Embedded Systems

Mini Project – Basic Home Autonomous

Sylvain JANNIN

H00387879

Msc Robotics

Table of content

Table of figures

1. Introduction

The goal of this mini project is to create the base of a home automation project. Even if I could have chosen a project oriented on robotic which is my favourite field in engineering, but I wanted to work on something different to change. I am really interested about home automation and I am planning to have one in the future. Working on this project was interesting and fun because it permits me to wonder about what I would like to have in my home and how I could implement it.

2. Hardware

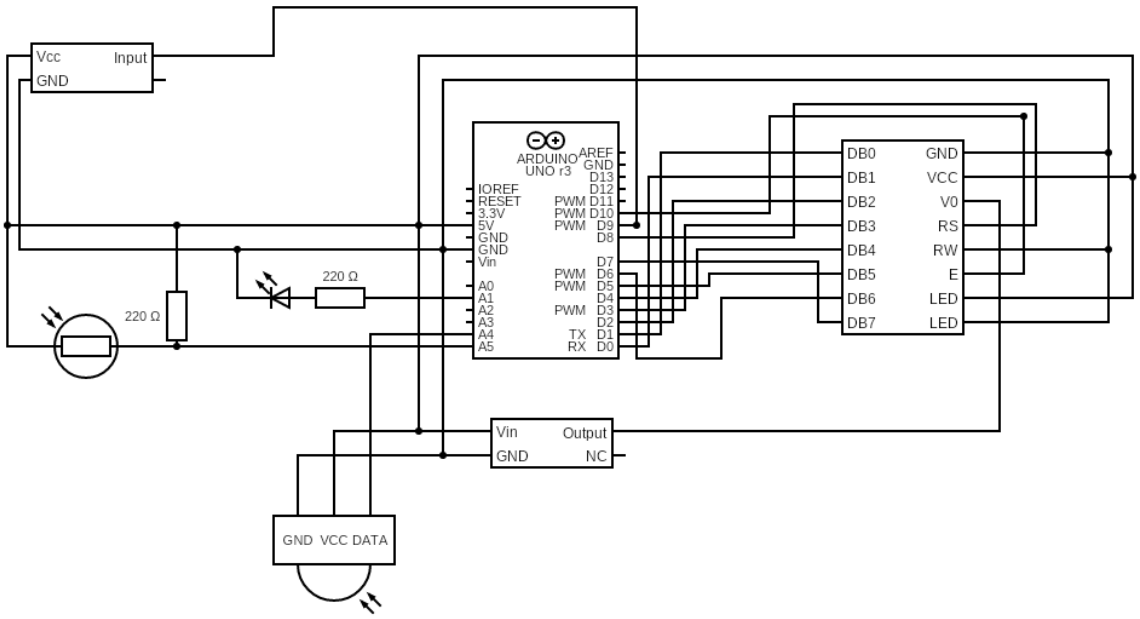

Figure 1: electronic schematic.

On the previous figure there is the project electric schematic. At the right, there is the **LCD display** connected with all the wires at the right of the Arduino (PORTB and PORTD) expect PB1 which is used for the servomotor. The integrated circuit below the Arduino is the potentiometer connected to the LCD display.

At the left, there is the other components used. The **servomotor** on the top right corner, the **light sensor** in the middle and the **PIR sensor** on the bottom. The servomotor is connected to PB1 because it uses a PWM link to a timer (timer 1 in our case).

The light sensor is connected on PC5 because we need to use an ADC to get the light. There is also a pulldown-resistor to have a better signal.

There is an LED connected to PC1 with a resistor in series. This LED represent the light in the room.

The PIR is connected to PC4. The goal of this device is to detect movements.

In the following figures we can see the final project in different states.

