**DEPARTMENT OF ELECTRONIC & TELECOMMUNICATION**

**ENGINEERING UNIVERSITY OF MORATUWA**

**EN1093: LABORATORY PRACTICE – 1**

****

**TIC TAC TOE PLAYING ROBOT**

170112K - W.H.O.G. DE SILVA

170130M – A.D.V.D.R. DHARMARATHNE

170131R – K.D.M.A. DHARMASIRI

170146R – B.M.I.P. DISSANAYAKE

**ABSTRACT**

Objective of the project is to design and build a robot that can play the famous “tic tac toe” game against a human player. The robot is controlled by the pic16F877A microcontroller and the playing area is a paper (dimensions) with nine squares in it.

The mechanism to get the input of human player is via a keypad with 12 keys. The output of the robot player is drawn on the paper using a 2D drawing mechanism with 2 linear actuators and a stencil. 2 DC motors and a hobby servo motor is used for the actuators and the stencil.

The game tic tac toe can be played with different algorithms. We used a simple non-recursive algorithm to determine the move of the robotic player based on the moves by the human player, which will minimize the winning chances of the human player. The algorithm is designed in such a way that the robot can exploit the blunders done by the human player, to win the game.

The robot devices an advanced algorithm to play a famous and simple game, in the meantime, give a familiar experience to the player by executing the game on a paper.

**ACKNOWLEDGEMENT**

We would like to offer our gratitude to every single person who helped us to achieve this. Most important personal we would like to thank is our supervisor Mr. Janith kalpa who supported us to learn the basic concepts which are essential and boosted us to self-learn about the required stuffs needed for this project. It was very helpful to clear the problems and doubts we had faced and gave us new perspective about them.

We pay our gratitude to all the lecturers, instructors and other academic staff who intimately welcomed us to share their knowledge and experiences. We are very grateful to the personal who are in charge of laboratories for allowing us to use the laboratories when needed and supported to solve the technical problems.

**CONTENT**

Abstract

Acknowledgement

1. Introduction

2. Specifications

3. How to play

4. Method

4.1 Components

4.2 Schematic diagrams

4.3 Input output devices

4.4 Motor controller

4.5 Power supply

4.6 Servo controller

4.7 Algorithm

5. Results

6. Discussion

7. References

8. Appendices

**1. INTRODUCTION**

Tic Tac Toe playing robot is a robot which can play tic tac toe game interactively with another player. Mainly this robot consists mainly of two parts. An input keyboard and a two-axis plotter. The human player can input his moves using the keypad, and the robot draws its move on a paper using the plotter. In general, a human player can play tic tac toe with this robot.

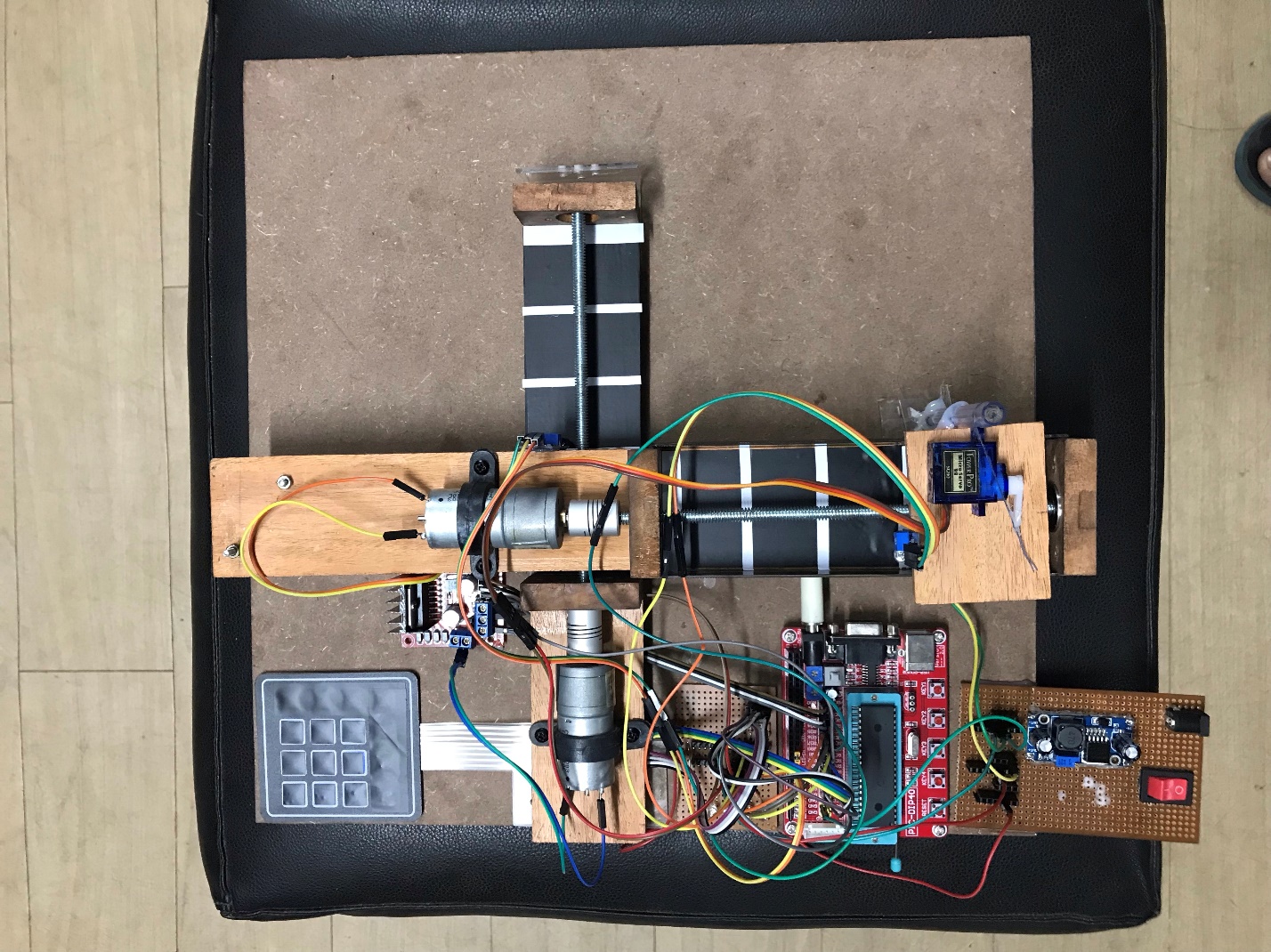
The machine is based on PIC 16F877a microcontroller. The algorithm allows the human player win if played with proper tactics. If not, the machine has a high chance of beating the human player.

