Automation of tasks in a greenhouse

David Diaz Martinez

Summary

Project Specifications

The main objective of this project is to build an automated system in a greenhouse capable of making decisions based on the measurements of environmental variables using the STM32 NUCLEO-F446RE MCU and programming it by means of the methods acquired through the course.

Automation of tasks in a greenhouse.
The system is accompanied by compatible sensors and actuators, capable of monitoring the behavior of environmental variables temperature and air humidity. An automated irrigation system capable of making decisions based on these measurements will be designed and implemented.

Control Algorithm.
The control of the greenhouse that is proposed considers that each plant has climatic conditions for its development that differ slightly. These parameters are stored in memory and are updated only in case of receiving the order to change the type of crop.

Temperature control.
One common system is using a called zenithal ventilation, which is a kind of moving window installed in the roof that allows escaping hot air from inside. It is activated by means of the geared motor that will move the rack which in turn moves the window.
Considering the measurement of a temperature sensor, control needs to be implemented acting on the opening angle of the window (by a servomotor) to meet this desired temperature setpoint.

Fig 1. Examples of zenithal ventilation for temperature control

Humidity Control: Ventilation control is essential to generate the appropriate humidity inside the greenhouse, for this, there are several methods, one of them is the incorporation of side fans that can extract excess humidity inside the greenhouse.

Fig 2. Examples of ventilation control for humidity regulation

And last, there is the control of the irrigation system that is governed by the time of day and the kind of plant. The irrigation pump is turned on at a certain moment of the day (10:00 am), carrying the water to the ground through sprinklers until one of these conditions be meet:

- Overcome certain threshold on soil humidity (measured by a sensor).
- After a certain period of time

Both conditions depend on the type of crop and are updated when the button (for change it) is pressed.

Inputs

- Temperature sensor.
- Ambient humidity sensor.
- Crop button selector
- Soil humidity sensor (Analog potentiometer)

Outputs

- 12V DC motor
- Servomotor
- Relay (Irrigation Pump)
- LCD Display.

We consider 3 types of crops, Tomato, Lettuce and Cucumber. The parameters are shown below:

Tomato:
Temperature: 20 - 30°C.
Relative soil humidity: 60– 75%
Irrigation time: 2h

Lettuce:
Temperature: 15– 18°C.
Relative soil humidity: 60– 80%
Irrigation time: 1h.

Cucumber:
Temperature: 22 – 26°C.
Relative soil humidity: 80%.
Irrigation time: 1:20 h.

The system is formed by:

STM32F446RET6 MCU
JGA25-371 12V DC Motor
MG90S Servomotor
ELEGOO 16x2 LCD Module
AZDelivery LCD I2C Serial Interface
DTH11 Temperature/Humidity Sensor
L298N Motor Shield.
470 Ω Potentiometer
10K Resistor
EFISH & ALSISK 12V 1A Power Adapter

Environmental variables, temperature and humidity are measured, then the PID algorithms calculate the needed angle output for the damper system to regulate the airflow (temperature regulation) and simultaneously regulate the fan speed (humidity regulation) for extract humidity excess.

The main variables are then shown in a display, which are:

- Measured Temperature
- Measured Humidity
- Irrigation Time
- Setpoints

For a clear understanding, the next diagram shows the flowchart of the main operations performed in this project. Later will be explained how are implemented every single one of them.

