Department of Electrical and Computer Engineering

Howard University

Washington, DC 20059

EECE 404: Senior Design II

Team Terminator

<u>2017-18 Final Report</u>

Submitted by:

Cory Bethrant, Maxime Keita

Faculty Advisor:Dr. Charles Kim

Instructor: Dr. Charles Kim

Date: April 24, 2018

Executive Summary

The project was born to start a vertically integrated project that would seek to create a AI-based game-playing robot for a fraction of the price of the expected cost. It had been observed that there are many libraries in existence to help with the creation of AI, specifically in our case Tensorflow, Minimax Algorithms and OpenCV (for Vision).

Table of Contents

Introduction and Background

The purpose of this project to build an intelligent robot that implemented in tic tac toe games. This robot will understand the rules and ways to play the game and make every decision based on the intelligence of itself. To test the intelligence of this robot, the opponent must be a human. This project is primarily focused on the feasibility of a AI-based game playing system, and if one can be created cheaply.

Cost

The parts required for the Arms assembly average at around $40 each. The Robotic Arm Edge, which is the base structure costs $40, while the Raspberry Pi is $49.99. The webcam for OpenCV costed $29.99. Total cost is vastly cheaper than expected.

Problem Statement

Our problem statement verbatim is ***"Is it possible to create an AI Robot capable of playing games in the physical space for cheap?"***

We intend to produce an AI robot using mainly existing components such as the Robotic Arm Edge, the Raspberry Pi, the Arduino, and a circuit-board camera compatible with the Arduino. All the components interface with each other using OpenCV and C++.

Current Status of Art

A paper based on this project was presented at the International Conference on Computer Vision and Robotics 2012 held at Bhubaneswar, India. There is no machine learning component in this device. It uses a brute force algorithm to compete.

Design Requirements

The arm has to be able to see the entire board before and after moves. The arm has to be cheap and capable of moving pieces independently.

Design Solution

As far as the physical arm goes, our design intends to use the Robotic Arm Edge.

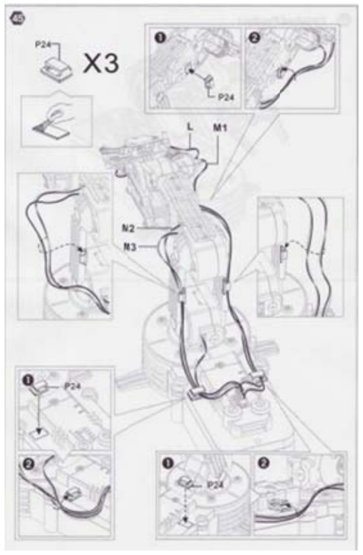


Figure 0.1

The Raspberry Pi has a digital pin structure represented by the following map:

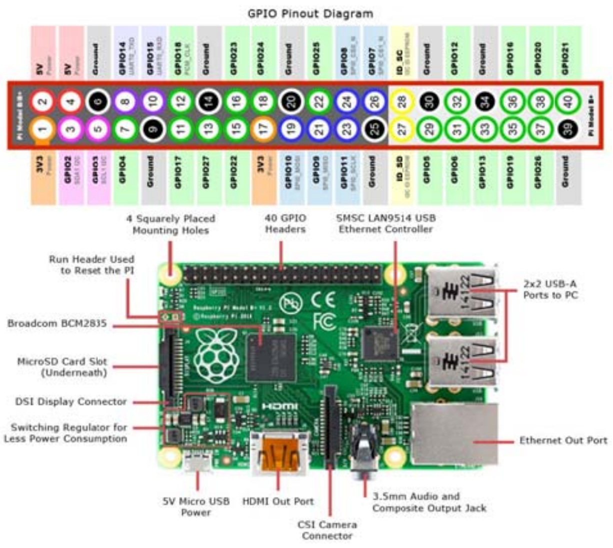
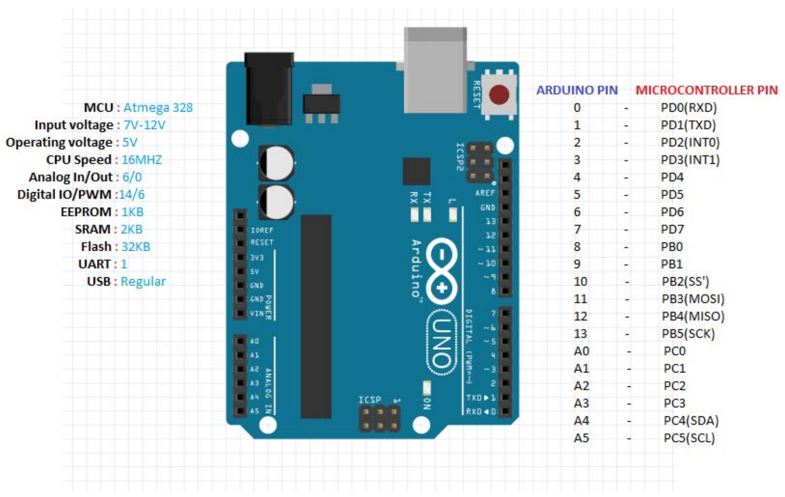