**2. SPECIFICATIONS**

This robot is 36cm long and 36cm wide. It consists of two moving arms which allows the robot to move to the corresponding position. Each arm is 30cm long. The input keyboard consists of 9keys. Each key is assigned for different positions.

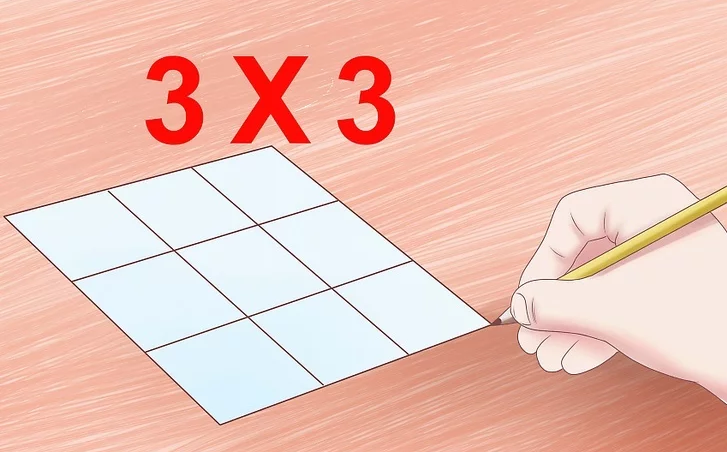
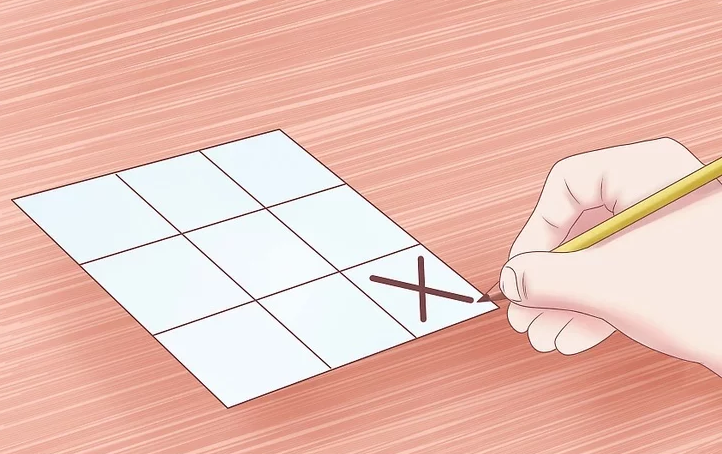
The playing area which consists of one large square divided in to equal 9 squares, is 9cm \* 9 cm. One small square which represent one position is 3cm \* 3cm.

Altogether this robot is weight around 1000g.

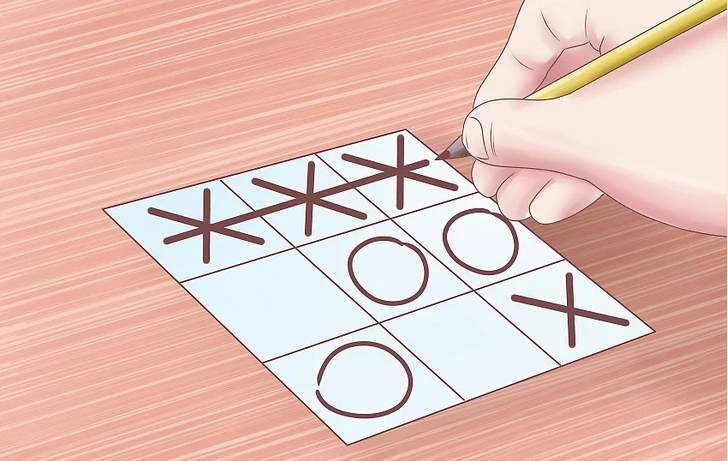


**3. HOW TO PLAY**

Tic Tac Toe is a two-player game. In this case it’s a human player and the robot. In this game there are two symbols (say ‘0’ and ‘\*’). Human player is assigned ‘0’, and the robot will play with ‘\*’. The human player can choose either to play first or second. In the playing area there are 9 squares. A player can mark their position by drawing his/him symbols in one of these squares. And the chances to play will be given alternatively to the human player and the robot. To win, a player should acquire 3 adjacent squares vertically, horizontally or diagonally. If both the players are unable to get 3 adjacent squares as mentioned above, then it’s a draw.



                          Play areaIn each move player can acquire one square

****

Alternatively, players continue                                            A player can win by acquiring 3 squares

**4. METHOD**

**4.1 Schematic design**

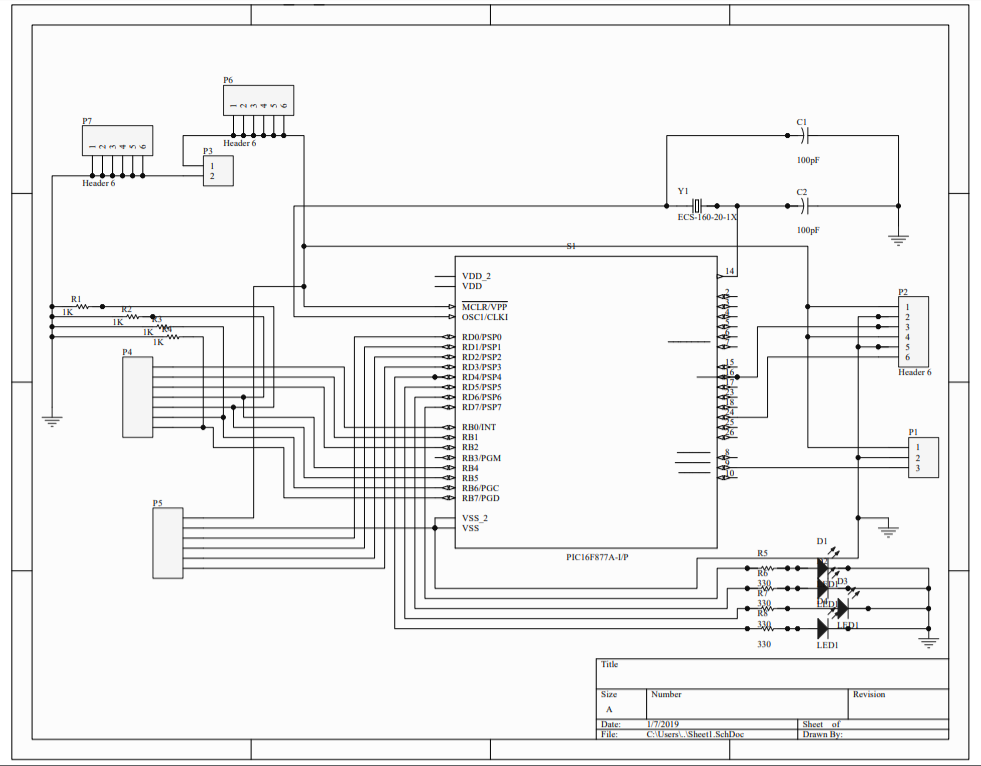
Schematic diagram consists of 3 main parts.