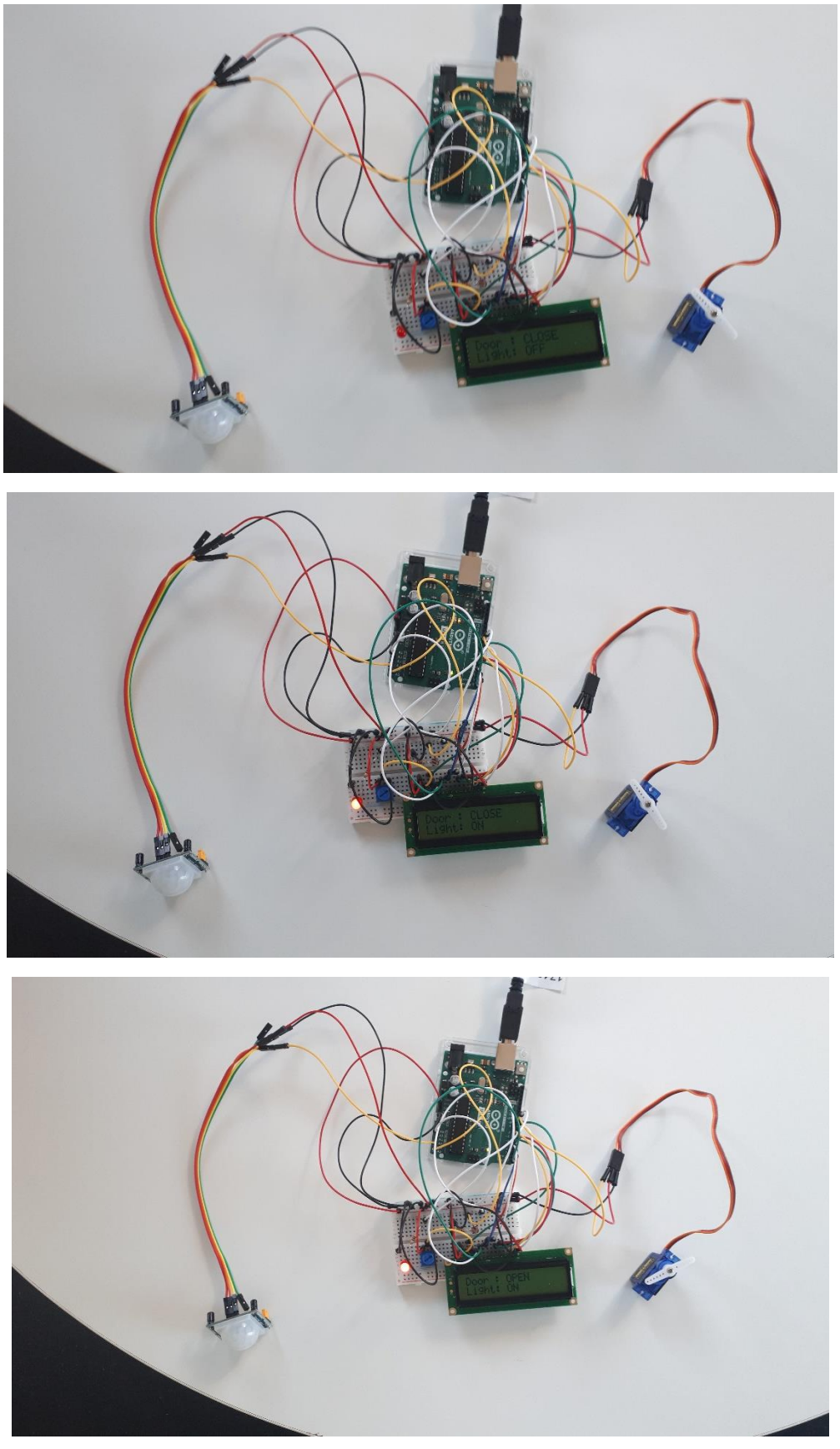

Figure 2: electornic circuit.

3. Programming

3.1. Program idea

The program is separated in 4 parts: initialisation, opening the door, turning on the light and updating the LCD.

