r'^2 + v^2 - 2vz' + z'^2 = l_2^2.$$

Now substituting for u in c_2 gives an equation only in v. Choosing the positive square root in substituting for u identifies the point on c_1 of height v in the right half of the plane and the negative choice in the left half of the plane. While the actual solution could be either, the value of v will be the same regardless. This substitution gives

$$l_1^2 - v^2 - 2r'\sqrt{l_1^2 - v^2} + r'^2 + v^2 - 2vz' + z'^2 = l_2^2.$$

A little algebra yields

$$(r'^2 + z'^2)v^2 - (2z'w)v + (w^2 - r'^2 l_1^2) = 0,$$

where $w = \frac{l_1^2 + r'^2 + z'^2 - l_2^2}{2}$, which can be solved by the quadratic formula. The positive square root is always chosen in solving the quadratic formula for v, because choosing the lower height may lead to a negative height for the elbow (this does not make sense physically) and because there is more downward movement than upward on the elbow.

Now that the vertical position of the elbow is established, the angles at which the shoulder and elbow need to be set can be found. Using the same reasoning as for α in the previous subsection,

$$\phi' = \pm\cos^{-1}\left(\frac{v}{l_1}\right).$$

Note that u may be positive or negative, which would be consistent with ϕ' being positive or negative. One of the easiest ways to decide which is to determine if the distance between (u, v) and (r', z')

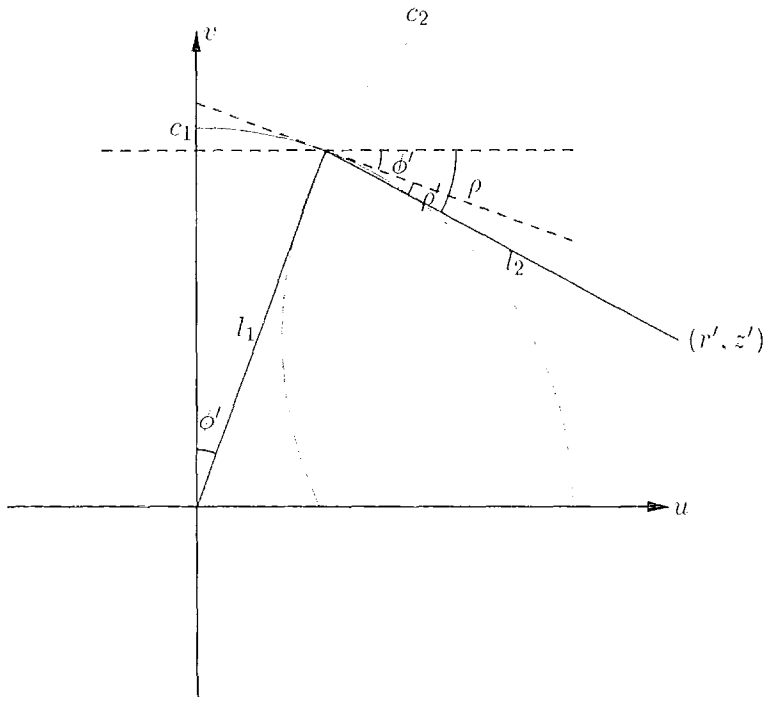

Figure 4: Idealized Robot Arm Viewed From the Side

is equal to l_2 with u positive. If it is then u must be positive, if not it must be negative and so must ϕ'.

The angle between the forearm and the horizontal is given by

$$\rho = \sin^{-1}\left(\frac{v - z'}{l_2}\right).$$

But on the RoboArm, the elbow angle is specified with respect to a plane perpendicular to the upper arm. So adjusting ρ to get the actual elbow angle

$$\rho' = \rho - \phi'.$$

6 Conclusion

After a high level of interest among prospective students, a surprisingly few students (three) enrolled. This created an atmosphere of pushing hard against the odds and a close deadline. Fortunately, the author and instructor had already written a preliminary version of the code to implement robot control as described in the previous section. The three students each focused on one of the following: the parent program, the child program and the evaluation function.

None of the students had substantial experience programming in C, (in which the project was developed), but all had some experience in other languages (mostly BASIC and Pascal). Consequently, some time was spent the first few days examining the basic features of the C language along with developing a design for the project. Approximately four to five days were devoted to developing code for the system leaving a couple of days for fine tuning.

Because the RoboArm does not use stepping motors, some amount of inaccuracy in positioning was observed. While this inaccuracy was small enough not to be a problem for single moves in the game, compounding of the error caused the robot to drift out of calibration over the course of five to ten moves. Consequently, before the robot picked up the next checker to play, the parent program paused to allow for any necessary recalibration.

On the final day of the program, the students challenged students in all other Summer Scholars classes to play against the robot. Of the five games played by students and counsellors, the computer/robot won four.

References

[1] Goldenrod Research, Inc., *RoboArm Operator's Manual.*

[2] Nilsson, Nils J. (1980), *Principles of Artificial Intelligence.* Los Altos, CA: Morgan Kaufmann.

[3] Charniak, E. & McDermott, D. (1987). *Introduction to Artificial Intelligence.* Reading, MA: Addison-Wesley.

*******Dutch Flag Continued From Page 40******

Final Comments

I have taught Computer Science too many times with the first example of an abstract data type being either ADT List or ADT Stack. By using ADT BucketsNPebbles instead in a recent course, students appeared able to grasp the concept better. It also made for good, enjoyable student participation in the process.

References

[1] Backhouse, Roland C, *Program Construction and Verification*, Prentice Hall, Englewood Cliffs, New Jersey, 1987, 189-194.

[2] Guttag, John & Liskov, Barbara, *Abstraction and Specification in Program Development*, The MIT Press, Cambridge, Massachusetts, 1986, 56-59.

[3] Helman, Paul & Veroff, Robert, *Walls and Mirrors Intermediate Problem Solving and Data Structures*, The Benjamin/ Cummings Publishing Co, 1988, 209-210.