1.       Power supply

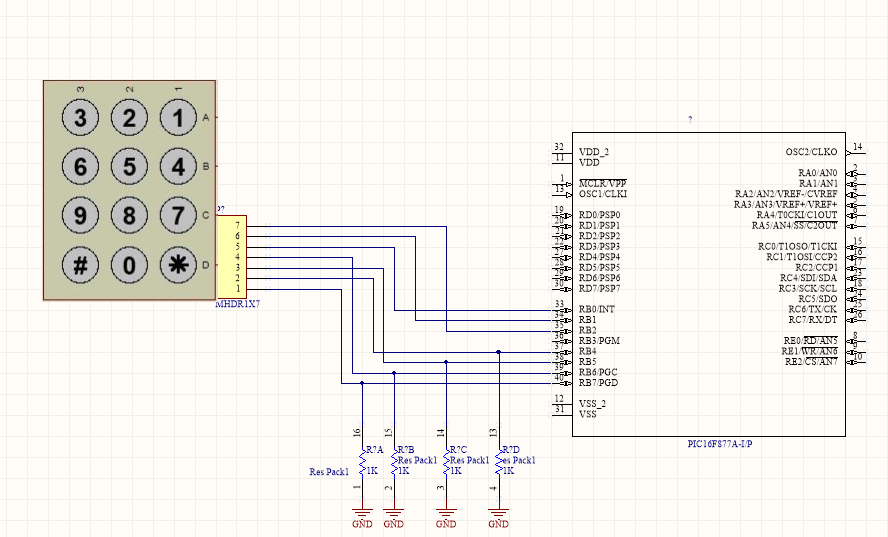
2.       Input - keyboard

3.       Output – robot arms

Also, this whole system is controlled through a development board with a pic 16F877A.



Whole schematic diagram



Keypad interfacing schematic

**4.2 Components**

The components that were used to build the robot can be separated into three parts based on their task.

       4.2.1 Controllers

        4.2.2 Actuators

        4.2.3 Input devices

4.2.1 Controllers

The Microchip PIC16F877A microcontroller was selected by us to implement our algorithm. As implementing an optimum algorithm was one of the main outcomes of the project, the algorithm had to be optimized thoroughly in order to get the maximum out of the available resources. Key specifications of the chosen 40-pin microcontroller are stated below. More details are available on the data sheet attached under appendices.

• Program memory:   14.3 KB

• Data Memory:   368 Bytes

• EEPROM:   256 Bytes

• Clock:   4MHz

For the ease of use, we devised a development board containing the pic microcontroller, the crystal, capacitors, reset switch, some indicating LEDs and pin headers for input/ output pins. We used PICKIT2 programmer to upload the program into the pic microcontroller.

4.2.2 Actuators

To draw the playing move of the robot player, we needed a mechanism for a stencil to move in the 2D plane with respect to two axes, x and y.

For that, we used 2 linear actuators built with thread bars and high-speed motors, placed perpendicular to each other, one on the top of the other.

Conversion of the rotational motion of the motors to a linear motion in the writing arms was very important. Shown below is a picture of one actuator used.

As the motor rotates, the thread bar also rotates. But as the bolt in the thread bar is constrained from rotating, it moves linearly along the thread bar. And thus, we have converted the rotational motion of the motor into a linear motion of the bolt.

Then we installed a second similar actuator on the top of that moving bolt. So, our mechanism has the ability to move in the 2D plane. The stencil attached to a small servo motor moves on the top of the second actuator, which can as a result, move to any position in the x-y plane. When moved to the required square, it can place it’s move using the stencil.



To build the Mechanical part of the Tic Tac Toe Machine following items and equipment were gathered.

* 1.     High Speed Motors                            X 2
* 2.     7 ½” Thread Bars                                X 2
* 3.     Wood Strips 10” Long                        X 2
* 4.     2 X 2 Wood Strips                               X 6
* 5.     Nuts                                               X 6
* 6.     Bearing                                              X 4
* 7.     Motor Thread Bar Connectors           X 2
* 8.     Caster Wheels                                   X 2

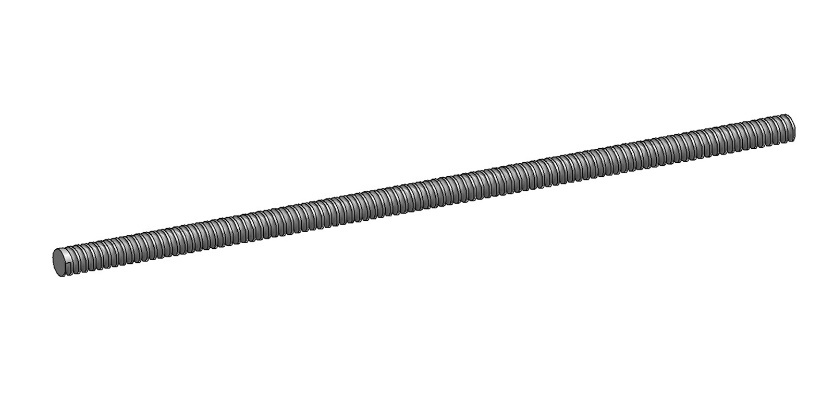
Firstly, we consider about building a single unit and process is repeated on the Second Unit. 2 X 2 Wood Strips must be extruded until a bearing can be fit into it. For that purpose, a collar is cut inside the wood strip also which will prevent bearing from falling down from the other side.



couplers used to connect motor shaft and thread bar

Bearings installed between frame and thread bar

Bearings indulge Two of 2 X 2 Wood Strips were placed and glued to 10” Long Wood Strip with a distance of 6” starting from one end. Now we take the Thread bar and lathe one end of it as it must be fit to the connector which connects Motor and Thread bar.



Thread bar used to make the linear actuators

Now Thread bar is inserted from one bearing and then making it run through two nuts and connected to the other bearing. The lathe end of the Thread bar is connected to the Connector and Motor is connected to the Connector from other side. We glue a 2 X 2 Wood Strip on the two nuts.

