PIC32-Commanded Remote Control Car Loaded with Inductive Sanitizer Foam Dispenser

Shiyuan Shan, Yuankai Xu, Zhan Xu *

August 4, 2022

1 Objectives

The objectives of this project are listed below.

- Complete the overall structure of the car.
- Implement the motivation control of the car through a controller. Omnidirectional wheels will be used to assure an agile and correct motion.
- Implement the function of inductive sanitizer dispenser. A distance sensor will be connected to the pneumatic device to control the spray of hand sanitizer.

2 Introduction

2.1 Overview

To prevent COVID-19, students on campus should take a lot of precautionary measures. One of the most common steps is to use hand sanitizer foam to wash hands. However, under some circumstances like round-table conference, it is not that convenient for all people to attach the hand sanitizer dispenser. Therefore, in this project, our group plan to design a remote control car which loads an inductive sanitizer dispenser, that guarantees everyone within an area to clean their hands.
In this project, one PIC32 board and one vex brain are used. PIC32 takes all

*This course is held by UM-SJTU Joint Institute, Shanghai Jiao Tong University. The mentor of this course is Dr. An Zou.

the computing work, while vex brain does the communication work between PIC32 and some peripherals including distance sensor and the controller.

2.2 Top-level Block Diagram

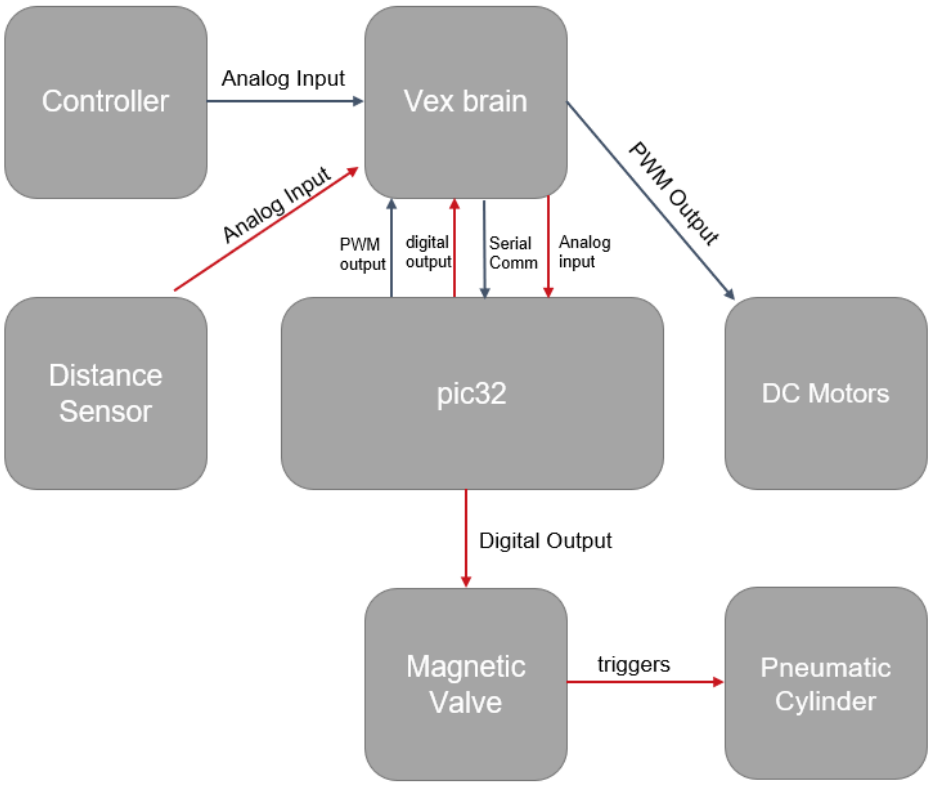

Figure 1: Top-level Block Diagram

3 High-level description

We will use PIC32 board in this project as the core logical computation part and peripherals include

- A controller connected to the vex brain.
- A distance sensor.
- DC motors that can drive the wheels of the car.

- A magnetic valve and matched pneumatic cylinder.