GPIO Pinout Diagram
4 Squarely Placed Mounting Holes
40 GPIO Headers
SMSC LAN9514 USB Ethernet Controller
Run Header Used to Reset the PI
Broadcom BCM2835
MicroSD Card Slot (Underneath)
DSI Display Connector
Switching Regulator for Less Power Consumption
2x2 USB-A Ports to PC
Ethernet Out Port
5V Micro USB Power
HDMI Out Port
CSI Camera Connector
3.5mm Audio and Composite Output Jack

Figure 0.2

The Arduino Uno has a digital pin structure represented by the following map:



MCU : Atmega 328
Input voltage : 7V-12V
Operating voltage : 5V
CPU Speed : 16MHZ
Analog In/Out : 6/0
Digital IO/PWM : 14/6
EEPROM : 1KB
SRAM : 2KB
Flash : 32KB
UART : 1
USB : Regular

ARDUINO PIN		MICROCONTROLLER PIN
0	-	PD0(RXD)
1	-	PD1(TXD)
2	-	PD2(INT0)
3	-	PD3(INT1)
4	-	PD4
5	-	PD5
6	-	PD6
7	-	PD7
8	-	PB0
9	-	PB1
10	-	PB2(SS')
11	-	PB3(MOSI)
12	-	PB4(MISO)
13	-	PB5(SCK)
A0	-	PC0
A1	-	PC1
A2	-	PC2
A3	-	PC3
A4	-	PC4(SDA)
A5	-	PC5(SCL)

Figure 0.3

Assembled Arm before controllers added:

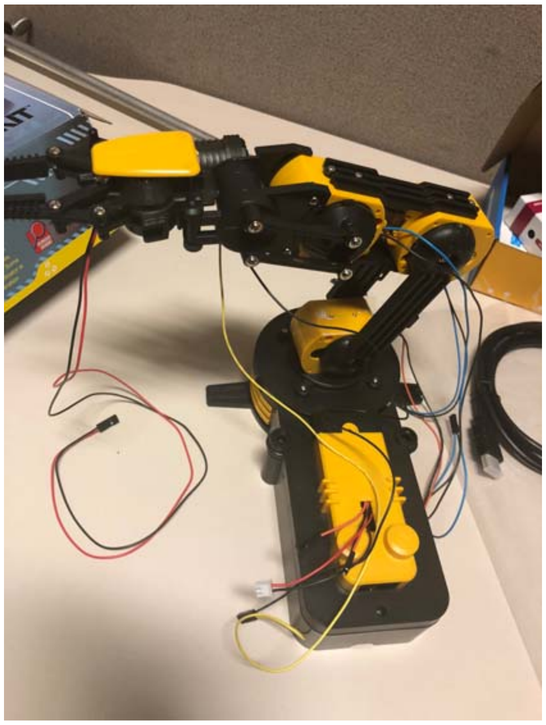


Figure 0.4

Project's Final Goal and Deliverable

This project's final goal is to create a perfectly functioning arm capable of playing a human opponent in a variety of board games in the physical world. The arm should be cheap and feasible to assemble.