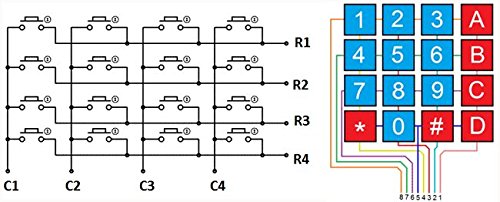
This is the simplest process to build a Single Slider Unit and it is repeated to build the second Slider Unit.

Later, one slider unit is placed on the other slider unit which will look like a Cross. When the Motors are powered Two sliders will move in a 2D plane, in X & Y directions.

4.2.3 Input devices

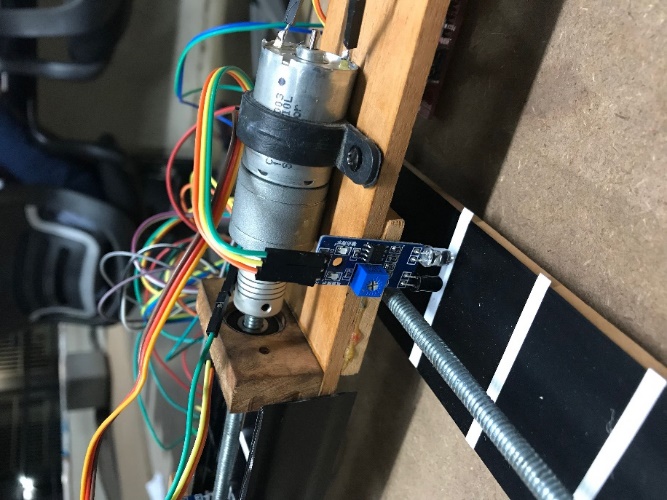
The main input device are the keypad and IR sensors. Keypad is used to input the moves of the human player. And the IR sensors are used to find the current location of the writing tool.

Keypad is with 12 buttons with different functionalities assigned to each one of them. The buttons on a keypad are arranged in rows and columns. Under these buttons there are membrane switches. Pressing a button closes the switch between a column and a row trace, allowing current to flow between a column pin and a row pin. By detecting the row and column through which the current flow, we can decide which button is pressed.



Keypad structure

The IR sensor is made of an IR LED and an IR photo transistor. The photo transistor detects the intensity of reflected IR light. As white and black colors reflect IR with different intensities, it can be used to differentiate between the two colors. The sensor threshold can be adjusted using a potentiometer.



IR sensor

**4.3 Power supply**

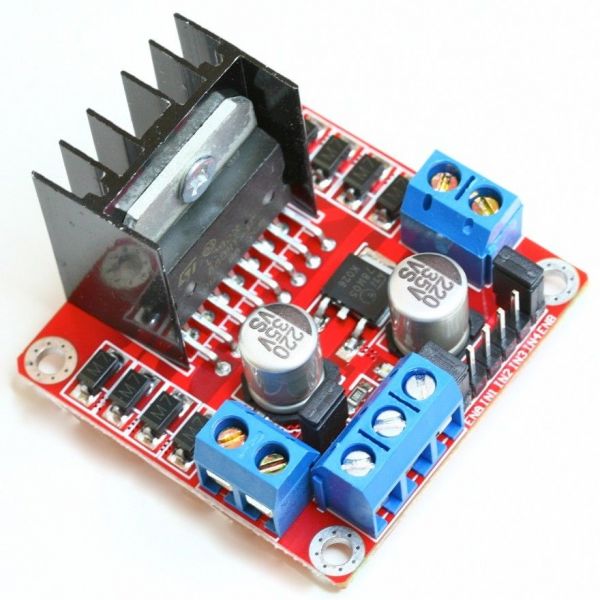
The tic-tac-toe robot is powered by a 12V- 2A dc power adapter. The 5V necessary to operate the microcontroller is obtained by regulating the 12V using a buck converter. The motor driver L298D is supplied with 5V from the buck converter to enable it. The motors are supplied with 12V directly from the adapter through microcontroller. The servo motor is also powered by 5V from the buck converter. Shown below is a schematic of the power supply for the entire project.



Power supply sketch

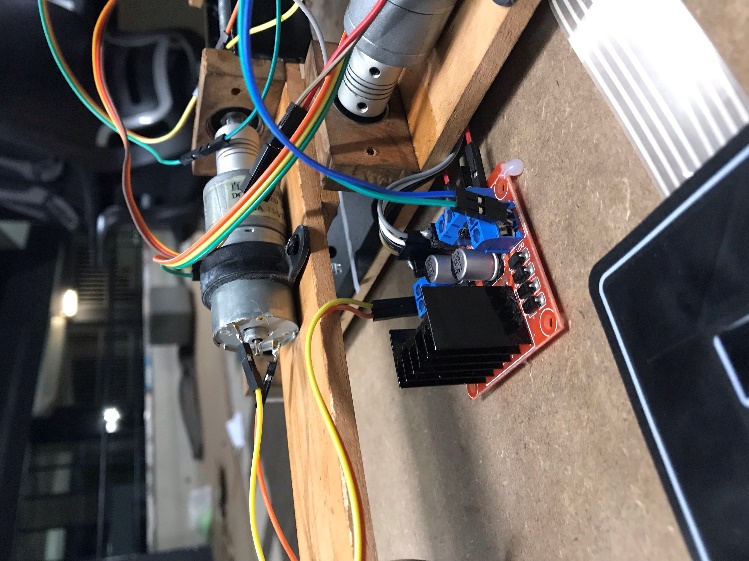
**4.5 Motor controller**

To control the two motors in the linear actuators, we used a L298D H bridge module.



L298D H bridge module

The motors require a 12V DC Voltage, and a 120-mA current at its maximum efficiency. The signal voltages coming out of the microcontroller is 5V, and it can’t provide such a high current for the operation of the motors. Therefore, we use the L298D motor driver. We can input the control signals, and the separate 12V- 2A power supply to operate the motors through the motor driver.