For car motivation control, when the user provides input via the controller, the vex brain sends the signal to PIC32 through UART. PIC32 will generate corresponding PWM output signals to control the DC motors. The movement of the car can be realized through three signals, the front and back movement, left and right movement, and clockwise and counter-clockwise rotation.

For sanitizer dispenser control, the vex brain will receive the analog signal from the the distance sensor, then transmit the analog signal directly to the PIC32 board. Through A/D converter, the PIC32 will judge whether object is within designed distance and send digital signal to control the magnetic valve, which will drive the pneumatic cylinder and finally generate an impulse to spray the sanitizer out.

4 Structure model

This part demonstrates the overall structure of our project, which is a Solidworks engineering drawing.

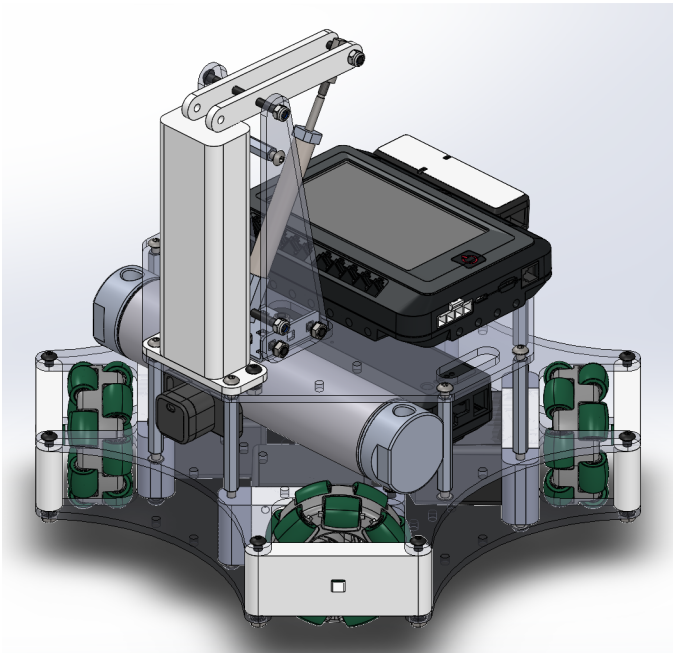

Figure 2: Structure model of the remote control car

5 Real product

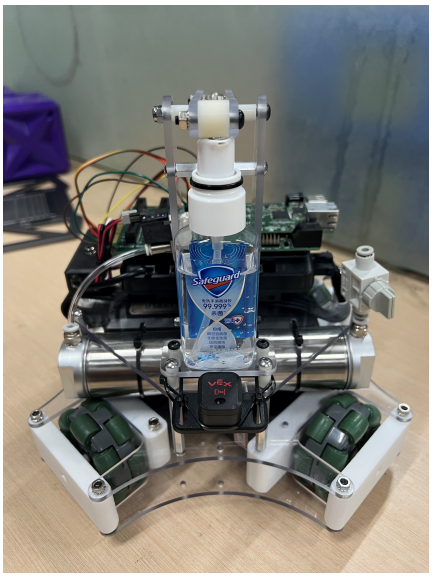

Figure 3: Front view

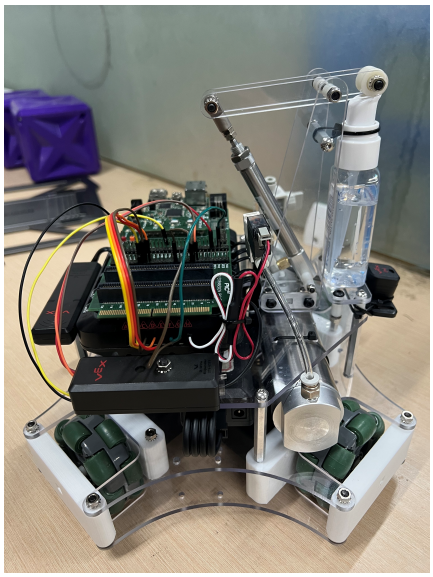

Figure 4: Side view

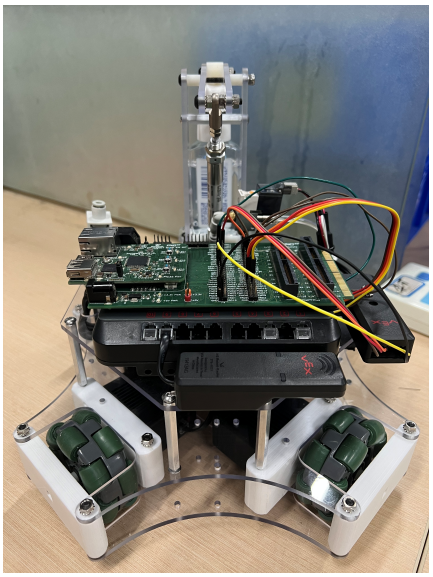

Figure 5: Back view

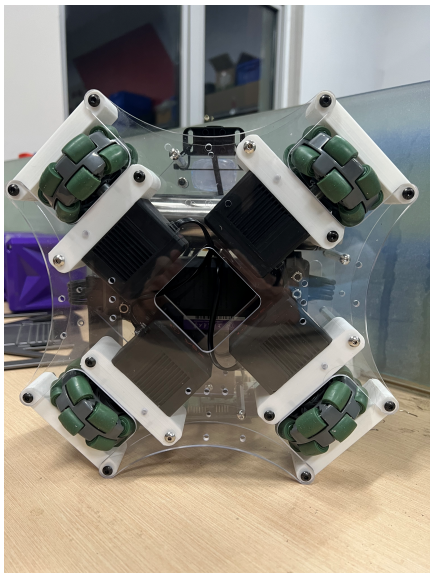

Figure 6: Bottom view

6 Function Test and Evaluation

We tested the performance of our project in two aspects: motivation control, and sanitizer dispenser module.

For movement control, we first tested the signals of the three channels: forward and backward movement, left and right movement, and clockwise and counter-clockwise rotation. The car perfectly execute the instructions. Then we try combined input, i.e., move to the left front, the car also move smoothly following the instructions.

For sanitizer dispenser module, we test the following situations:

1. put a hand 100mm upon the distance sensor for 1s.
2. put a hand 30mm upon the distance sensor for 1s.
3. rapidly swing a hand 30mm upon the distance sensor, the time it is upon the distance sensor will be less than 0.2s.

and we got the corresponding results:

1. the inductive sanitizer form dispenser doesn't work.
2. the dispenser works and dispenses some sanitizer.
3. the inductive sanitizer form dispenser doesn't work.

Both parts of the project function normally. The objectives of the project have been well achieved: a) to design the overall structure of a remote control car loaded with an inductive sanitizer dispenser, b) to realize the movement control of the car, and c) to implement the function of inductive sanitizer foam dispenser. Therefore, this project is considered a success.