Project's Spring 2018 Target Goal

To create a perfectly functioning arm capable of playing a human opponent in tic-tac-toe in the physical world. The arm should be cheap and feasible to assemble.

Implementation, Testing and Evaluation

The first step was to assemble the arm and mount the camera. Then to wire the system together. Afterwards the next step was to develop a MiniMax algorithm to use recursive machine learning to defeat the human opponent. Then OpenCV was used to feed in real-time data to algorithm to determine where the board is and where pieces are located. (Pieces are required to be in frame) The last modify MiniMax Algorithm to use the arm and OpenCV events instead of console for game output. We were able to completely assemble everything and code the minimax algorithm to completion. The final steps to be finished to complete the interface between components, OpenCV-based recognition and the piece moving functions.

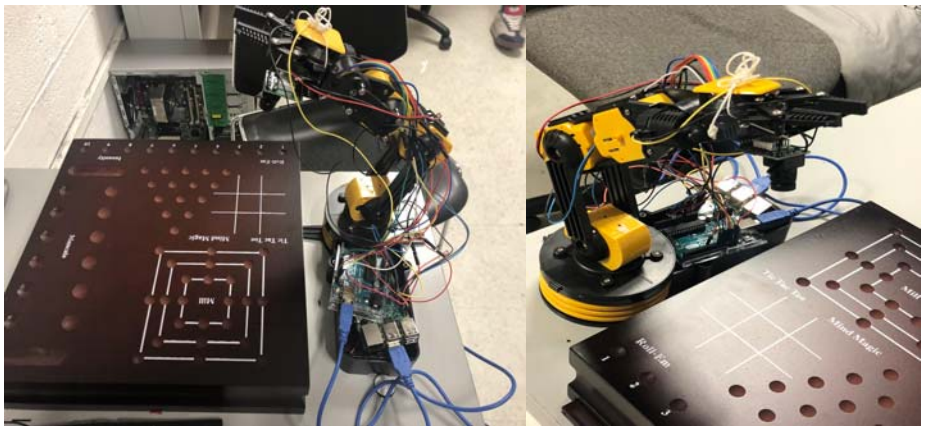


Figure 0.5

Conclusion

A significant amount of knowledge has been gained from this project. Specifically when it comes to the creation of a physical prototype. The ability to move from idea to functional prototype is invaluable to the thousands of companies based in the US. The project also gave experience on presenting/demos, which are required to drive an engineer's inventions.First of all, it helped us in managing a project and sharing the load of work. Secondly, it showed that simple but well adapted algorithms are often more efficient than more general and complex ones. Lastly, it gave us the opportunity to work at the interface between three related disciplines: Artificial Intelligence, Vision and Robotics, which lead to very interesting issues when studied together. This project is definitely one that can be continued and will be viewed as enjoyable by VIP participants.

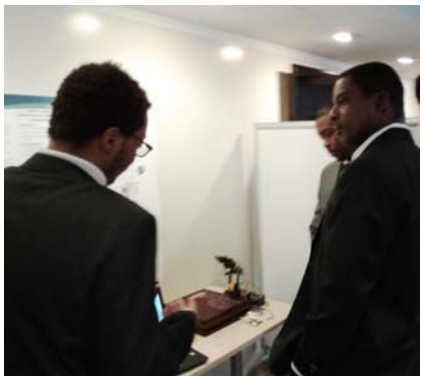


Figure 0.6

Recommendation for Future work

To complete the interfacing between components, game recognition and porting over the MiniMax algorithm to function with the external components instead of the console. After the final step would be improving to be able to play additional games, mainly Chess.

