Connect Four Robot

Implementation of AI-strategies in a Connect Four robot

XIYAO SONG

LISELOTT HULTROS
Connect Four Robot

Implementation of AI-strategies in a Connect Four robot

XIYAO SONG, LISELOTT HULTROS

Bachelors’s Thesis at ITM
Supervisor: Nihad Subasic
Examiner: Nihad Subasic

TRITA-ITM-EX 2018:57
Abstract

Connect four is a two player game in which the players take turns placing discs in a 6x7 grid. Each player has discs in their own specific color. Their aim is to get four of their own discs in a horizontal, vertical, or diagonal alignment. While doing so they have to prevent their opponent from getting four of their discs in a row.

In this project, the purpose is to design and build a connect four playing robot, that can play connect four opposite a human. To do so, the robot has to be able to physically move and drop discs in the grid of the game, it also has to be able to figure out in which column to drop the discs, and detect where the player puts their discs.

To achieve this, a demonstrator was built. It consists of a frame around the game, onto which motors are placed to move and drop discs into the game. Photoreflectors are placed above each column to detect the player’s discs. The motors and sensor are controlled by an Arduino, and a Raspberry Pi is tasked to run the AI algorithm to calculate the next moves for the robot.

This was done with a satisfying result and at the end of the project the demonstrator could play connect four against a human opponent and almost always win.
Referat

Fyra i rad-robot

Fyra i rad är ett sällskapsspel för två personer i vilket de två spelarna turas om att lägga brickor i ett rutnät på 6x7 rutor. Varje spelare har brickor i sin egen specifika färg. Målet för varje spelare är att få fyra av sina egna brickor i en horisontell, vertikal, eller diagonal rad. Detta medan de hindrar den andra spelaren från att få fyra av sina brickor i en rad.

Syftet med detta projekt är att designa och bygga en fyra i rad-spelande robot som kan spela mot en människa. För att göra det behöver roboten kunna fysiskt flytta och släppa brickor i spelets kolumner, räkna ut vilken kolumn som den ska lägga brickor i, samt kunna läsa av var motståndaren lägger sina brickor.


Detta genomfördes med tillfredsställande resultat och vid projektets slut kunde roboten spela fyra i rad mot en mänsklig motståndare och nästan alltid vinna.
Acknowledgements

We have received a lot of help from others throughout this project, and for this, we would like to thank the following people:

Nihad Subasic for being a supportive and helpful course coordinator.

Thomas Östberg for providing some of the materials used in the construction and also some guidance in what to use.

Staffan Qvarnström for helping with some mechanical problems we encountered during the building process.

Johan Ehrenfors and Martin Gylling for contributing to the project in several ways with good ideas and their own experiences from when they did their Bachelor’s degree.

Liselott Hultros, Xiyao Song
Royal Institute of Technology, Stockholm, May 2018
# Contents

1 Introduction ............................................. 1
   1.1 Background ........................................... 1
   1.2 Purpose ............................................... 1
   1.3 Scope .................................................. 2
   1.4 Method ............................................... 2

2 Theory .................................................. 3
   2.1 Electronic Components ................................. 3
      2.1.1 Stepper motor ..................................... 3
      2.1.2 Servo motor ........................................ 3
      2.1.3 H-bridge ............................................ 4
      2.1.4 Microcontroller ..................................... 4
      2.1.5 Raspberry Pi ......................................... 4
      2.1.6 Photoreflector ....................................... 4
      2.1.7 Multiplexer .......................................... 4
   2.2 Connect Four Algorithm ................................. 5
      2.2.1 Minimax .............................................. 5
      2.2.2 Alpha Beta Pruning .................................. 6
      2.2.3 Iterative Deepening ................................ 7

3 Demonstrator ........................................... 9
   3.1 Construction .......................................... 9
      3.1.1 Frame ............................................... 9
      3.1.2 Magazine ............................................ 11
      3.1.3 Claws ............................................... 11
   3.2 Electronics .......................................... 12
      3.2.1 Arduino Uno ......................................... 12
      3.2.2 Raspberry Pi 3 Model B+ ............................. 12
      3.2.3 Stepper Motor Driver ............................... 12
      3.2.4 Servo Motors ....................................... 13
      3.2.5 Stepper Motor ....................................... 13
      3.2.6 Photoreflectors and multiplexer ..................... 14
   3.3 Software .............................................. 15
3.3.1 Arduino .................................................. 15
3.3.2 Raspberry Pi ........................................... 15
3.4 Tests and Results ........................................ 16

4 Discussion and Conclusions 17
  4.1 Discussion ............................................... 17
  4.2 Conclusions ............................................ 18

5 Recommendations for Future Work 19

Bibliography 21

Appendices 22

A Arduino Code 23

B Python Code 27

C Overall System Wiring Diagram 37

D Multiplexer-Photoreflector Wiring Diagram 39
List of Figures

2.1 Channel selection in a multiplexer. (Created in Google Drawings) . . . 5
2.2 An example of the Minimax algorithm. (Created in Google Drawings) . 6
2.3 An example of the Alpha Beta Pruning algorithm. (Created in Google Drawings) ........................................ 7
2.4 Search order of two iterations of Iterative Deepening search on a search tree. (Created in Google Drawings.) ........................................ 7

3.1 The Connect Four robot with its frame and magazine wrapped around the game. ................................................................. 10
3.2 Close up of the sensors. ............................................................ 10
3.3 Top section of the magazine. ...................................................... 11
3.4 Claws fastened onto the timing belt holding a disc. ......................... 12
3.5 The limit switch and the stepper motor on the back side of the robot.. 13
3.6 The custom made circuit board for the multiplexer-photoreflector sub-system. ................................................................. 14

