EyeBot Jr. Tutorial
Sugiono, Neubronner, Bräunl, UWA 2005
http://robotics.ee.uwa.edu.au

Using the software

1. Run the program “PICAXE Programming Editor”.
2. Choose “28X” for mode and the options set as “4 MHz” and “256x gosubs” and click OK.
3. After writing the program, press F4 to check the syntax of your program before loading the program to EyeBot Jr.
4. To load your program to EyeBot Jr., connect the serial cable to the serial download port of EyeBot Jr. (Top left).
   Turn on the EyeBot Jr.
   Press F5 to load your program to EyeBot Jr.

Pin Label	PortC 5	PortC 7	PortC 0
Function	$\overline{CE}$	$\overline{WR}$	LATCH
Motor	1	X	1
Servo	1	X	1
Write Display	0	0	0
Read Display	0	1	0
Keypad	1	X	X
Speaker	1	X	0
Serial Out	1	X	0
Digital Out	1	X	0

NOTES:

Pin0 refers to input pin 0
0 refers to output pin 0
Portc 0 refers to portc pin 0 (I/O port)
Analogue input 0 reads battery
Analogue input 1 reads keypad and check display ready

Value of ADC1	Condition
255	Display Ready
206	KEY1
152	KEY2
102	KEY3
52	KEY4

- Motor control
  To select motor, we must first set the corresponding ports according to the table.

  high portc 5
  low portc 7
  high portc 0

Motor A is controlled from output pins 0 and 1
Motor B is controlled from output pins 2 and 3
Motor A & B speed is controlled from portc 1

Motor C is controlled from output pins 4 and 5
Motor D is controlled from output pins 6 and 7
Motor C & D speed is controlled from portc 2

Example 1 Set Motor A forward and Motor B reverse at 100% speed

```
high portc 5
low portc 7
high portc 0
high portc 1
low 0
high 1
high 2
low 3
```

To set the directions of all four motors in one line, we can use the command let pins.

Example 2 Switch outputs 7,5,3,1 on
let pins = %10101010

To use PWM on the motors, we use pwmout command.
Syntax: PWMOUT pin,period,dutycycles

- pin is 1 for Motor A & B, 2 for Motor C & D
- Period is a variable/constant (0-255) which sets the PWM period.
- Duty is a variable/constant (0-1023) which sets the PWM duty cycle.

NOTE: Duty can't be set more than 4 x period, as the on-time can be more than the period.

Example 3 Set Motor A forward and Motor B reverse at 50% speed

```
high portc 5
low portc 7
high portc 0
pwmout portc 1 255,510
let pins = %00000110
```

- **Servo**
  Syntax: servo pin,pulse

Pin is a variable/constant(0-7) which specifies which output pin to use
Pulse is variable/constant(75-225) which specifies the servo position

Do not use a pulse value less than 75 or greater than 225, as this may cause the servo to malfunction. Due to tolerances in servo manufacture all values are approximate and will require fine-tuning by experimentation.
Servo cannot be used at the same time as pwmout as they share a common timer.

Example 4 Set servo 1 to move left and right with 1 second delay

```
main:  high portc 5
       low portc 7
       high portc 0
       servo 0,75
       pause 1000
       servo 0,225
       pause 1000
       goto main
```

- **Analogue input**

Syntax: readadc channel,variable

Readadc is used to read analogue input values (8 bit resolution) and transfer it to a variable.
Readadc10 is used to read analogue input values (10 bit resolution) and transfer it to a word variable.

Example 5 Run motor A at 100% fwd when KEY1 is pressed

```
main:  readadc 1,b0
       if b0 < 216 and b0 > 196 then KEY1
       goto main

KEY1: high portc 5
       low portc 7
       high portc 0
       high portc 1
       let pins %00000010
```

- **Speaker**

Syntax: sound pin,(note, duration, note, duration…)
Note(s) are variables/constants (0-255) which specify type and frequency. Note 0 is silent for the duration. Notes 1-127 are ascending tones. Notes 128-255 are ascending white noises.
Note values: A(49), As(51), B(54), C(57), Cs(61), D(65), Ds(71), E(78), F(88), Fs(101), G(119).