References

A Simple Autonomous Robotic Manipulator for playing Chess against any opponent in Real Time http://www.nandanbanerjee.com/files/ICCVR-08AUG12-011%20paper.pdf

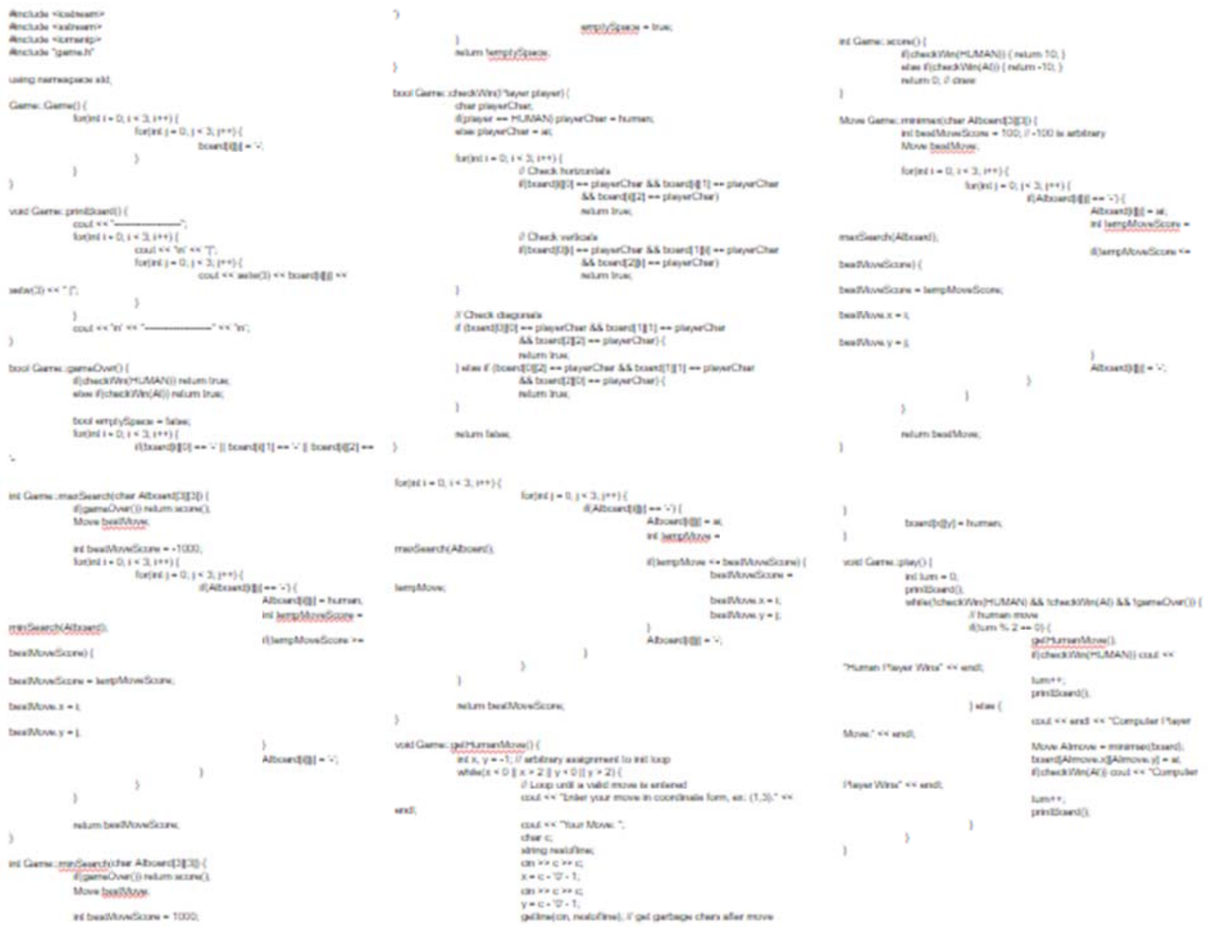
Deep Learning Machine Teaches Itself Chess in 72 Hours, Plays at International Master Level https://www.technologyreview.com/s/541276/deep-learning-machine-teaches-itself-chess-in-72-hours-plays-at-international-master/