C.1 Overall system wiring diagram. (Created in Fritzing) .................... 37

D.1 Multiplexer-Photoreflector Wiring Diagram. (Created in Edraw) ....... 39
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>CAD</td>
<td>Computer Aided Design</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DC</td>
<td>Direct Current</td>
</tr>
<tr>
<td>I/O</td>
<td>Inputs/Outputs</td>
</tr>
<tr>
<td>IR</td>
<td>Infrared</td>
</tr>
<tr>
<td>KTH</td>
<td>Kungliga Tekniska Högskolan (Royal Institute of Technology)</td>
</tr>
<tr>
<td>LCD</td>
<td>Liquid Crystal Display</td>
</tr>
<tr>
<td>LED</td>
<td>Light Emitting Diode</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This chapter contains some background information about the project as well as what the overall goals are and how to achieve them.

1.1 Background

Connect Four is a two-player game in which each player has a set of discs in a specific color. They take turns dropping the discs in a 6 by 7 grid. The purpose of the game is for the player to get four of their discs in a row, horizontally, vertically, or diagonally, whilst of course preventing the other player from doing the same.

The logic of the game has already been solved[1], the player who plays first can always win if they play perfectly. However, a person rarely does that and only one mistake can make them lose their advantage.

1.2 Purpose

The goal of this project is to create a robot that can play connect four opposite a person. To do this the following problems has to be solved:

- What sensors to use and how to place them to record the state of the game?
- How to construct an algorithm to calculate the next move?
- How to translate the calculations into physical actions?
1.3 Scope

The main objective is to create a robot which can figure out where to put discs in a connect four game, and also physically do it. The robot does not have to be able to sort the discs on its own and put them back in the depository. This can be done manually. Given the casual nature of this game, the calculation time is limited to 20 seconds, and the human player will always play first. This gives the player a chance to win, it also means that the AI algorithm doesn’t have to be perfect given the time constraints, but good enough to beat the average human player.

1.4 Method

The construction will be designed around and adapt to an off the shelf Connect Four game. The robot has to be able to pick up discs, one at a time, move them, and place them in the desired column of the grid. This will be done with one stepper motor and two servo motors controlling and moving a device that can hold one disc at a time.

The construction is going to be built mainly with acrylic glass that will be cut by a laser cutter. Other necessary components such as gears, pulleys and electronic components will be purchased.
Chapter 2

Theory

In this project a multitude of electronic components will be needed, this chapter serves to explain the theory behind them, and the core methods behind the AI-algorithm solving the Connect Four game will also be described here.

2.1 Electronic Components

This section aims to explain the theories behind some of the electronic components used in this project.

2.1.1 Stepper motor

A stepper motor is a DC-motor in which the rotation is divided into several, equally sized steps. The position of the motor can be very specifically controlled by knowing how many steps the motor should turn, thus an open loop control system is usually sufficient without the need for closed loop feedback control. The rotation is controlled by polarizing the two sets of coils inside the motor, the magnetized rotor will then align itself accordingly and turn the motor a certain degree[3].

2.1.2 Servo motor

Servo motors are small electric motors with high torque and energy efficiency. A servo motor generally consists of three components, a DC-motor, a potentiometer and a control circuit. This enables the servo motor to be regulated by a closed feedback control system. The potentiometer which is a variable resistor gets turned by the motor and the precise position of the motor is then read by the control circuit which in turn sends control signals back to the motor so that it can reach the intended position[12].
2.1.3 H-bridge

An H-bridge is a commonly used circuit to control DC-motors. It consists of 4 switching elements in an H configuration. By turning on one pair of diagonally placed switches, the current flows one way, turning on the other pair results in reversal of the current flow. A DC-motor can thus be driven either forward or backward. Each stepper motor contains two sets of coils, and two H-bridges are therefore required to drive one stepper motor[18].

2.1.4 Microcontroller

A microcontroller is a simple computer on an integrated circuit designed for embedded systems. It contains a central processing unit, on-board memory and programmable Inputs/Outputs. Unlike a regular computer, a microcontroller has no operating system, it is programmable through compiling C/C++ code down to machine code which the CPU understands and executes. A microcontroller is often less powerful than a regular computer, but it allows direct lower level access to the electronic components connected to it[2].

2.1.5 Raspberry Pi

A Raspberry Pi is a small and affordable single-board computer. Unlike a microcontroller, a Raspberry Pi has an operating system, it runs Linux. It also has a much faster CPU than a microcontroller. This combines with its low price and small size makes it ideal for robotic applications where a lot of computational power is required[6].

2.1.6 Photoreflector

A photoreflector consists of an IR LED and a phototransistor. The LED emits light, while the phototransistor measures the amount of reflected light. If more light is reflected, more current will flow through the phototransistor. Because current is proportional to voltage according to Ohm’s Law, by using a resistor, the current can be indirectly measured by measuring the voltage through the phototransistor[19].

2.1.7 Multiplexer

A multiplexer is a combinational logic switching device. It has multiple input lines and a single common output line. Switching between the input lines or “channels” is achieved by turning on/off the appropriate data select inputs. The combination of the data select inputs is in effect a binary representation of the channel number. See figure 2.1 for an example. Data from the selected channel is then forwarded to the output. This can be done at a very high speed, and multiple channels of data can be scanned and passed through the single output on the multiplexer, eliminating the need for multiple inputs on the receiving device[5].
2.2. CONNECT FOUR ALGORITHM

The core concepts towards constructing a Connect Four algorithm are described in this section.