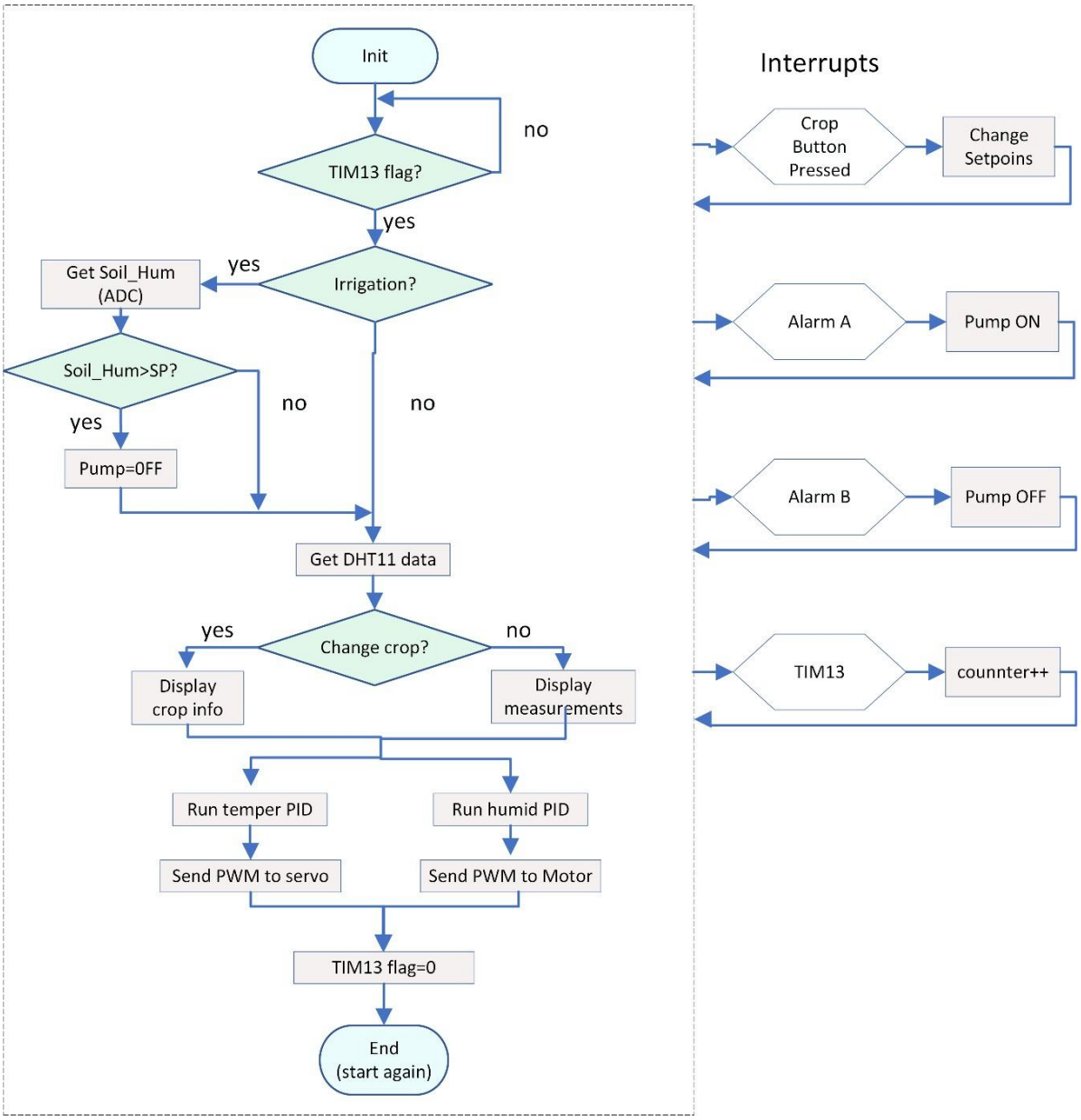


Fig 3. Flowchart of the code

System Construction and Implementation

Below the system wiring diagram is shown:

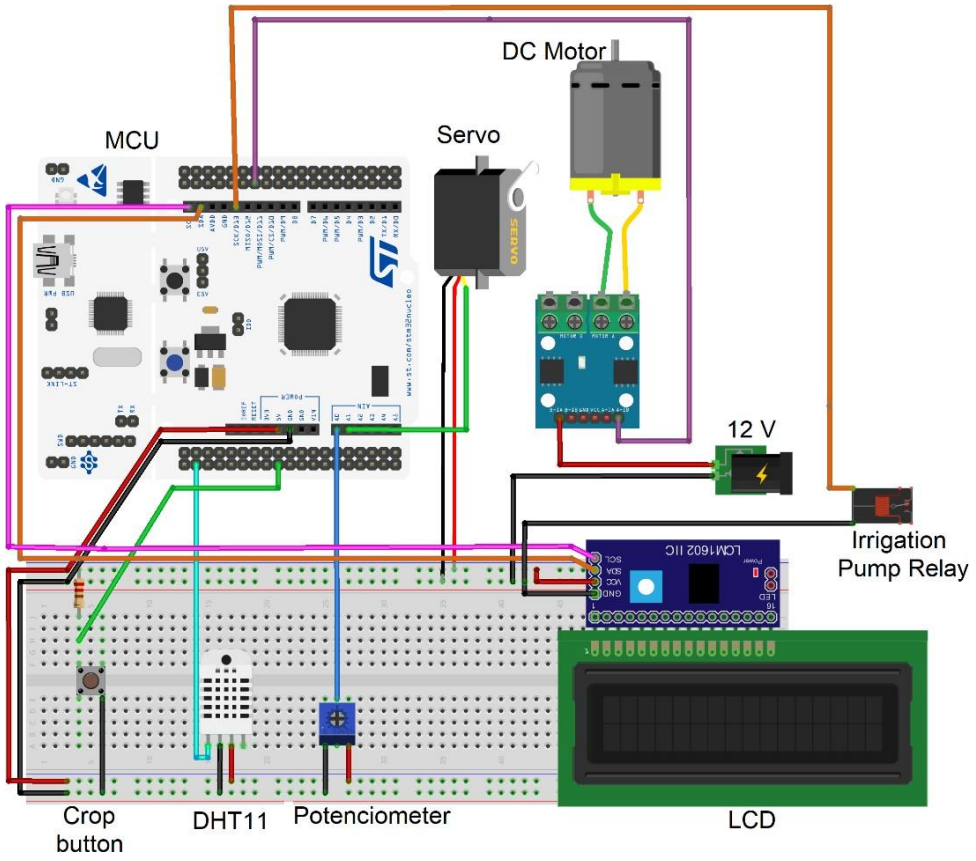


Fig 4. Connection diagram of the system

Implementation

After mounting the parts and components in the breadboard, it's necessary organize the connection with the MCU board, and the correct assignment of the I/O ports to each peripheral and device used.

Next table shows how the distribution of the MCU I/O.

Table 1. distribution of the I/O ports and resources in the MCU.

GPIO		Function	Description	Purpose
Port	Pin			
A	0	ADC1_IN0	A/D Converter 1 Channel 0	Soil humidity sensor
	1	TIM2_CH2	Timer 2 Channel 2 (PWM Generation mode)	Servomotor Position Control output
	5	GPIO_Output	Output Port	Irrigation Pump (LED)
	6	TIM3_CH1	Timer 3 Channel 1 (PWM Generation mode)	Motor Speed Control output
	15	GPIO_Input	Digital Input	Crop selector button
B	8	I2C1_SCL	I2C1 Serial Clock	I2C-LCD Module output
	9	I2C2_SDA	I2C1 Serial Data	
C	12	GIPI_InOut	DHT11 sensor data pin	Read data from DHT11
-	-	TIM1	Timer 1 (Internal)	Operation of DHT11

		TIM2	Timer 2 (Internal)	PWM for servo
		TIM3	Timer 3 (Internal)	PWM for DC Motor
-	-	TIM13	Timer 13 (Internal)	Sampling time and PID
-	-	RTC	Real Time Clock (Internal)	Irrigation operation