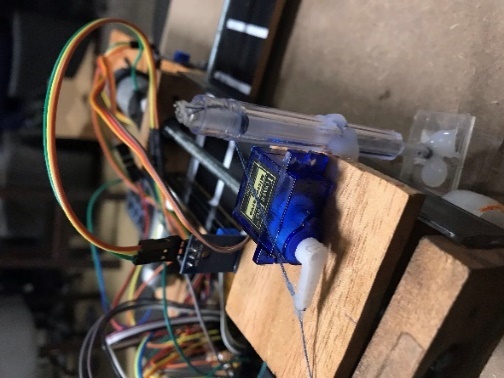
Motor driver

**4.6 Servo control**

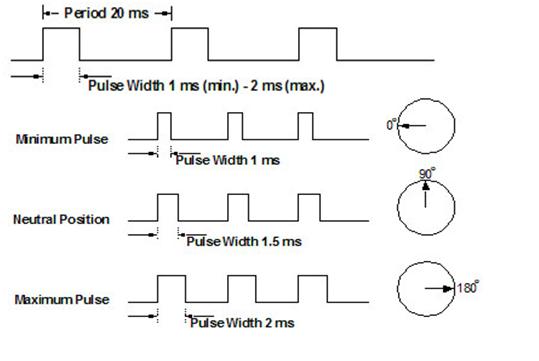
We used a 9g micro servo to operate the mechanism to draw the computer’s move on the paper. A stamp with the mark "[**×**](https://en.wikipedia.org/wiki/Multiplication_sign)" is attached to the servo motor, which is attached to the platform that moves in the x-y plane. Once the platform reaches to the desired position, the servo arm is rotated back and forth such that the stamp is placed on the paper.

Servo motors has the ability to turn by a specific angle, according to the signal given by the microcontroller.

Servos are controlled by sending an electrical pulse of variable width, or pulse width modulation (PWM), through the control wire. There is a minimum pulse, a maximum pulse, and a repetition rate. A servo motor can usually only turn 90° in either direction for a total of 180° movement. The motor's neutral position is defined as the position where the servo has the same amount of potential rotation in the both the clockwise or counter-clockwise direction. The PWM sent to the motor determines position of the shaft, and based on the duration of the pulse sent via the control wire; the rotor will turn to the desired position. The servo motor expects to see a pulse every 20 milliseconds (ms) and the length of the pulse will determine how far the motor turns. For example, a 1.5ms pulse will make the motor turn to the 90° position. Shorter than 1.5ms moves it in the counterclockwise direction toward the 0° position, and any longer than 1.5ms will turn the servo in a clockwise direction toward the 180° position.



9g servo motor



Servo motor input signal

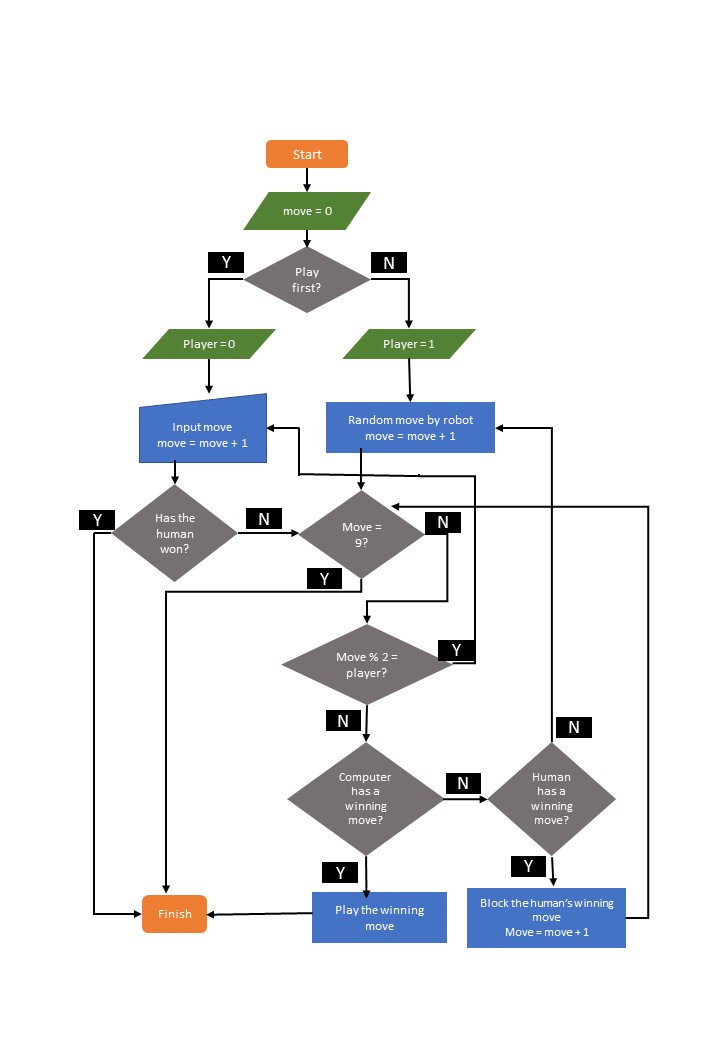
**4.7 Algorithm**

The algorithm used in the tic tac toe robot, is a rather simple, non-recursive algorithm. The program stores the 9 squares of the play area in its memory as an integer array. The blank squares will be saved with the value zero, those played by the human as 1, and those played by the robot as -1.

The played has the freedom to choose to play first or second.

The program, in its memory has all the potential winning positions. The algorithm plays random moves unless there’s a winning chance for the robot, or for the human player. If there’s a chance for the human to win in its next move, the robot blocks it. It there’s a winning move for the robot, it plays it and wins.

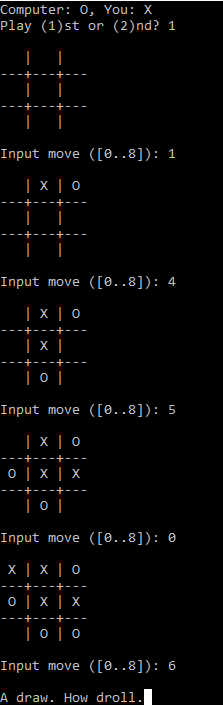
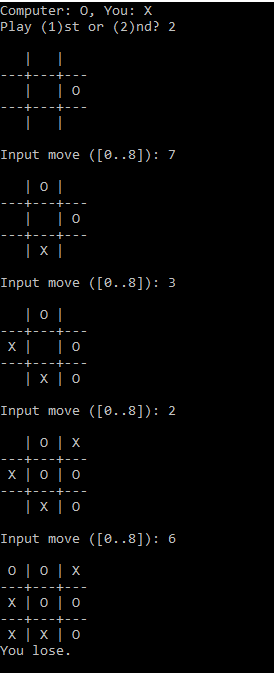
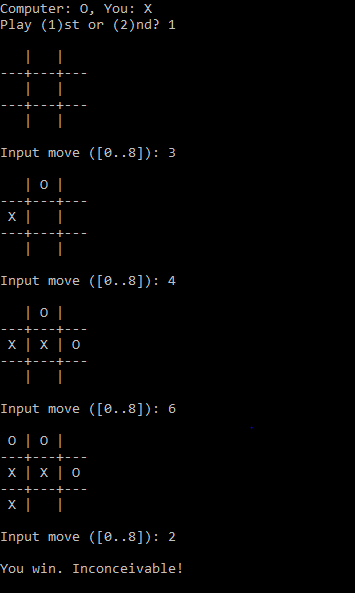
It at the end of 9 moves, nobody has secured to acquire three adjacent squares, the game ends in a draw.