2.2.1 Minimax

Solving Connect Four can be seen as finding the best path in a decision tree. Each node in the search tree is a certain position in the game, and each position has a score, it tells you how likely you are to win the game at that position. Given a position, a player has to choose a move which leads to, in the case of Connect Four, 7 other possible moves for the opponent. When it’s your turn, you would like to choose a move that maximizes your score. On the other hand, when it’s your opponent’s turn, he would like to do the same for himself, thus minimizing your score on his turn. This process repeats itself until a winning move is found, or all possible moves have been exhausted in the case of a draw [15].

Figure 2.2 shows a simple example of how the Minimax algorithm works. The white nodes are maximizing nodes and the black ones are minimizing nodes. A white node will choose the node with the highest score directly below it, and a black node will do the opposite, choose the node with the lowest score. The result in this case is the highlighted red line.
The Minimax algorithm searches through all the possible moves to find the best path, it is also known as a depth first search algorithm. This is however time consuming, a standard 6x7 Connect Four game has $4.2 \times 10^{12}$ possible positions [4], this is clearly not practical in a playable game setting. Other methods will have to be incorporated to trim the search tree and shorten the computing time.

### 2.2.2 Alpha Beta Pruning

Alpha beta pruning is a method to eliminate large parts of the search tree from consideration, this saves a lot of time and makes the search algorithm more efficient.

Alpha is the worst score that the maximizing player is guaranteed to have, and beta is the worst score that the minimizing player is guaranteed to have. Before exploring the next node, it checks if alpha is greater than or equal to beta, if that’s the case, then this node and all its descendants don’t have to be explored because they won’t affect the outcome of the game [13].

In this example in figure 2.3, when the black node with the score 3 is explored, the maximizing white node above it knows that the worst score it can have is greater than or equal to 3. Its alpha value gets updated to $\geq 3$ while its beta value of 1 is retained from the black node above it. Now alpha is greater than beta, the node with the question mark doesn’t have to be explored, it is pruned. This makes sense because the black minimizing node has to choose between 1 and $\geq 3$, and it will always choose 1, so it doesn’t matter what the question mark is, that node will never come into play. On the right side, the beta value of the first minimizing node gets updated to $\leq -3$, its alpha value of 1 is retained from earlier. Alpha is now greater than beta, those 3 nodes in the lower right corner can be pruned.
2.2. CONNECT FOUR ALGORITHM

The optimal path stays the same as before even though less nodes have been explored.

![Diagram of Alpha Beta Pruning algorithm](image)

**Figure 2.3.** An example of the Alpha Beta Pruning algorithm. (Created in Google Drawings)

2.2.3 Iterative Deepening

Iterative deepening is a depth-limited version of the depth-first search algorithm which is run repeatedly with increasing depth. For each iteration, the depth is increased by one, and the result is saved, so that the next iteration can use the previous result to prune the search tree more efficiently. Figure 2.4 shows two iterations of Iterative Deepening search on a search tree. This also has the added benefit that at any time, the program can play the best move it has found so far [14]. This is very useful to our robot, as a time limit of 20 seconds has been set in order to keep the flow of the game.

![Diagram of Iterative Deepening search](image)

**Figure 2.4.** Search order of two iterations of Iterative Deepening search on a search tree. (Created in Google Drawings.)
Chapter 3

Demonstrator

This chapter aims to present the constituent parts of the robot and the build process.

3.1 Construction

The demonstrator is mainly made from laser cut acrylic glass and 3D-printed parts.

3.1.1 Frame

The frame is made out of acrylic glass cut into desired shapes with a laser cutter. It is designed to fit around a store purchased Connect Four game. It is a relatively simple construction made almost exclusively out of rectangles of different sizes with various holes and grooves cut into them and screwed together. To stabilize the frame, two base plates, one on each side, were designed in CAD using Solid Edge ST9, and 3D-printed. Figure 3.1 shows the whole robot with the frame and magazine wrapped around the Connect Four game.

Figure 3.2 shows the seven sensors that are fitted into cut-outs in the frame, one above each column.
Figure 3.1. The Connect Four robot with its frame and magazine wrapped around the game.

Figure 3.2. Close up of the sensors.
3.1. CONSTRUCTION

3.1.2 Magazine
The magazine is also made out of acrylic glass and consists of a vertical holder, and a dispenser connected to a servo motor. It is standing on a 3D-printed base plate. The servo motor rotates the dispenser to feed the game one disc at a time from the magazine. Figure 3.3 shows a close up photo of the top section of the magazine.

3.1.3 Claws
A pair of claws were used to grab, hold and drop discs into the game. One of the claws is fixed, the other one can be rotated by a servo motor. The whole assembly was fastened onto the timing belt so that it can be moved horizontally by a stepper motor mounted on one side of the frame. Figure 3.4 shows the claws in a closed position holding a disc.
3.2 Electronics

The electronics used in the project are described in this section. For the complete wiring diagram, see Appendix C.

3.2.1 Arduino Uno

An Arduino Uno was used as the microcontroller in this project. It’s an open source platform and is easy to use and develop for[10]. It has 6 analog and 14 digital I/O pins[11]. The Arduino Uno controls the motors and reads the sensors. More about the software loaded onto the Arduino in section 3.3 Software.

3.2.2 Raspberry Pi 3 Model B+

A Raspberry Pi 3 Model B+[7] was used to run the AI-algorithm to calculate the next move. It was powered by a micro-USB cable, it was also connected to the Arduino through another USB-cable which provided power to the Arduino and also enabled communications between the two devices. A LCD display was also attached to the Raspberry Pi to provide some useful information to the player. The software running on the Raspberry Pi will be discussed in section 3.3 Software.