A Source code for PIC32

```
//#include <p32xxxx.h>
#include <stdio.h>
#include <math.h>
#include <proc/p32mx795f512l.h>
#include "LCD.h"
//#include "LCD.c"

int input_ = 0;
int8_t uart_buffer[4];
int ch1 = 0, ch3 = 0, ch4 = 0;
//int input_test = 10;

void delay() {
    unsigned int i = 0;
    while(i < 10000) {
        i++;
    }
}

void run() {
    //LF, RF, LB, RB ? [-100, 100]
    int LF = (ch3 + ch4 + ch1) / 3;
    int RF = (-ch3 + ch4 + ch1) / 3;
    int LB = (ch3 - ch4 + ch1) / 3;
    int RB = (-ch3 - ch4 + ch1) / 3;

    LF = (LF + 100) / 2;
    RF = (RF + 100) / 2;
    LB = (LB + 100) / 2;
    RB = (RB + 100) / 2;

//    OC1RS = LF / 100 * PR2;
    OC5RS = LF / 100 * PR2;
    OC2RS = RF / 100 * PR2;
    OC3RS = LB / 100 * PR2;
    OC4RS = RB / 100 * PR2;
    delay();
}
```

```
#pragma interrupt UART_ISR ipl6 vector 24
void UART_ISR(){
    IEC0bits.U1RXIE = 0;
    uart_buffer[0] = U1RXREG;
    ch1 = uart_buffer[0];
    run();
    IFS0bits.U1RXIF = 0;
    IEC0bits.U1RXIE = 1;
}

#pragma interrupt UART_ISR_2 ipl6 vector 32
void UART_ISR_2(){
    IEC1bits.U2RXIE = 0;
    uart_buffer[1] = U2RXREG;
    ch3 = uart_buffer[1];
    run();
    IFS1bits.U2RXIF = 0;
    IEC1bits.U2RXIE = 1;
}

#pragma interrupt UART_ISR_3 ipl6 vector 51
void UART_ISR_3(){
    IEC2bits.U5RXIE = 0;
    uart_buffer[2] = U5RXREG;
    ch4 = uart_buffer[2];
    run();
    IFS2bits.U5RXIF = 0;
    IEC2bits.U5TXIE = 1;
}

//
void UART_init() {
    U1BRG = 51;                     //9600 baud rate, BRGH = 0
    U1MODE = 0;
    U1MODEbits.STSEL = 0;           //1 stop bit
    U1MODEbits.PDSEL = 0b00;        //8-bit data, no parity
    U1MODEbits.BRGH = 0;            //16x baud clock enabled
    U1MODEbits.UEN = 0b00;          //UxTX and UxRX pins are enabled and used
```

```
79      U1STAbits.URXISEL = 0b00;       //interrupt when buffer is not empty
80      IEC0bits.U1RXIE = 1;            //set receive interrupt
81      IFS0bits.U1RXIF = 0;
82      IPC6bits.U1IP = 0b110;          //Priority level 6
83      IPC6bits.U1IS = 0b11;           //Sub-priority level 3
84      U1STAbits.URXEN = 1;            //UARTx receiver enabled
85  //    U1STAbits.UTXEN = 1;            //UARTx transmitter enabled
86      U1MODEbits.ON = 1;              //UARTx is enabled
87
88      U2BRG = 51;                     //9600 baud rate, BRGH = 0
89      U2MODE = 0;
90      U2MODEbits.STSEL = 0;           //1 stop bit
91      U2MODEbits.PDSEL = 0b00;        //8-bit data, no parity
92      U2MODEbits.BRGH = 0;            //16x baud clock enabled
93      U2MODEbits.UEN = 0b00;          //UxTX and UxRX pins are enabled and used
94      U2STAbits.URXISEL = 0b00;       //interrupt when buffer is not empty
95      IEC1bits.U2RXIE = 1;            //set receive interrupt
96      IFS1bits.U2RXIF = 0;
97      IPC8bits.U2IP = 0b110;          //Priority level 6
98      IPC8bits.U2IS = 0b11;           //Sub-priority level 3
99      U2STAbits.URXEN = 1;            //UARTx receiver enabled
100 //    U2STAbits.UTXEN = 1;            //UARTx transmitter enabled
101     U2MODEbits.ON = 1;              //UARTx is enabled
102
103     U5BRG = 51;                     //9600 baud rate, BRGH = 0
104     U5MODE = 0;
105     U5MODEbits.STSEL = 0;           //1 stop bit
106     U5MODEbits.PDSEL = 0b00;        //8-bit data, no parity
107     U5MODEbits.BRGH = 0;            //16x baud clock enabled
108 //    U5MODEbits = 0b00;            //UxTX and UxRX pins are enabled and used