- **Display**

  To display text easier on EyeBot Jr. several subroutines must be copied at the bottom of your program. These subroutines will automatically select (and deselect) the LCD and set all the ports for reading/writing data into the LCD controller.
  To display text on the LCD, first we need to initialise the LCD to set the initial settings. Completing that, we need to buffer the characters into a variable first, before sending it to the LCD controller. We use the for-loop and a 'lookup' command in BASIC to buffer the characters.

  Example 6 Display Hello World! On LCD

```
gosub init_screen
gosub clr_text
gosub clr_menu
for b0 = 0 to 11
        lookup b0,("Hello World!"),b1
        b1 = b1 - $20
        let pins = b1
        gosub data_write
        pause 1
        let pins = $C0
        gosub cmd_write
        pause 1
next b1
```

  The LCD's character generator is off by 20 hex, so we need to subtract $20 from the buffer.
  LCD command $C0, will display the current character and increase the address pointer. Command $C2 will display the current character and decrease the address pointer. Command $C4 will display current character and remain in the same address pointer.
  Sub-procedure clr_text will clear the text area of the LCD (First 7 lines), while clr_menu will clear the menu area of the LCD (Last line).

APPENDIX

```
symbol ready = b0

main:
` your program here
end

init_screen:
        let pins = %10000001            ` set display to ROM and XOR-Mode
        gosub cmd_write
        pause 1
        let pins = $00                  ` $00
        gosub data_write
        pause 1
        let pins = $00                  ` $00
        gosub data_write
        pause 1
        let pins = %01000000            ` Set text home position ($0000)
        gosub cmd_write
        pause 1
        let pins = $10                  ` $10
        gosub data_write
        pause 1
        let pins = $00                  ` $00
        gosub data_write
        pause 1
        let pins = %01000001            ` Set number of text area ($0000)
        gosub cmd_write
        pause 1
        let pins = $00                  ` $00
        gosub data_write
        pause 1
        let pins = $00                  ` $00
        gosub data_write
        pause 1
        let pins = %00100010            ` Set offset register
        gosub cmd_write
        pause 1
        let pins = %10100000            ` Set 8 line cursor
        gosub cmd_write
        pause 1
        let pins = %10010111            ` set display mode to text only, cursor displayed and
blinking
        gosub cmd_write
        return

data_write:
        low portc 6                     ` set for data (not command)
        low portc 7                     ` set to write
        low portc 5                     ` enable LCD
        low portc 0                     ` latch
        high portc 0
        high portc 5                    ` disable LCD
        return


cmd_write:
        high portc 6                    ` set for command (not data)
        low portc 7                     ` set to write
        low portc 5                     ` enable LCD
        low portc 0                     ` latch
        high portc 0
        high portc 5                    ` disable LCD
        return

chk_status:
        high portc 6                    ` set for command (not data)
        high portc 7                    ` set to read
        low portc 5                     ` enable LCD
```

```
        low portc 0                             ` latch
        high portc 0
        readadc 1,ready
        if ready < 250 then chk_status` check if display is ready, if not keep checking
        high portc 5                            ` disable the LCD
        return

clr_text:
        pause 1
        let pins = $00
        gosub data_write
        pause 1
        let pins = $00
        gosub data_write
        pause 1
        let pins = $24
        gosub cmd_write
        pause 1
        for b1 = 0 to 111               `loop for 112 characters (16 columns x 7 rows)
                let pins = $00
                gosub data_write
                pause 1
                let pins = $C0
                gosub cmd_write
                pause 1
        next b1
        return

clr_menu:
        pause 1
        let pins = $70
        gosub data_write
        pause 1
        let pins = $00
        gosub data_write
        pause 1
        let pins = $24
        gosub cmd_write
        pause 1
        for b1 = 112 to 127                     ` loop for 16 last characters
                let pins = $00
                gosub data_write
                pause 1
                let pins = $C0
                gosub cmd_write
                pause 1
        next b1
        return
```