For the first one, we created the function *init_registers* containing all the initialisations and the initialisations of the inputs and outputs. We set all PORTC to input except PC1 (the output for the LED) and all PORTB and PORTD to output. Then we initialise, the PWM, the ADC, the LCD and the timer interrupt.

Then, to open the door there is the function *Open_door*. There is an if statement which check the state of PC4 which is connected to the PIR, it is high it detects something and open the door, otherwise it closes the door. Furthermore, if it detects something it activates the timer and it raises the flags *detect_flag* and *door_flag* use for the *Light_sensor* and *update_LCD* functions. To open and close the door, we change the servo angle by changing the register *OCR1A*.

For the timer we have the function *timer_1ms* which is the timer that count for 1ms. To count for more than 1ms we use the overflow interrupt, and two global variables: *counter_overflow* and *limit_overflow*. Each time the interruption is called, it increments the overflow counter and we keep *detect_flag* high. When the overflow counter is equal to the overflow limit, we stop the timer. By the same time, we also reset the counter and we set the flag to 0. The equation is: $T = limit_{overflow} * 10^{-3}$. If we want T=5s: $Limit_{overflow} = 5000$.

The light sensor is managed in the *Light_sensor* function. We first call the *get_PotentiometerValue* function, this function reads the value of the light sensor. Then, the if statement is used to know if we must turn on or off the LED. The LED is turned on and *LED_flag* is set to 1 if the value of the light sensor is low enough if *detect_flag* is high. In other word, the statement is true if it is dark enough and if we detected something.

The LCD is updated within the *update_LCD* function. We create 2 strings: *data_soor* and *data_light* which are used to store the data we sent to the LCD. The data contain the information about the door and the light: is the door open or closed and is the light on or off. To know the states, we use the 2 flags *door_flag* and *LED_flag*. If the door is open *door_flag* is high and if the light in turned *LED_flag* is high. To write on the LCD we use the function *Lcd8_Write_String* two times: one for *data_door* on the first line, one for *data_light* on the second line. to change the cursor position we use the function *Lcd8_Set_Cursor*.

The next figure represents the flowcharts of the code.