Algorithm flow diagram

**5. RESULTS**

In order to test the algorithm, we executed the code first in Code blocks IDE. the game was simulated and the chances of winning, drawing and losing was calculated. As the capabilities of the pic microcontroller restricted the execution of an unbeatable algorithm, the algorithm that we used had a high tendency of ending the game in a draw. In a lesser percentage of time, the machine or the human player will win.



Draw Lose Win

**6. DISCUSSION**

**Recursive function issue**

One of the most widely used algorithms used to play the game tic tac toe is the minimax algorithm. It is implemented by recursively checking the possibility of winning in the next move. The minimax algorithm, if executed correctly can make the tic tac toe robot unbeatable. But we encountered a problem with executing that algorithm in the pic microcontroller. PIC 16F877A has an 8-level deep call stack, but the minimax algorithm runs recursively to check the possibilities of thousands of game combinations. This cause a stack overflow in the microcontroller, and a misbehavior of the program. On the other hand, the MPLAB with Xc8 compiler doesn’t allow to build the program with recursive functions. So, we had to use a comparatively less intelligent and non-recursive algorithm for our robot, which can be beaten if played with good tactics.

**Linear actuator fabrication issue**

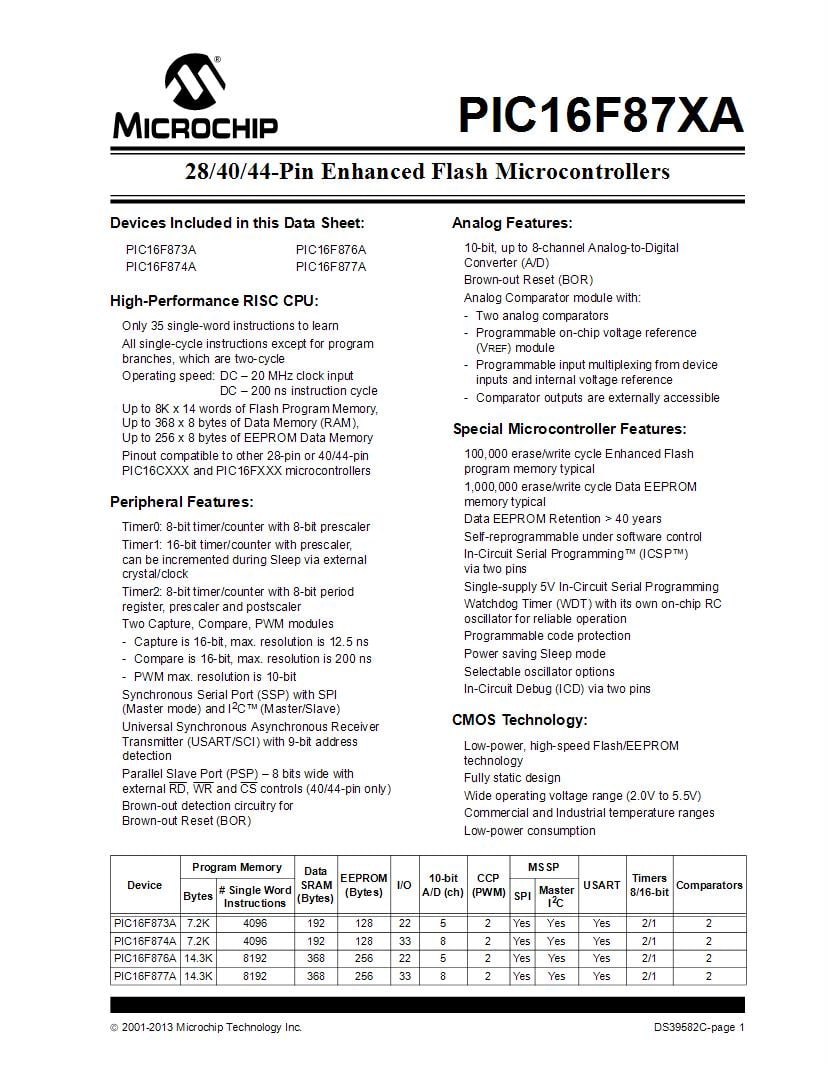
In order to make the drawing mechanism of the robot, we had to make 2 linear actuators, which is placed one on the top of the other, so that the writing tool can move in the x-y plane. We made it by converting the rotational motion of a DC motor to linear motion using 2 nuts and a thread bar going through the nuts. To make this part, we had to find motors, thread bars and nuts, bearings and couplers with matching dimensions, as there were no such complete setups to be purchased. There were many practical problems that arose in the process such as unavailability of components in the shops, unavailability of tools and machines required to do the necessary alterations. As a result, we spend a considerable amount of time to fabricate these actuators. Finally, we were able to come up with a satisfactory setup, but the threads of the thread bar were too close and the rpm of the motors were not enough. Therefore, the linear motion is slower than we had planned and our tic tac toe machine takes more time than expected to complete one game.

**7. REFERENCES**

1. *Homework Writing Machine*. [online] Instructables.com. Available at: <https://www.instructables.com/id/Homework-Writing-Machine/>
2. Becoming Human: Artificial Intelligence Magazine. (2019). *Reinforcement Learning: Train a bot to play tic-tac-toe.*. [online] Available at: <https://becominghuman.ai/reinforcement-learning-step-by-step-17cde7dbc56c>.
3. Google Books. (2019). *An Introduction to PIC Microcontrollers*. [online] Available at: <https://books.google.lk/books/about/An_Introduction_to_PIC_Microcontrollers.html?id=DrJwAAAACAAJ&source=kp_book_description&redir_esc=y>