Raspberry Turk

http://www.raspberryturk.com/

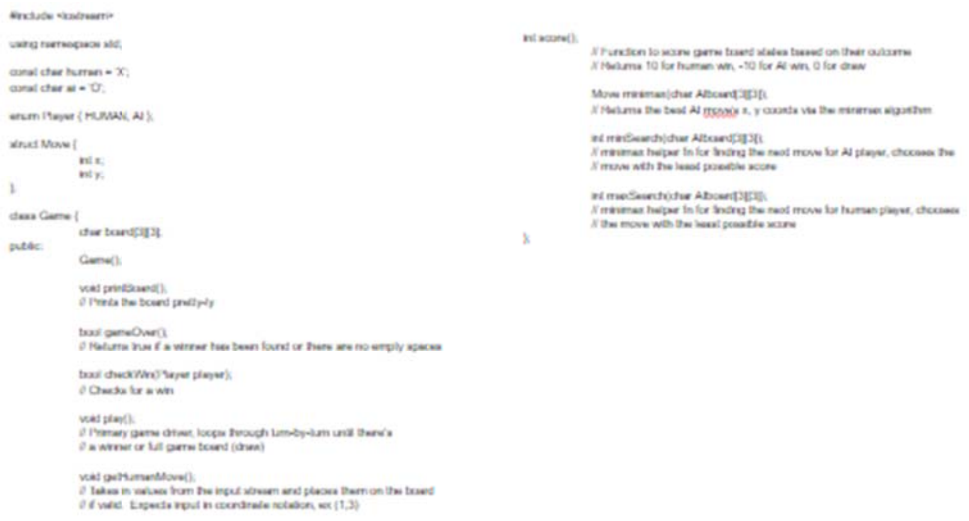
Source Code

game.cpp



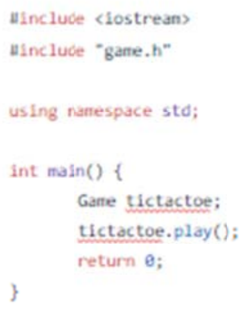
```
#include <iostream>
#include <sstream>
#include <iomanip>
#include "game.h"

using namespace std;

Game::Game() {
        for(int i = 0; i < 3; i++) {
                for(int j = 0; j < 3; j++) {
                        board[i][j] = '-';
                }
        }
}

void Game::printBoard() {
        cout << "-------------------";
        for(int i = 0; i < 3; i++) {
                cout << '\n' << "|";
                for(int j = 0; j < 3; j++) {
                        cout << setw(3) << board[i][j] <<
setw(3) << " |";
                }
        }
        cout << '\n' << "-------------------" << '\n';
}

bool Game::gameOver() {
        if(checkWin(HUMAN)) return true;
        else if(checkWin(AI)) return true;

        bool emptySpace = false;
        for(int i = 0; i < 3; i++) {
                if(board[i][0] == '-' || board[i][1] == '-' || board[i][2] ==
'-')
                        emptySpace = true;
        }
        return !emptySpace;
}

bool Game::checkWin(Player player) {
        char playerChar;
        if(player == HUMAN) playerChar = human;
        else playerChar = ai;

        for(int i = 0; i < 3; i++) {
                // Check horizontals
                if(board[i][0] == playerChar && board[i][1] == playerChar
                        && board[i][2] == playerChar)
                        return true;

                // Check verticals
                if(board[0][i] == playerChar && board[1][i] == playerChar
                        && board[2][i] == playerChar)
                        return true;
        }

        // Check diagonals
        if (board[0][0] == playerChar && board[1][1] == playerChar
                && board[2][2] == playerChar) {
                return true;
        } else if (board[0][2] == playerChar && board[1][1] == playerChar
                && board[2][0] == playerChar) {
                return true;
        }

        return false;
}

int Game::maxSearch(char AIboard[3][3]) {
        if(gameOver()) return score();
        Move bestMove;

        int bestMoveScore = -1000;
        for(int i = 0; i < 3; i++) {
                for(int j = 0; j < 3; j++) {
                        if(AIboard[i][j] == '-') {
                                AIboard[i][j] = human;
                                int tempMoveScore =
minSearch(AIboard);
                                if(tempMoveScore >=
bestMoveScore) {
bestMoveScore = tempMoveScore;
bestMove.x = i;
bestMove.y = j;
                                }
                                AIboard[i][j] = '-';
                        }
                }
        }

        return bestMoveScore;
}

int Game::minSearch(char AIboard[3][3]) {
        if(gameOver()) return score();
        Move bestMove;

        int bestMoveScore = 1000;
        for(int i = 0; i < 3; i++) {
                for(int j = 0; j < 3; j++) {
                        if(AIboard[i][j] == '-') {
                                AIboard[i][j] = ai;
                                int tempMove =
maxSearch(AIboard);
                                if(tempMove <= bestMoveScore) {
                                        bestMoveScore =
tempMove;
                                        bestMove.x = i;
                                        bestMove.y = j;
                                }
                                AIboard[i][j] = '-';
                        }
                }
        }

        return bestMoveScore;
}

void Game::getHumanMove() {
        int x, y = -1; // arbitrary assignment to init loop
        while(x < 0 || x > 2 || y < 0 || y > 2) {
                // Loop until a valid move is entered