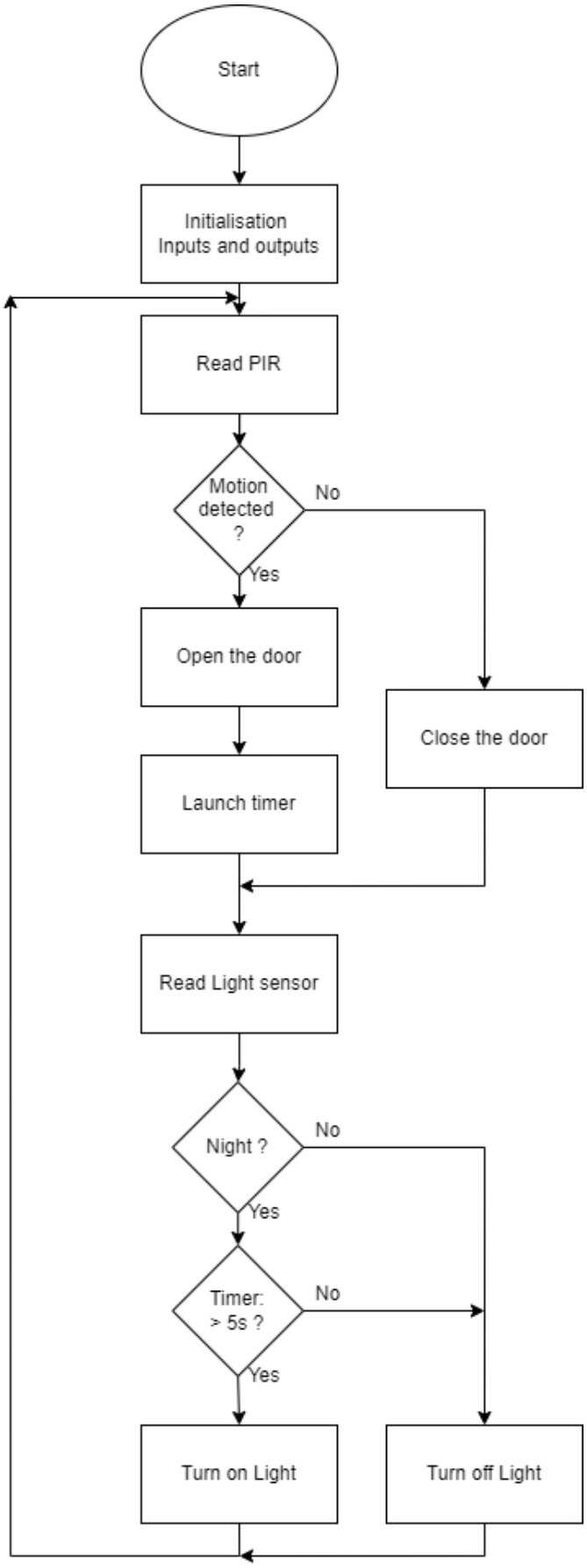

Figure 3: code flowchart.

3.2. Registers

3.2.1. Timer

We want a timer of 1ms with timer 0 coded on 8 bits, we cannot use timer 1 because this one is already used for the servomotor. the clock frequency is 16MHz and we choose a prescaler of 64. To have a prescaler of 64 we set: *TCCROB* = 0b000000011. With the prescaler it becomes:

$$f_{pres} = \frac{f_{clk}}{prescalaire} = \frac{16 \times 10^6}{64} = 250\,000\ Hz$$

Thanks to the prescaler we have a slower clock, now we calculate the value used to calculate the *TCNT0* value to have a 1ms timer.

$$N = T_{delay} \times f_{pres} = 10^{-3} \times 250\,000 = 250$$

We want the timer to count 250 ticks before an overflow:

$$TCNT0 = 2^8 - 1 - N = 256 - 1 - 250 = 5$$

When we want the timer to stop, in the overflow interrupt we set to 1 *TOV0* in *TIFR0* register and *TCCR0B* register to 0.

To summarize: *TCCROB* = 0x03 and $TCNT0$ = 5 when the timer is on, otherwise we *TOV0 = 1* and *TCCR0B* = 0x00.

3.2.2. Interrupt

We want an interruption on every timer 1 overflow. For this we have *TIMER0_OVF_vect*. To initialise it, we set *TOEI0* to one in *TIMSK0* register and then we call *sei* in the initialisation to activate global interrupt.

3.2.3. PWM

We init the PWM in We use a PWM on fast pwm mode with a prescaler of 8 and we use the non-inverting mode. We use the first timer, so the output is on PB1. For the fast PWM we need *WG11, WG12* and *WG13* set to 1. To have a prescaler of 8, for this we set *CS11* to one. For the non-inverting mode, we set *COM1A1*. We also set *ICR1* to 40000 to define the PWM period.

For the servo angle, we need to change the value of *OCR1A*. We want the servo to open and close the door, so we need a difference of 90°. We chose 0° when it is closed and 90 when it is open. Our servo motor works for a duty cycle from 2.5% to 12.5%

$$\frac{\alpha}{100} = \frac{OCR1A}{\text{ICR1A}}$$

$$OCR1A = \frac{40000\ \alpha}{100} = 400\ \alpha$$

- Angle = 0° → α = 7.5 → $OCR1A = 400 \times 7.5 = 3000$
- Angle = 90° → α = 12.5 → $OCR1A = 400 \times 12.5 = 5000$

3.2.4. ADC

The registers of the ADC are set in the "*initADC*" function. We take as reference voltage AVCC (5V) and we plug the output of the potentiometer in PC5. We also want to use a prescaler of 8 and we want the ADC to be enable

- *ADMUX*:
  - *REFS0* is set to 1 to take *AVCC* as reference.
  - *MUX0* and *MUX2* are set to 1 to use ADC on PC5.
- *ADCRSA*:
  - *ADPS0* and *ADPS1* are set to one to have a prescaler of 8.
  - *ADEN* is set to 1 to enable the ADC.