**8. APPENDICES**

1. Summary datasheet of PIC16F877



1. Code for the tic tac toe machine- MPLAB

/\*

pin configurations

\* keypad

\* rows IN

\* r1= b7

\* r2= b6

\* r3= b5

\* r4= b4

\* columns OUT

\* c1= b2

\* c2 = b1

\* c3= b0

\* motor driver OUT

\* xmf = d0

\* xmb = d1

\* ymf = d2

\* ymb = d3

\* limit switches IN

\* x0 = c0

\* x1 = c1

\* x2 = c2

\* x3 = c3

\* y0 = c4

\* y1 = c5

\* y2 = c6

\* y3 = c7

\* indicator LEDs

D4 , D5, D6, D7

\* ir sensors IN

\* x axis = c1

\* y axis = c5

\* servo OUT

\* signal E1

\*/

#define \_XTAL\_FREQ 4000000

#define TMR2PRESCALE 4

#include <xc.h>

#pragma config FOSC = HS // Oscillator Selection bits (HS oscillator)

#pragma config WDTE = OFF // Watchdog Timer Enable bit (WDT enabled)

#pragma config PWRTE = OFF // Power-up Timer Enable bit (PWRT disabled)

#pragma config BOREN = ON // Brown-out Reset Enable bit (BOR enabled)

#pragma config LVP = OFF // Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit (RB3 is digital I/O, HV on MCLR must be used for programming)

#pragma config CPD = OFF // Data EEPROM Memory Code Protection bit (Data EEPROM code protection off)

#pragma config WRT = OFF // Flash Program Memory Write Enable bits (Write protection off; all program memory may be written to by EECON control)

#pragma config CP = OFF // Flash Program Memory Code Protection bit (Code protection off)

int winchecks[24][2] = {{0,1},{0,2},{1,2},{3,4},{3,5},{4,5},{6,7},{7,8},{6,8},{0,3},{3,6},{0,6},{1,4},{1,7},{4,7},{2,5},{5,8},{2,8},{0,4},{0,8},{4,8},{2,4},{4,6},{2,6}};

int wins[8][3] = {{0,1,2},{3,4,5},{6,7,8},{0,3,6},{1,4,7},{2,5,8},{0,4,8},{2,4,6}};

int board[9] = {0,0,0,0,0,0,0,0,0};

int a=0;

int draw = -1;

int player =0;

int xgo=0;

int ygo=0;

void servoRotate0(){

unsigned int i;

for(i=0;i<50;i++)

{

RE1 = 1;

\_\_delay\_us(800);

RE1 = 0;

\_\_delay\_us(19200);

}

}

void servoRotate90(){

unsigned int i;

for(i=0;i<50;i++)

{

RE1 = 1;

\_\_delay\_us(1500);

RE1 = 0;

\_\_delay\_us(18500);

}

}

void servoRotate180(){

unsigned int i;

for(i=0;i<50;i++)

{

RE1 = 1;

\_\_delay\_us(2200);

RE1 = 0;

\_\_delay\_us(17800);

}

}

void stencil(){

servoRotate90(); //90 Degree

\_\_delay\_ms(500);

servoRotate0(); //0 Degree

\_\_delay\_ms(1000);

servoRotate90(); //90 Degree

\_\_delay\_ms(500);

}

void comeback2(){ //rotates the motors such that the writing tool returns to the original position

int xback=0;

int yback=0;

PORTDbits.RD5 = 1;

\_\_delay\_ms(1000);

PORTDbits.RD5 = 0;

PORTDbits.RD1 = 1;

PORTDbits.RD0 = 0;

\_\_delay\_ms(4000);

while(1){

PORTDbits.RD1 = 1;

PORTDbits.RD0 = 0;

//xgoforward();

if (RC1 ==0){

xback++;

if (xback== xgo){

PORTDbits.RD0 = 0;

PORTDbits.RD1 = 0;

// brake();

break;

}

else{

PORTDbits.RD0 = 0;

PORTDbits.RD1 = 1;//xgoforward();

\_\_delay\_ms(5000);

}

}

else{

PORTDbits.RD0 = 0;

PORTDbits.RD1 = 1;//xgoforward();

}

}

PORTDbits.RD2 = 0;

PORTDbits.RD3 = 1;

\_\_delay\_ms(4000);

while(1){

PORTDbits.RD2 = 0;

PORTDbits.RD3 = 1;//ygoforward();

if (RC5 ==0){

yback++;

if (yback== ygo){

PORTDbits.RD2 = 0;

PORTDbits.RD3 = 0;

//brake();

break;

}

else{

PORTDbits.RD2 = 0;

PORTDbits.RD3 = 1;//ygoforward();

\_\_delay\_ms(5000);

}

}

else{

PORTDbits.RD2 = 0;

PORTDbits.RD3 = 1;//ygoforward();

}

}

PORTDbits.RD5 = 1;

\_\_delay\_ms(500);

PORTDbits.RD5 = 0;

\_\_delay\_ms(500);

PORTDbits.RD5 = 1;

\_\_delay\_ms(500);

PORTDbits.RD5 = 0;

\_\_delay\_ms(500);

stencil();

}

void newdrawy(){ // writing moves to the particular position played by the computer

xgo = 3-(draw%3);

ygo = 3- (draw/3);

int xstate=0;

int ystate=0;

PORTDbits.RD5 = 1;

\_\_delay\_ms(1000);

PORTDbits.RD5 = 0;

PORTDbits.RD0 = 1;

PORTDbits.RD1 = 0;

\_\_delay\_ms(4000);

while(1){

PORTDbits.RD0 = 1;

PORTDbits.RD1 = 0;

//xgoforward();

if (RC1 ==0){

xstate++;

if (xstate== xgo){

PORTDbits.RD0 = 0;

PORTDbits.RD1 = 0;

// brake();

break;

}

else{

PORTDbits.RD0 = 1;

PORTDbits.RD1 = 0;//xgoforward();

\_\_delay\_ms(5000);

}

}

else{

PORTDbits.RD0 = 1;

PORTDbits.RD1 = 0;//xgoforward();

}

}

PORTDbits.RD2 = 1;

PORTDbits.RD3 = 0;

\_\_delay\_ms(4000);

while(1){

PORTDbits.RD2 = 1;

PORTDbits.RD3 = 0;//ygoforward();

if (RC5 ==0){

ystate++;

if (ystate== ygo){

PORTDbits.RD2 = 0;

PORTDbits.RD3 = 0;

//brake();

break;

}

else{

PORTDbits.RD2 = 1;

PORTDbits.RD3 = 0;//ygoforward();

\_\_delay\_ms(5000);

}

}

else{

PORTDbits.RD2 = 1;

PORTDbits.RD3 = 0;//ygoforward();

}

}

PORTDbits.RD5 = 1;

\_\_delay\_ms(500);

PORTDbits.RD5 = 0;

\_\_delay\_ms(500);