                cout << "Enter your move in coordinate form, ex: (1,3)." <<
endl;
                cout << "Your Move: ";
                char c;
                string restofline;
                cin >> c >> c;
                x = c - '0' - 1;
                cin >> c >> c;
                y = c - '0' - 1;
                getline(cin, restofline); // get garbage chars after move
        }
        board[x][y] = human;
}

int Game::score() {
        if(checkWin(HUMAN)) { return 10; }
        else if(checkWin(AI)) { return -10; }
        return 0; // draw
}

Move Game::minimax(char AIboard[3][3]) {
        int bestMoveScore = 100; // -100 is arbitrary
        Move bestMove;

        for(int i = 0; i < 3; i++) {
                for(int j = 0; j < 3; j++) {
                        if(AIboard[i][j] == '-') {
                                AIboard[i][j] = ai;
                                int tempMoveScore =
maxSearch(AIboard);
                                if(tempMoveScore <=
bestMoveScore) {
bestMoveScore = tempMoveScore;
bestMove.x = i;
bestMove.y = j;
                                }
                                AIboard[i][j] = '-';
                        }
                }
        }

        return bestMove;
}

void Game::play() {
        int turn = 0;
        printBoard();
        while(!checkWin(HUMAN) && !checkWin(AI) && !gameOver()) {
                // human move
                if(turn % 2 == 0) {
                        getHumanMove();
                        if(checkWin(HUMAN)) cout <<
"Human Player Wins" << endl;
                        turn++;
                        printBoard();
                } else {
                        cout << endl << "Computer Player
Move:" << endl;
                        Move AImove = minimax(board);
                        board[AImove.x][AImove.y] = ai;
                        if(checkWin(AI)) cout << "Computer
Player Wins" << endl;
                        turn++;
                        printBoard();
                }
        }
}
```

game.h



```
#include <iostream>

using namespace std;

const char human = 'X';
const char ai = 'O';

enum Player { HUMAN, AI };

struct Move {
                int x;
                int y;
};

class Game {
                char board[3][3];
public:
                Game();

                void printBoard();
                // Prints the board pretty-ly

                bool gameOver();
                // Returns true if a winner has been found or there are no empty spaces

                bool checkWin(Player player);
                // Checks for a win

                void play();
                // Primary game driver, loops through turn-by-turn until there's
                // a winner or full game board (draw)

                void getHumanMove();
                // Takes in values from the input stream and places them on the board
                // if valid.  Expects input in coordinate notation, ex (1,3)

                int score();
                // Function to score game board states based on their outcome
                // Returns 10 for human win, -10 for AI win, 0 for draw

                Move minimax(char AIboard[3][3]);
                // Returns the best AI move's x, y coords via the minimax algorithm

                int minSearch(char AIboard[3][3]);
                // minimax helper fn for finding the next move for AI player, chooses the
                // move with the least possible score

                int maxSearch(char AIboard[3][3]);
                // minimax helper fn for finding the next move for human player, chooses
                // the move with the least possible score
};
```

play.cpp



```
#include <iostream>
#include "game.h"

using namespace std;

int main() {
        Game tictactoe;
        tictactoe.play();
        return 0;
}
```