Once the registers are set, we set to 1 *ADSC* in *ADCRSA* to enable ADC. To read the value we read registers *ADCL* and *ADCH*, the *ADCL* is the low value and *ADCH* the high value. In the end, the value given by the *ADC* is "*ADCH + ADCL*". But, since The ADC is coded on 10 bits, we have to do an extra step to have the correct value. Before adding them to obtain the value we need to shift *ADCH* of 8 bits.

4. Conclusion

This project is more a proof of concept or a teaser about how could look an autonomous home. There are so many more things to implement, like managing the windows and checking the weather, checking the heat and the most interesting one: making the home controllable via a smartphone application.

For the case of the mini project, we managed to implement and to make the prototype the way we want. One thing to improve is to change the way we update the LCD, instead having a if every x iteration, we could use another timer that update the LCD every second.

5. Annexe – Code

```
/*
 * Mini_Project__JANNIN_Sylvain.c
 *
 * Created: 17/11/2021 14:41:22
 * Author : Sylvain Jannin - H00387879
 */

#define D0 eS_PORTD0//For LCD library
#define D1 eS_PORTD1
#define D2 eS_PORTD2
#define D3 eS_PORTD3
#define D4 eS_PORTD4
#define D5 eS_PORTD5
#define D6 eS_PORTD6
#define D7 eS_PORTD7
#define RS eS_PORTB0
#define EN eS_PORTB2
#define F_CPU 16000000UL
#include <avr/io.h>
#include "lcd.h"
#include <util/delay.h>
#include <string.h>
#include <avr/interrupt.h>

//init global variables
int counter_overflow;      //count the number of time the timer has overflowed
int limit_overflow;        //number of overflowed. These 2 variables have a timer used
int detect_flag;           //indicate if it has detected someone for the last "x"
milliseconds (the x values depend on the values of limit_overflow: if limit_overflow =
1000 -> x = 1000 ms)
int LED_flag;              //Indicates if the LED is on or not.
int door_flag;                     //Indicates is the door is open or not.



// ---------------- init register ----------------
void timer_1ms()
{
       int N = 250 ;
       TCNT0 = 256 -1 - N;
       TCCR0A = 0x00;
       TCCR0B = 0b00000011;//prescaler of 64
}

void init_PWM()
{
       DDRB |= (1 << PINB1); // Set pin 9 to output PWM
       /* 1. Set Fast PWM mode 14: WGM11, WGM12, WGM13 to 1*/
       /* 2. Set pre-scale of 8 */
       /* 3. Set Fast PWM non-inverting mode */
       TCCR1A |= (1 << WGM11) | (1 << COM1A1);
       TCCR1B |= (1 << WGM13) | (1 << WGM12) |(1 << CS11);
       /* 4. Set ICR1: ICR1 is the top defining PWM period */
       ICR1 = 40000;
}


void initADC(void)
{
```

```
	ADMUX |= (1 << REFS0) | (1 << MUX2) | (1 << MUX0); //reference voltage on AVCC,
MUX = 0101 -> ADC5
	ADCSRA |= (1 << ADPS1) | (1 << ADPS0);		//ADC clock prescaler / 8
	ADCSRA |= (1 << ADEN);							//enables
the ADC
}


void init_registers()
{

	DDRC = 0;	//set PORTC as Input
	DDRD = 0xFF;//set PORTD as output
	DDRB = 0xFF;//set PORTB as output

	DDRC |= 1 << PINC1; //set PC1 as output //for the LED

	init_PWM();
	initADC();
	Lcd8_Init();  //init LCD

	TIMSK0 = (1 <<TOIE0); //enable Timer0 overflow interrupt
	sei();			//enable interrupt

}


// ---------------- function ----------------
uint16_t get_PotentiometerValue()
{
	uint16_t potentiometerValue;
	initADC();