PORTDbits.RD5 = 1;

\_\_delay\_ms(500);

PORTDbits.RD5 = 0;

stencil();

comeback2();

}

void xgoforward(){ // rotates the motor in the x axis forward

/\* xmf = d0

\* xmb = d1\*/

PORTDbits.RD0 = 1;

PORTDbits.RD1 = 0;

}

void ygoforward(){ //rotates the motor in the y axis forward

/\*

\* ymf = d2

\* ymb = d3\*/

PORTDbits.RD2 = 1;

PORTDbits.RD3 = 0;

}

void brake(){ //stopes the motors

PORTDbits.RD0 = 0;

PORTDbits.RD1 = 0;

PORTDbits.RD2 = 0;

PORTDbits.RD3 = 0;

}

void getkey(){ //gets the input from the keypad, played by the human player

a= 0;

while(1){

PORTBbits.RB0 = 0;

PORTBbits.RB1 = 0;

PORTBbits.RB2 = 1;

if (RB7==1){

a = 7;

break;

}

if (RB6==1){

a = 8;

break;

}

if (RB5==1){

a = 9;

break;

}

if (RB4==1){

a = 10;

break;

}

PORTBbits.RB0 = 0;

PORTBbits.RB1 = 1;

PORTBbits.RB2 = 0;

if (RB7==1){

a = 4;

break;

}

if (RB6==1){

a = 5;

break;

}

if (RB5==1){

a = 6;

break;

}

if (RB4==1){

a = 11;

break;

}

PORTBbits.RB0 = 1;

PORTBbits.RB1 = 0;

PORTBbits.RB2 = 0;

if (RB7==1){

a = 1;

break;

}

if (RB6==1){

a = 2;

break;

}

if (RB5==1){

a = 3;

break;

}

if (RB4==1){

a = 12;

break;

}

}

}

int first(){ //determines if the human player wants to play first or second

while( a!=1 && a!= 2){

getkey();

}

if (a==1){

player =1;

}

if (a==2){

player =2;

}

return player;

}

void init(){ //configures the pins of the PIC as input or output

TRISBbits.TRISB0 = 0;

TRISBbits.TRISB1 = 0;

TRISBbits.TRISB2 = 0;

TRISBbits.TRISB4 = 1;

TRISBbits.TRISB5 = 1;

TRISBbits.TRISB6 = 1;

TRISBbits.TRISB7 = 1;

TRISDbits.TRISD0 = 0;

TRISDbits.TRISD1 = 0;

TRISDbits.TRISD2 = 0;

TRISDbits.TRISD3 = 0;

TRISDbits.TRISD4 = 0;

TRISDbits.TRISD5 = 0;

TRISDbits.TRISD6 = 0;

TRISDbits.TRISD7 = 0;

TRISCbits.TRISC1 = 1;

TRISCbits.TRISC5 = 1;

TRISEbits.TRISE1=0;

//TRISCbits.TRISC7 = 0;

PORTDbits.RD0 = 0;

PORTDbits.RD1 = 0;

PORTDbits.RD2 = 0;

PORTDbits.RD3 = 0;

PORTBbits.RB1 = 0;

PORTBbits.RB2 = 0;

PORTBbits.RB3 = 0;

servoRotate90(); //90 Degree

\_\_delay\_ms(500);

}

int win(const int board[9]) {

//determines if a player has won, returns 0 otherwise.

//unsigned wins[8][3] = {{0,1,2},{3,4,5},{6,7,8},{0,3,6},{1,4,7},{2,5,8},{0,4,8},{2,4,6}};

int i;

for(i = 0; i < 8; ++i) {

if(board[wins[i][0]] != 0 &&

board[wins[i][0]] == board[wins[i][1]] &&

board[wins[i][0]] == board[wins[i][2]])

return board[wins[i][2]];

}

return 0;

}

void randomMove(int board[]){ //generates and plays a random move

int count=0;

int i=0;

for(i = 0; i < 9; ++i) {

if( board[i]==0){

count++;

}

}

int r = rand()%count;

//printf("%d",randy()%count);

int h=-1;

for(i = 0; i < 9; ++i) {

if( board[i]==0){

h++;

}

if (h== r){

//printf("ok");

board[i]=1;

draw = i;

newdrawy();

return;

}

}

}

int wincheck(int board[]){ // checks if the human player or the computer has a chance to win in the next move

int q=0;

while(q<24){

//int a= winchecks[q][0];

if (board[winchecks[q][0]]==board[winchecks[q][1]] && board[winchecks[q][0]]==1 && (board[wins[q/3][0]]!=-1 && board[wins[q/3][1]]!=-1 && board[wins[q/3][2]]!=-1 )){

return q;//ans[2];

}

q++;

}

int b=0;

while(b<24){

if (board[winchecks[b][0]]==board[winchecks[b][1]] && board[winchecks[b][0]]==-1 && (board[wins[b/3][0]]!=1 && board[wins[b/3][1]]!=1 && board[wins[b/3][2]]!=1 )){

return b;

}

b++;

}

return -1;

}

void computerMove(int board[]) { //move done by the robot

PORTDbits.RD7 = 1;

\_\_delay\_ms(1000);

PORTDbits.RD7 = 0;

int p =wincheck(board);

if (p ==-1){

randomMove(board);

}

else

{ p /=3;

int k =0;

while(k<3){

if(board[wins[p][k]]==0){

board[wins[p][k]]= 1;

draw = wins[p][k];

newdrawy();

return;

}

k++;

}

}

}

void playerMove(int board[9]) { //move of the human player

PORTDbits.RD6 = 1;

\_\_delay\_ms(1000);

PORTDbits.RD6 = 0;

int move =0;

a = 0;

do {

while (a==0){

getkey();

}

move = a-1;

} while (move >= 9 || move < 0 && board[move] == 0);

board[move] = -1;

PORTDbits.RD6 = 1;

\_\_delay\_ms(1000);

PORTDbits.RD6 = 0;

}

int main() { //calls the previous functions in the required order to run the tic tac toe machine

init();

stencil();

player=0;

player = first();

if(player!=0){

PORTDbits.RD4 = 1;

\_\_delay\_ms(1000);

PORTDbits.RD4 = 0;

}

int turn;

for(turn = 0; turn < 9 && win(board) == 0; ++turn) {

if((turn+player) % 2 == 0)

computerMove(board);

else {

playerMove(board);

}

}

while(1){

PORTDbits.RD6 = 1;

\_\_delay\_ms(500);

PORTDbits.RD6 = 0;

\_\_delay\_ms(500);

}

}