3.2.3 Stepper Motor Driver

Instead of two separate H-bridges to control the stepper motor, a DRV8825 stepper motor driver was used. The driver takes two inputs, STEP and DIR to drive the motor through two pairs of pins which are connected to the two sets of coils inside of the motor. When STEP is set to HIGH through one of the digital pins on the
3.2. ELECTRONICS

Arduino, the motor rotates one step. DIR can be set to either HIGH or LOW to determine the direction of rotation of the motor\[^9\]. The driver itself was powered by the 5V power output on the Arduino. However the motor needed a separate 12V power supply and a 470\(\mu\)F capacitor was also added to protect the circuit from surges. To avoid heat problems on both the driver and the stepper motor and to reduce power consumption, the SLEEP pin on the driver was also connected to the Arduino. When it is set to LOW, the driver is put to sleep, drawing no power itself, neither does it pass any power to the stepper motor. When the stepper motor is needed, the SLEEP pin can then be set to HIGH to resume its normal operations.

3.2.4 Servo Motors

One servo motor was used to operate the dispenser. In its initial position the circular cut-out on the dispenser is lined up with the magazine so that one disc would fall into it. The servo motor then turns anti-clockwise and brings the disc to its drop-off point. Once the disc is dropped off, the servo motor then rotates the dispenser back to the initial position waiting for the signal for when the next disc is needed.

The other servo motor was used to operate the claws. When the claws are positioned above the desired column, the servo motor rotates one of the claws so that the disc falls into the column.

3.2.5 Stepper Motor

A stepper motor was used to transport the claws to the correct positions to either receive a disc from the dispenser or to drop it off into the grid. A limit switch was added so that the initial position can be reset each time making the system more robust. Figure 3.5 shows the limit switch and the stepper motor on the back side of the frame.

Figure 3.5. The limit switch and the stepper motor on the back side of the robot.
3.2.6 Photoreflectors and multiplexer

Seven photoreflectors of the RPR-220[17] model made by ROHM Semiconductors were used, one above each column to detect falling discs. They were calibrated to only detect short range disruptions of light intensity to eliminate false positives. This was achieved partly by connecting the phototransistor to a 10k Ohm resistor so that they would be insensitive to distances larger than the thickness of the grid of the game. And partly through the appropriate trigger threshold set in software. The Arduino reads the voltage from each photoreflector through a CD4051B multiplexer[8]. The threshold was set to as high as possible without missing any positive signals. The wiring diagram for the multiplexer and the sensors can be found in Appendix D. Figure 3.6 shows the custom made circuit board for this subsystem.

![Custom made circuit board for the multiplexer-photoreflector subsystem.](image)

**Figure 3.6.** The custom made circuit board for the multiplexer-photoreflector subsystem.
3.3 Software

The software used in this project will be discussed in this section.

3.3.1 Arduino

The Arduino is responsible for controlling the motors and reading the sensors. It initiates the robot by opening the claws, and moves them out a certain distance before homing to its origin by triggering the limit switch. Meanwhile a disc is fed from the magazine, the claws then grabs the disc. At this time, it starts to read the sensors, it will continue to do so until one of the sensors is triggered. If that happens, it sends the data about which column was played to the Raspberry Pi and waits for instructions. The Raspberry Pi calculates which column the robot wants to play and sends that information back to the Arduino. The Arduino can now move the claws holding the disc to that column and drop it, whilst the dispenser feeds another disc. The claws can now move back to its initial position by triggering the limit switch and grab another disc. It’s now time to read the sensors again and the loop repeats. The complete code can be found in Appendix A.

3.3.2 Raspberry Pi

The AI-algorithm to solve for the next move is based on the work by Martin Saveski of MIT [16]. It’s written in Python and utilizes Minimax, Alpha Beta Pruning and Iterative Deepening to search for the best move. The evaluation function to score a given position used in this algorithm is a heuristic function. In other words, it’s based on experimentation and only approximates the true value of the position. This improves the speed of the algorithm by sacrificing accuracy. The scores for positions in the bottom three rows are pre-calculated and saved in text files as these positions in the beginning of the game are the most time consuming to calculate. The order in which moves are considered has also been optimized to improve the performance of Alpha Beta Pruning. It starts in the middle and alternating one column left and right.

We have modified the code so that the human player always plays first. The code for USB communication with Arduino has been incorporated and the code for the LCD display has also been added. It shows which column the player has played, which column the robot has played, how many seconds it took for the robot to think, the depth reached by the search algorithm and of course the win/loss messages. The code has also been modified to be able to run continuously even after a completed game. This eliminates the need to manually restart the program after each game. The complete code can be found in Appendix B.
3.4 Tests and Results

The demonstrator was tested with multiple opponents on the day of the KEXPO, an exhibition of Bachelor’s degree projects arranged by KTH open to the public. Around 15-20 games were played that day and the robot only lost twice. The construction was not as robust as we’d hoped and needed adjusting multiple times.
Chapter 4

Discussion and Conclusions

4.1 Discussion

The robot functioned mostly as expected. However, there were a few problems.

The biggest problem was that the photoreflectors didn’t always register the discs that were played. This was probably because of failures in the connections. To avoid this, multi-stranded wires should have been used so that there wouldn’t be any single points of failure. It would also withstand the vibrations generated by the movements a lot better.

Another problem was that the magazine wasn’t in any way connected to the frame, therefore it could move and the discs would no longer fall into the desired place. This could have been solved by making a combined holder for the frame and the magazine so that the distance between them wouldn’t change so much so that it would stop working.