	ADCSRA |= (1 << ADSC); //start ADC conversion
	while((ADCSRA & (1 << ADSC))) //wait until ADSC bit is clear, i.e., ADC
conversion is done
	{}
	//read ADC value
	uint8_t theLowADC = ADCL;
	potentiometerValue = ADCH << 8 | theLowADC;		//shift ADCH of 8 bits
	return potentiometerValue;
}



void Open_door()
{
	if(PINC & (1 << PINC4))	//if PINC4 is high = if it detects something
	{
		OCR1A = 5000;		//Open the door
		timer_1ms();		//start the timer
		detect_flag = 1;	//raise detect_flag
		door_flag = 1;			//raise door_flag
	}
	else
	{
		OCR1A = 3000;		//close the door
		door_flag = 0;			//Low door_flag
	}

}
```

```
void Light_sensor()//need to check a timer
n,
{
	uint16_t potentiometerValue;
	potentiometerValue = get_PotentiometerValue();		//Read the light sensor
	if(potentiometerValue < 510 && detect_flag == 1)	//if it is too dark + it
detects something
	{
		PORTC |= (1 << PINC1);	//Turn on led PC1
		LED_flag = 1;		//Raise flag
	}
	else
	{
		PORTC &= ~(1 << PINC1); //Turn off led PC1
		LED_flag = 0;		//Low flag
	}
}

void update_LCD()
{
	Lcd8_Clear();		//clear the LCD

	//init variables we send to the LCD
	char* data_door;
	char* data_light;

	//Is door open or close
	if(door_flag == 1)
	{
		data_door = "Door : OPEN";
	}
	else
	{
		data_door = "Door : CLOSE";
	}

	//Light on or off
	if(LED_flag == 1)
	{
		data_light = "Light: ON";
	}

	else
	{
		data_light = "Light: OFF";
	}

	Lcd8_Set_Cursor(0,0);	//set the position of the cursor on the first line
	Lcd8_Write_String(data_door);//write on the LCD data_door
	Lcd8_Set_Cursor(2,0);	//set the position of the cursor on the second line
	Lcd8_Write_String(data_light);//write on the LCD data_light
}


// ---------------- interrupt ----------------
ISR(TIMER0_OVF_vect)//TimerOverflow interrupt
{
	counter_overflow ++;//each time the interrupt occurred
	detect_flag = 1;		//raise the detect_flag
	if(counter_overflow >= limit_overflow)   //If the number of overflowed is
reached
	{
```

```
                //Stop timer
                TCCR0B = 0;
                TIFR0 = (1<<TOV0);
                counter_overflow = 0;       //reset the counter
                detect_flag = 0;            //Low the flag
        }
}

// ---------------- Main ----------------
int main(void)
{
        limit_overflow = 5000;      //The timer count for 1ms, the value of limit
overflow makes the timer having a period of : T = 1ms * limit_overflow. In our case it
is five seconds
        counter_overflow = 0;       //count the number of overflow before the limit
        double counter_LCD = 0;

        init_registers();           //Initialiase registers, Input and output

        while(1)
        {
                Open_door();        //Manage the door: actuator and servo
                Light_sensor();             //Manage the light

                counter_LCD ++;
                if(counter_LCD >= 10000)//We cannot rewrite on the LCD for every
iteration and adding a delay would the 2d timer working properly, so to have a quick
implementation we update it for every x iterations, the values has been choosed if the
result was satifaying enough since we do not have constraint enough
                {
                        update_LCD(); //update the LCD
                        counter_LCD=0;      //reset counter
                }
        }
        return(1);
}
```