109     U5STAbits.URXISEL = 0b00;       //interrupt when buffer is not empty
110     IEC2bits.U5RXIE = 1;            //set receive interrupt
111     IFS2bits.U5RXIF = 0;
112     IPC12bits.U5IP = 0b110;         //Priority level 6
113     IPC12bits.U5IS = 0b11;          //Sub-priority level 3
114     U5STAbits.URXEN = 1;            //UARTx receiver enabled
115 //    U5STAbits.UTXEN = 1;            //UARTx transmitter enabled
116     U5MODEbits.ON = 1;              //UARTx is enabled
```

```
120 void ADC_init(){
121     TRISB = 0xFFFF;                         //set PORTB as input
122
123     AD1CON1bits.SSRC = 0b010;               //Timer3 period match ends sampling a
124     AD1CON1bits.SIDL = 0b1;                 //discontinue module operation in id
125     AD1CON1bits.ASAM = 0b1;                 //auto sampling
126     AD1CON2 = 0x0000;
127     AD1CON2bits.SMPI = 0b0000;              //Interrupt after each sample
128     AD1CON3bits.ADCS = 0b1111111;           //T_AD = 512 * T_PB
129     AD1CON3bits.SAMC = 0b00100;             //Auto-sample time bits = 4 * T_AD
130     AD1CON3bits.ADRC = 0b0;                 //PBCLK
131     AD1CHSbits.CH0SA = 0b0001;              //Channel 0 positive input is AN1
132     AD1PCFGbits.PCFG0 = 0;
133     AD1CSSL = 0;                            //no scan
134     T3CON = 0x0;
135     TMR3 = 0x0;
136     PR3 = 7999;
137     T3CONbits.TCKPS = 0b011;                //1:8
138     IFS1bits.AD1IF = 0;                     //Clear interrupt
139     IEC1SET = 0x0002;                       //Enable ADC interrupt
140     T3CONbits.ON = 1;
141     AD1CON1bits.ADON = 1;                   //turn on the ADC
142 }
143
144 int judge_dist(int dist){
145     //dist: 0 ~ 3300mm
146     //return 1 if dist < 50mm
147     if (dist < 1024 / 66){
148         return 1;
149     }
150     else
151         return 0;
152 }
153
154 void mag_valve(){
155     LATGbits.LATG7 = 1;
156     delay();
157     LATGbits.LATG7 = 0;
```

```
154  void mag_valve(){
155      LATGbits.LATG7 = 1;
156      delay();
157      LATGbits.LATG7 = 0;
158      delay();
159  }
160
161  void MCU_init() {
162      asm("di");
163      TRISECLR = 0x00FF;          //PORTE output
164      TRISGbits.TRISG7 = 0;       //RG7 output
165      INTCONbits.MVEC = 1;        // Enable multiple vector interrupt
166      TRISDbits.TRISD1 = 0;       // OC2 as output
167      TRISDbits.TRISD2 = 0;       // OC3 as output
168      TRISDbits.TRISD3 = 0;       // OC4 as output
169      TRISDbits.TRISD4 = 0;       // OC5 as output
170      asm("ei");                  // Enable all interrupts
171      TMR2 = 0;
172      PR2 = 999;
173      T2CON = 0x0;
174      T2CONbits.TCKPS = 0b011;    // 1:8
175  //      IPC2SET = 0x00000014;       // Interrupt priority level 5, subpriority l
176      IPC2bits.T2IP = 5;          //Interrupt priority level 5
177      IPC2bits.T2IS = 3;          //subpriority level 3
178      IFS0bits.T2IF = 0;
179      IEC0bits.T2IE = 1;          // Enable Timer2 interrupts
180      T2CONbits.TCS = 0;
181      T2CONbits.TGATE = 0;
182  }
183
184  #pragma interrupt TMR2_ISR ipl5 vector 8
185  void TMR2_ISR() {
186      IEC0bits.T2IE = 0;
187      T2CONbits.ON = 0;
188      TMR2 = 0x0;
189      IFS0bits.T2IF = 0;      // Clear timer2 interrupt flag
190      IEC0bits.T2IE = 1;
191  }
```

```
193 void PWM_init() {
194     OC2CON = 0x0000;