The third problem was that the losses could have been avoided. The losses occurred during similar conditions. We suspect that there’s an oversight in the evaluation function of the algorithm. It would sometimes value the opportunity to win in the next move higher than stopping the opponent from winning this move. For example, if the opponent had three discs in a diagonal line, and the robot had three discs in a diagonal line just above it. The robot would neglect to block the human player from winning. This could be remedied by checking for losing positions first before going into the search tree. But overall we were still quite pleased with how it performed and the players seemed to have fun playing against the robot.
4.2 Conclusions

The initial goal of constructing a Connect Four robot capable of physically playing a game of Connect Four against a human opponent and in most cases win has been met.

Photoreflectors placed above each column was able to detect the falling discs placed by human players and record the state of the game.

Although we didn’t write the algorithm from scratch, we do have a deeper understanding of how the combination of Minimax, Alpha Beta Pruning and Iterative Deepening could help solve a game like Connect Four.

With the Arduino controlling the motors and sensors working in concert with the Raspberry Pi running the AI-algorithm, data from the physical world can be fed into the calculations and the result can be translated into physical actions.
Chapter 5

Recommendations for Future Work

Due to the scope of this project the demonstrator was not able to sort the discs and place them in the magazine. This could be a natural place to start to increase the autonomy of the robot. Opportunities to further optimize the algorithm could also be explored. Another obvious improvement is to make the construction more robust and foolproof. An effort could be made to make the robot more consumer friendly by improving the user interface and add different difficulty levels.
Bibliography


Appendix A

Arduino Code

1 /////////////////////////////////////////////////////
2 // Authors: Xiyao Song and Liselott Hultros //
3 // KTH, May 2018 //
4 // Connect Four Robot Arduino Code //
5 // Bachelor's Project in Mechatronics //
6 //////////////////////////////////////////////////////////////////////////////////
7
8 // Include libraries
9 #include <Servo.h>
10
11 // Declare pins
12 const byte sensor = A0; // multiplexer to Arduino analog in
13 // The multiplexer address select pins (A/B/C)
14 const byte addressA = 4; // low-order bit
15 const byte addressB = 5;
16 const byte addressC = 6; // high-order bit
17 const byte ledpin = 7; // sensor ledpin
18 // Servo pins
19 const byte dispenserPin = 2;
20 const byte servoClawPin = 3;
21 // Stepper motor driver pins
22 const byte dirPin = 8;
23 const byte stepPin = 9;
24 const byte sleepPin = 10;
25 // End switch pin
26 const byte switchPin = 11;
27
28 // Create servo objects
29 Servo servoClaw;
Servo dispenser;

// Declare global variables
// int columnrobot;

void setup() {
  // Declare pinModes
  pinMode(sensor, INPUT);
  pinMode(addressA, OUTPUT);
  pinMode(addressB, OUTPUT);
  pinMode(addressC, OUTPUT);
  pinMode(ledpin, OUTPUT);
  pinMode(stepPin, OUTPUT);
  pinMode(dirPin, OUTPUT);
  pinMode(sleepPin, OUTPUT);
  pinMode(switchPin, INPUT_PULLUP);
  // Declare servo pins
  servoClaw.attach(servoClawPin);
  dispenser.attach(dispenserPin);
  // Start serial monitor
  Serial.begin(115200);
  servoClaw.write(10);
  delay(1000);
  placedisc(2);
  // Serial.println("setup");
} // end of setup

void placedisc(int column) { // function to physically place a disc given a column number
  // Go to the correct column
  // servoClaw.write(10); // keep the claws closed
  digitalWrite(sleepPin, HIGH); // wake up the stepper motor
  digitalWrite(dirPin, HIGH); // set direction to outwards
  // Calculate the number of steps: 500 steps to 1st column
  // then add 330 steps to each subsequent one
  int steps = 440 + 335 * (column - 1);
  for(int x = 0; x < steps; x++) { // move the stepper motor to the correct column
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(500);
  }
}
servoClaw.write(85);  // drop the disc by opening the claws
digitalWrite(sleepPin, LOW);  // put stepper motor to sleep
dispenser.write(10);  // open dispenser
delay(2000);
dispenser.write(180);  // close dispenser
delay(2000);
digitalWrite(sleepPin, HIGH);  // wake up the stepper motor
// Homing until switch is triggered
digitalWrite(dirPin, LOW);  // set direction to home/inwards
while (digitalRead(switchPin) != HIGH) {  // move until the switch is triggered
digitalWrite(stepPin,HIGH);
delayMicroseconds(500);
digitalWrite(stepPin,LOW);
delayMicroseconds(500);
}
digitalWrite(sleepPin, LOW);  // put stepper motor to sleep
servoClaw.write(10);  // close the claws to pick up a disc
delay(1000);

int readSensor (const byte which) {  // function to read one sensor
  // Select correct MUX channel by turning on/off the corresponding binary bits
digitalWrite (addressA, (which & 1) ? HIGH : LOW);  // low -order bit
digitalWrite (addressB, (which & 2) ? HIGH : LOW);
digitalWrite (addressC, (which & 4) ? HIGH : LOW);  // high-order bit
  // Now read the sensor
int ambient = 0;
int lit = 0;
int value = 0;
digitalWrite(ledpin, LOW);  // disable the LEDs
delay(5);
ambient = analogRead(sensor);  // read the ambient light
digitalWrite(ledpin, HIGH);  // enable the LEDs
delay(5);
lit = analogRead(sensor);  // read the light with LEDs on
value = lit - ambient;  // calculate the true value
return value;
APPENDIX A. ARDUINO CODE

```java
109 } // end of readSensor
110
111 int readDisc () { // function to read all the sensors and
    return the played column
112     int i = 0;
113     while ( readSensor (i % 7) < 160) { // cycle through MUX
114         i++;
115     }
116     return i % 7 + 1; // return the column number by adding 1
117 }
118
119 void loop () {
120     byte columnhuman = readDisc(); // Reads the column played
121         by human
122     Serial.println (columnhuman); // Sends the column number
123         to Arduino via serial
124
125     // Waits for response from Raspberry Pi and places a disc
126         in the right column
127     bool received = false;
128     while (received == false) {
129         if (Serial.available() > 0) {
130             byte columnrobot = Serial.read() - '0';
131             received = true;
132             placedisc (columnrobot);
133             delay (1000);
134         }
135     }
```
Appendix B