195     OC2CONbits.OCM = 0b110;             //PWM mode, fault pin disabled
196     OC2CONbits.OCTSEL = 0;              //Timer2 is used
197     OC2CONbits.OC32 = 0;                //16-bit
198     OC2RS = 0;
199     OC2R = 0;
200
201     OC3CON = 0x0000;
202     OC3CONbits.OCM = 0b110;             //PWM mode, fault pin disabled
203     OC3CONbits.OCTSEL = 0;              //Timer2 is used
204     OC3CONbits.OC32 = 0;                //16-bit
205     OC3RS = 0;
206     OC3R = 0;
207
208     OC4CON = 0x0000;
209     OC4CONbits.OCM = 0b110;             //PWM mode, fault pin disabled
210     OC4CONbits.OCTSEL = 0;              //Timer2 is used
211     OC4CONbits.OC32 = 0;                //16-bit
212     OC4RS = 0;
213     OC4R = 0;
214
215     OC5CON = 0x0000;
216     OC5CONbits.OCM = 0b110;             //PWM mode, fault pin disabled
217     OC5CONbits.OCTSEL = 0;              //Timer2 is used
218     OC5CONbits.OC32 = 0;                //16-bit
219     OC5RS = 0;
220     OC5R = 0;
221
222 //    OC1CONbits.ON = 1;
223     OC2CONbits.ON = 1;
224     OC3CONbits.ON = 1;
225     OC4CONbits.ON = 1;
226     OC5CONbits.ON = 1;
227 }
228
229 int main() {
230     SYSKEY = 0x0;                       // ensure OSCCON is locked
```

```
228
229 int main() {
230     SYSKEY = 0x0;                       // ensure OSCCON is locked
231     SYSKEY = 0xAA996655;                // write Key1 to SYSKEY
232     SYSKEY = 0x556699AA;                // write Key2 to SYSKEY
233     OSCCONbits.PBDIV = 0b00;            // configure OSCCON.PBDIV to 1:1
234     SYSKEY = 0x0;                       // Lock OSCCON
235
236     MCU_init();;
237     UART_init();
238     PWM_init();
239     ADC_init();
240
241     LATGbits.LATG7 = 0;
242     int timer = 0;
243     int launchReady = 0;
244     int objDetectedLast = 1;
245     while(1){
246         while(!IFS1 & 0x0002) {}
247         delay();
248         input_ = ADC1BUF0;
249         int objDetected = judge_dist(input_);
250         if (objDetected ) {
251             if (!objDetectedLast) {
252                 timer = 0;
253                 launchReady = 1;
254             } else if (timer > 3 & launchReady){
255                 mag_valve();
256                 launchReady = 0;
257             }
258         }
259         timer++;
260         objDetectedLast = objDetected;
261         IFS1CLR = 0x0002;
262     }
263 }
264
```

```c
#include <p32xxxx.h>

// define macros for LCD instructions
#define LCD_IDLE                0x33
#define LCD_2_LINE_4_BITS       0x28
#define LCD_2_LINE_8_BITS       0x38
#define LCD_DSP_CSR             0x0c
#define LCD_CLR_DSP             0x01
#define LCD_CSR_INC             0x06
#define LCD_SFT_MOV             0x14

// define macros for interfacing ports
#define RS                      PORTDbits.RD1
#define E                       PORTFbits.RF1
#define Data                    PORTE

typedef unsigned char uchar;
//int input_DistSensor = 0;               //analog input
//int LF = 0, RF = 0, LB = 0, RB = 0;     //output for motors
//int Ch3 = 0, Ch4 = 0, Ch1 = 0;          //controller input

//struct bits {
//    unsigned timer2_done : 1;
//} flags;

/* define constant strings for display */
//const uchar startStr1[] = "Digital Clock";

/* Function prototypes */
void MCU_init(void);
void LCD_init(void);
void LCD_putchar(uchar c);
void LCD_puts(const uchar *s);
void LCD_goto(uchar addr);
void GenMsec(void);
void DelayUsec(uchar num);
void DelayMsec(uchar num);