Python Code

#########################################################################
# Author: Martin SAVESKI
# Date: May 2009
# License: MIT
#
# Modified by: Xiyao Song & Liselott Hultros
# Date: May 2018
#
# AI Agent for Connect Four
#
# ConFour.py
# * Evaluation Function – optimized using hash table
# * Alpha Beta Pruning Search Algorithm
# * Iterative Deepening Search Algorithm
#########################################################################

import serial
import time
from I2C_LCD_driver import *

ser = serial.Serial('/dev/ttyACM0', 115200)
ser.close()
mylcd = I2C_LCD_driver.led()
# Evaluation Method, uses the hash tables to evaluate lines

```python
def eval(t):
    evaluation=0
    evaluation+=17[1000000*t[0][0]+100000*t[0][1]+10000*t[0][2]]
    evaluation+=17[1000000*t[1][0]+100000*t[1][1]+10000*t[1][2]]
    evaluation+=17[1000000*t[2][0]+100000*t[2][1]+10000*t[2][2]]
    evaluation+=17[1000000*t[3][0]+100000*t[3][1]+10000*t[3][2]]
    evaluation+=17[1000000*t[4][0]+100000*t[4][1]+10000*t[4][2]]
    evaluation+=17[1000000*t[5][0]+100000*t[5][1]+10000*t[5][2]]
    evaluation+=16[1000000*t[0][0]+100000*t[0][1]+10000*t[0][2]]
    evaluation+=16[1000000*t[1][1]+100000*t[1][2]]
    evaluation+=16[1000000*t[2][2]]
    evaluation+=16[1000000*t[4][4]+100000*t[4][5]+t[4][6]]
    evaluation+=16[1000000*t[5][5]+100000*t[5][6]]
    evaluation+=16[1000000*t[0][6]+100000*t[0][7]+10000*t[0][8]]
    evaluation+=16[1000000*t[0][7]+100000*t[0][8]]
    evaluation+=16[1000000*t[0][8]]
    evaluation+=16[1000000*t[2][0]+100000*t[2][1]+10000*t[2][2]]
    evaluation+=16[1000000*t[2][1]+100000*t[2][2]]
    evaluation+=16[1000000*t[2][2]]
    evaluation+=16[1000000*t[2][5]+100000*t[2][6]]
    evaluation+=16[1000000*t[2][6]]
    evaluation+=16[1000000*t[3][0]+100000*t[3][1]+10000*t[3][2]]
    evaluation+=16[1000000*t[3][1]+100000*t[3][2]]
    evaluation+=16[1000000*t[3][2]]
    evaluation+=16[1000000*t[3][4]+100000*t[3][5]+t[3][6]]
    evaluation+=16[1000000*t[3][5]+100000*t[3][6]]
    evaluation+=16[1000000*t[3][6]]
    evaluation+=16[1000000*t[4][0]+100000*t[4][1]+10000*t[4][2]]
    evaluation+=16[1000000*t[4][1]+100000*t[4][2]]
    evaluation+=16[1000000*t[4][2]]
    evaluation+=16[1000000*t[4][4]+100000*t[4][5]+t[4][6]]
    evaluation+=16[1000000*t[4][5]+100000*t[4][6]]
    evaluation+=16[1000000*t[4][6]]
    evaluation+=16[1000000*t[5][0]+100000*t[5][1]+10000*t[5][2]]
    evaluation+=16[1000000*t[5][1]+100000*t[5][2]]
    evaluation+=16[1000000*t[5][2]]
    evaluation+=16[1000000*t[5][3]+100000*t[5][4]+10*t[5][5]+t[5][6]]
    evaluation+=16[1000000*t[5][4]+100000*t[5][5]+t[5][6]]
    evaluation+=16[1000000*t[5][5]+100000*t[5][6]]
    evaluation+=16[1000000*t[5][6]]
    evaluation+=15[1000000*t[1][0]+100000*t[1][1]+10000*t[1][2]]
    evaluation+=15[1000000*t[2][2]+100000*t[2][3]+10000*t[2][4]]
    evaluation+=15[1000000*t[3][5]+t[4][6]]
evaluation += 15[10000*t[4][0]+1000*t[3][1]+100*t[2][2]+1]
  \rightarrow 0*t[1][3]+t[0][4]

evaluation += 15[10000*t[5][2]+1000*t[4][3]+100*t[3][4]+1]
  \rightarrow 0*t[2][5]+t[1][6]

evaluation += 14[27*t[0][3]+9*t[1][4]+3*t[2][5]+t[3][6]]

#valid moves
order=[3, 2, 4, 1, 5, 0, 6]

def validMoves(intable):
    global order
    moves=[[]
    for col in order:
        for row in range(6):
            if intable[row][col]==0:
                moves.append([row, col])
                break
    return moves

#Alpha Beta Pruning Search Algorithm

def alphabetaPruning(intable, depth):
    def ab(intable, depth, alpha, beta):
        values=[]; v=-10000000
        for a,s in validMoves(intable):
            intable[a][s]=1
            v=max(v, abmin(intable, depth-1, alpha, beta))
            intable[a][s]=0
        largest=max(values)
        dex=values.index(largest)
        return [dex, largest]

    def abmax(intable, depth, alpha, beta):
        moves=validMoves(intable)
        if(depth==0 or not moves):
            return eval(intable)
        v=-10000000
        for a,s in moves:
            intable[a][s]=1
            v=max(v, abmin(intable, depth-1, alpha, beta))

    return ab(intable, depth, -10000000, 100000000)
APPENDIX B. PYTHON CODE

```python
intable[a][s]=0
if v >= beta: return v
alpha=max(alpha, v)
return v

def abmin(intable, depth, alpha, beta):
    moves=validMoves(intable)
    if (depth==0 or not moves):
        return eval(intable)
    v+=10000000
    for a,s in moves:
        intable[a][s]=2
        v=min(v, abmax(intable, depth-1, alpha, beta))
        intable[a][s]=0
        if v <= alpha: return v
    beta=min(beta, v)
    return v

return ab(intable, depth, -10000000, +10000000)

#returns the minutes*60 + seconds in the actual time
def t(): return ((time.gmtime()[4])*60)+time.gmtime()[5]

#Iterative Deepening Search Algorithm
def iterDeepening(intable):
    global order

timeout=t()+19
depth=1
res=alphabetaPruning(intable, depth)
while True:
    tStart=t()
    if abs(res[1])>5000: #terminal node
        print "Nearly done!"
        return res[0]
    tmp=res[0]
    #changing the order in considering moves
    while tmp!=0:
        order[tmp-1],order[tmp]=order[tmp],order[tmp-1]
        tmp-=1
        depth+=1
    res=alphabetaPruning(intable, depth)
    tEnd=t()
```

#returns the minutes*60 + seconds in the actual time
def t(): return ((time.gmtime()[4])*60)+time.gmtime()[5]

#Iterative Deepening Search Algorithm
def iterDeepening(intable):
    global order

    timeout=t()+19
    depth=1
    res=alphabetaPruning(intable, depth)
    while True:
        tStart=t()
        if abs(res[1])>5000: #terminal node
            print "Nearly done!"
            return res[0]
        tmp=res[0]
        #changing the order in considering moves
        while tmp!=0:
            order[tmp-1],order[tmp]=order[tmp],order[tmp-1]
            tmp-=1
            depth+=1
        res=alphabetaPruning(intable, depth)
        tEnd=t()
runTime=tEnd-tStart
if timeout < tEnd+(4 * runTime) or depth > 42:
    print "DEPTH", depth
    mylcd.lcd_display_string(" → ", 2)
    mylcd.lcd_display_string("DEPTH %d : " % depth,
    → 2)
    return res[0]

#similar to validMoves but with None if the move of that
col in not valid
def humanMoves(intable=[]):
    cols=[]; rows=[]
    for col in range(7):
        for row in range(6):
            if intable[row][col]==0:
                cols.append(col)
                rows.append(row)
            break
    return cols, rows

isNum=re.compile("[0-9]")

#human move for a given col
def hmove(intable, x):
    cols, rows = humanMoves(intable)
    if isNum.match(x)==None and x!='': x=int(x)-1
    while x not in cols:
        print "INVALID MOVE!!!"
        x=readserial()
    if isNum.match(x)==None and x!='': x=int(x)-1
    intable[rows[cols.index(x)]]]=2

#moves in slot x according to valid moves function
def move(intable,x,who):
    val=validMoves(intable)
    intable[val[x][0]][val[x][1]]=who
coltoplay = val[x][1]+1
    print "Robot plays: ", coltoplay
    mylcd.lcd_display_string(" → ", 3)
    mylcd.lcd_display_string("Robot played %d" % coltoplay,
    → 3)
    ser.write(str(coltoplay))  
    # Sends the
    → next move to Arduino
#Function to read from USB

def readserial():
    x = ser.read()
    time.sleep(1)
    print "Player plays: ", x
    ser.reset_input_buffer()
    return x

#GAME
agenttally=0; playertally=0

while True:
    mylcd.lcd_display_string("", 1)
    mylcd.lcd_display_string("", 2)
    mylcd.lcd_display_string("", 3)
    agent=0; player=0;
    table = [[0, 0, 0, 0, 0, 0, 0],
             [0, 0, 0, 0, 0, 0, 0],
             [0, 0, 0, 0, 0, 0, 0],
             [0, 0, 0, 0, 0, 0, 0],
             [0, 0, 0, 0, 0, 0, 0]]
    table.reverse()
    14=[int(i) for i in open("eval/ev_table.txt")]

    15={}
    for li in open("eval/5ki.txt"):
        tok=li.split()
        15[int(tok[0])]=int(tok[1])

    16={}
    for li in open("eval/6ki.txt"):
        tok=li.split()
        16[int(tok[0])]=int(tok[1])

    17={}
    for li in open("eval/7ki.txt"):
        tok=li.split()
        17[int(tok[0])]=int(tok[1])

    #the player plays first
draw(table)
while validMoves(table):
    print "Your turn!"
    mylcd.lcd_display_string("Your turn!", 1)
    n = readserial()  # Receives the played column from Arduino
    mylcd.lcd_display_string("", 3)
    mylcd.lcd_display_string("You played %d %
    \rightarrow float(n), 3)
hmove(table, n)
draw(table)