```

```
1   //#include <p32xxxx.h>
2   #include <stdio.h>
3   #include <math.h>
4   #include <proc/p32mx795f512l.h>
5   #include "LCD.h"
6
7   /* initialize the LCD module */
8   void LCD_init() {
9       /* send LCD configuration commands */
10      DelayMsec(15);              //wait for 15 ms
11      RS = 0;                     //send command
12      Data = LCD_IDLE;            //function set - 8 bit interface
13      DelayMsec(5);               //wait for 5 ms
14      Data = LCD_IDLE;            //function set - 8 bit interface
15      DelayUsec(100);             //wait for 100 us
16      Data = LCD_IDLE;            //function set
17      DelayMsec(5);
18      Data = LCD_IDLE;
19      DelayUsec(100);
20      LCD_putchar(LCD_2_LINE_4_BITS);
21      DelayUsec(40);
22      LCD_putchar(LCD_DSP_CSR);       //display on
23      DelayUsec(40);
24      LCD_putchar(LCD_CLR_DSP);       //clear display
25      DelayMsec(5);
26      LCD_putchar(LCD_CSR_INC);       //increment cursor, no display shift
27  }
28
29  /* Send one byte c (instruction or data) to the LCD */
30  void LCD_putchar(uchar c) {
31      E = 1;
32      Data = c;               //sending higher nibble
33      E = 0;                  //producing falling edge on E
34      E = 1;
35      Data <<= 4;             //sending lower nibble through higher 4 ports
36      E = 0;                  //producing falling edge on E
37  }
38
39  /* Display a string of characters *s by continuously calling LCD_putchar() */
40  void LCD_puts(const uchar *s) {
```

```
28
29    /* Send one byte c (instruction or data) to the LCD */
30    void LCD_putchar(uchar c) {
31        E = 1;
32        Data = c;               //sending higher nibble
33        E = 0;                  //producing falling edge on E
34        E = 1;
35        Data <<= 4;             //sending lower nibble through higher 4 ports
36        E = 0;                  //producing falling edge on E
37    }
38
39    /* Display a string of characters *s by continuously calling LCD_putchar() */
40    void LCD_puts(const uchar *s) {
41        uchar pos = *s;
42        RS = 1;                 //write data
43        DelayUsec(40);
44    //    delay();
45        while(pos) {
46            LCD_putchar(pos);
47            pos = *++s;
48            DelayMsec(40);
49    //        delay();
50        }
51        RS = 0;
52    }
53
54    /* go to a specific DDRAM address addr */
55    void LCD_goto(uchar addr) {
56        uchar t_addr = 0x80 + addr;
57        LCD_putchar(t_addr);
58        DelayMsec(40);
59    }
60
61    /* configure timer SFRs to generate num us delay*/
62    void DelayUsec(uchar num) {
63        TMR2 = 0x0000;          // Clear contents of TMR2
64        PR2 = num;              // generate 1 us delay
65        T2CONbits.ON = 1;
66        while(T2CONbits.ON);
67    }
```

```
44      //    delay();
45          while(pos) {
46              LCD_putchar(pos);
47              pos = *++s;
48              DelayMsec(40);
49      //        delay();
50          }
51          RS = 0;
52      }
53
54      /* go to a specific DDRAM address addr */
55      void LCD_goto(uchar addr) {
56          uchar t_addr = 0x80 + addr;
57          LCD_putchar(t_addr);
58          DelayMsec(40);
59      }
60
61      /* configure timer SFRs to generate num us delay*/
62      void DelayUsec(uchar num) {
63          TMR2 = 0x0000;          // Clear contents of TMR2
64          PR2 = num;              // generate 1 us delay
65          T2CONbits.ON = 1;
66          while(T2CONbits.ON);
67      }
68
69      /* configure timer SFRs to generate 1 ms delay*/
70      void GenMsec() {
71          TMR2 = 0x0000;          // Clear contents of TMR2
72          PR2 = 1000;             // generate 1 ms delay
73          T2CONbits.ON = 1;
74          while(T2CONbits.ON);
75      }
76
77      /* Call GenMsec() num times to generate num ms delay*/
78      void DelayMsec(uchar num) {
79          uchar i;
80          for (i=0; i<num; i++) {
81              GenMsec();
82          }
83      }
```

B Vex brain demo