if win(table)==2:
    player+=1
draw(table)
    print "You have won!"
    mylcd.lcd_display_string(""
    \rightarrow ", 1)
    mylcd.lcd_display_string("You Have won!", 1)
    playertally+=1
    break

cStart=t()
print "Hmmm let me think ?!??!"
mylcd.lcd_display_string("", 1)
mylcd.lcd_display_string("Thinking...", 1)
move(table, iterDeepening(table), 1)
draw(table)
tthink = t() - cStart
print "After ", tthink, " seconds thinking!"
mylcd.lcd_display_string("", 2)
mylcd.lcd_display_string("After %d seconds!" %
    \rightarrow tthink, 2)
if win(table)==1:
    agent+=1
    print "Robot has won!"
    mylcd.lcd_display_string(""
    \rightarrow ", 1)
    mylcd.lcd_display_string("Robot has won!", 1)
time.sleep(5)
agenttally+=1
break
if agent==player:
    print "DRAW"
    mylcd.lcd_display_string(""
    mylcd.lcd_display_string("Draw!", 1)
else:
    print "ROBOT",agenttally,"",playertally," PLAYER"
    mylcd.lcd_display_string(""
    mylcd.lcd_display_string("ROBOT %d : " %
    mylcd.lcd_display_string("PLAYER %d" %
    mylcd.lcd_display_string("ROBOT %d : " %
    mylcd.lcd_display_string("PLAYER %d"
    mylcd.lcd_display_string(""
    mylcd.lcd_display_string("Reset the game!", 1)
    time.sleep(10)

# CON4UTILS.PY
# All functions work on reversed table
import re
import serial

# from ConFour import readserial
# represents the table

def draw(a=[]):
    def tc(a):
        if a==0: return " "
        if a==1: return "X"
        if a==2: return "Y"

    print "
    --------------------------- 
    print "
    [ %s ] [ %s ] [ %s ] [ %s ] [ %s ] [ %s ]
    % (tc(a[5][0]), tc(a[5][1]), tc(a[5][2]), tc(a[5][3]),
    tc(a[5][4]), tc(a[5][5]), tc(a[5][6]))
    print "
    [ %s ] [ %s ] [ %s ] [ %s ] [ %s ] [ %s ]
    % (tc(a[4][0]), tc(a[4][1]), tc(a[4][2]), tc(a[4][3]),
    tc(a[4][4]), tc(a[4][5]), tc(a[4][6]))
    print "
    [ %s ] [ %s ] [ %s ] [ %s ] [ %s ] [ %s ]
    % (tc(a[3][0]), tc(a[3][1]), tc(a[3][2]), tc(a[3][3]),
    tc(a[3][4]), tc(a[3][5]), tc(a[3][6]))
    print "
    [ %s ] [ %s ] [ %s ] [ %s ] [ %s ] [ %s ]
    % (tc(a[2][0]), tc(a[2][1]), tc(a[2][2]), tc(a[2][3]),
    tc(a[2][4]), tc(a[2][5]), tc(a[2][6]))
    print "
    [ %s ] [ %s ] [ %s ] [ %s ] [ %s ] [ %s ]
    % (tc(a[1][0]), tc(a[1][1]), tc(a[1][2]), tc(a[1][3]),
    tc(a[1][4]), tc(a[1][5]), tc(a[1][6]))
    print "
    [ %s ] [ %s ] [ %s ] [ %s ] [ %s ] [ %s ]
    % (tc(a[0][0]), tc(a[0][1]), tc(a[0][2]), tc(a[0][3]),
    tc(a[0][4]), tc(a[0][5]), tc(a[0][6]))
    print "
    %%%%%%%%%%%%% %
    print "
    1 2 3 4 5 6 7 
    \n\n"

    # computes string hash value of the state of the table
    def toHash(table):
        out="";
        for j in range(7):
            for i in range(6):
                if table[i][j]==0: break
                else: out+=str(table[i][j])
                out+="|"
        return out

    # determines whether there is a winner
    def win(intable=[]):
        w1=[1,1,1,1]
w2=[2,2,2,2]

#horizontal
for i in range(6):
    for j in range(4):
        if [intable[i][j],intable[i][j+1],intable[i][j+2],intable[i][j+3]]==w1:
            return 1
        if [intable[i][j],intable[i][j+1],intable[i][j+2],intable[i][j+3]]==w2:
            return 2

#vertical
for i in range(7):
    for j in range(3):
        if [intable[j][i],intable[j][i+1],intable[j+2][i],intable[j+3][i]]==w1:
            return 1
        if [intable[j][i],intable[j][i+1],intable[j+2][i],intable[j+3][i]]==w2:
            return 2

#left to right
for j in range(3):
    for i in range(4):
        if [intable[j][i],intable[j+1][i+1],intable[j+2][i+2],intable[j+3][i+3]]==w1:
            return 1
        if [intable[j][i],intable[j+1][i+1],intable[j+2][i+2],intable[j+3][i+3]]==w2:
            return 2

#right to left
for j in range(3):
    for i in range(6,2,-1):
        if [intable[j][i],intable[j+1][i-1],intable[j+2][i-2],intable[j+3][i-3]]==w1:
            return 1
        if [intable[j][i],intable[j+1][i-1],intable[j+2][i-2],intable[j+3][i-3]]==w2:
            return 2
Appendix C

Overall System Wiring Diagram

Figure C.1. Overall system wiring diagram. (Created in Fritzing)
Appendix D

Multiplexer-Photoreflector Wiring Diagram

Figure D.1. Multiplexer-Photoreflector Wiring Diagram. (Created in Edraw)