Components and Libraries implementation

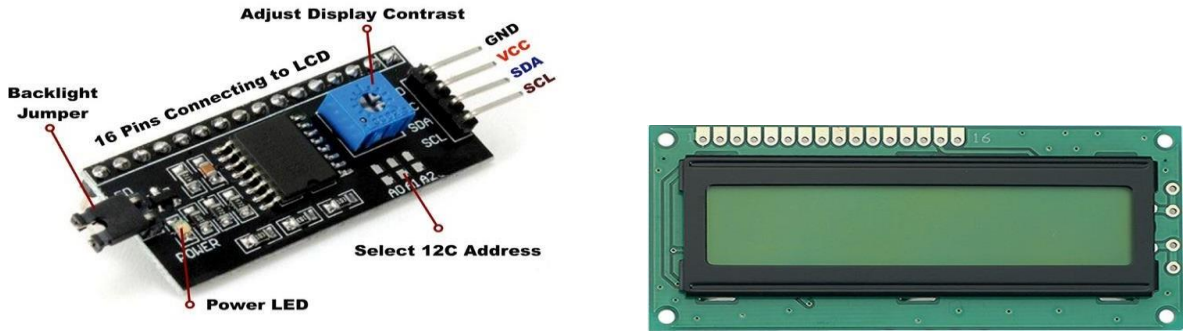
I2C-LCD Serial Interface

For the I2C-LCD Serial Interface, a library was used so it would be much easier to manage this device (LCD display). In the header .h file are defined the different variables, constants, and structures so as the respective functions. Then the implementation of such functions is carried on the source .c file.

Due to the limited pin numbers on the NUCLEO-F446RE MCU and having into account that the used LCD in this project has a 16-pin layout, an interface device is used to save space on the board for being used in other peripherals.

The interface adapter is a module PCF8574 chip wich makes the work easy with just two pins. The serial interface adapter is connected to the 16x2 LCD and provides two signal output pins (SDA and SCL) which are used to communicate with the MCU. A 10k potentiometer allows adjustment of the LCD contrast, the 'LED' pins control the LCD backlight.

- Operating Voltage: 5V DC
- I2C control using PCF8574
- Can have 8 modules on a single I2C bus
- I2C Address: 0X20~0X27



a) b)

Fig. 5 a) Module PCF8574 b) LCD display

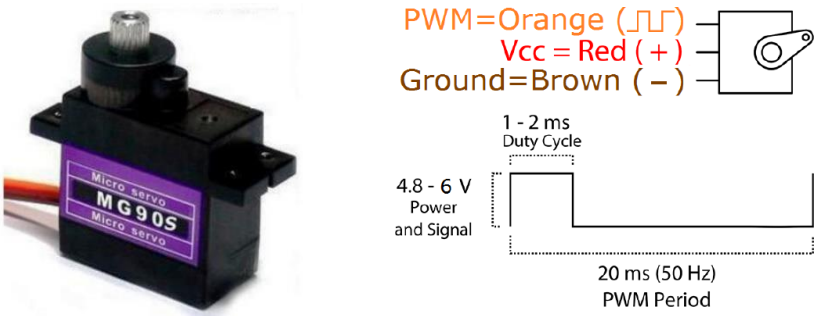
The `extern I2C_HandleTypeDef hi2c1` states the handler declared after the configuration on the CubeMX is made and the code on the `main.c` generated and can be changed accordingly.

Then the functions declared for managing the communication are:

- `void lcd_init (void)` //initializes the LCD in 4-bit mode
- `void lcd_send_cmd (char cmd)` //sends the commands to the LCD
- `void lcd_send_data (char data)` //sends data to the LCD
- `void lcd_send_string (char *str)` //sends a string to the LCD
- `void lcd_put_cur(int row, int col)` //puts the cursor on desired position on LCD
- `void lcd_clear (void)` //clear the LCD

Servomotor

MG90S Servo can rotate approximately 180 degrees (90 in each direction) and works just like the standard kinds but smaller. Tiny and lightweight with high output power.



a) b)

Fig. 6 Servomotor MG90S. a) Body. b) Pin diagram and period requirements

Specifications

- Weight: 13.4 g
- Dimension: 22.5 x 12 x 35.5 mm approx.
- Stall torque: 1.8 kgf·cm (4.8), 2.2 kgf·cm (6V)
- Operating speed: 0.1 s/60 degree (4.8 V), 0.08 s/60 degree (6 V)
- Operating voltage: 4.8 V – 6.0 V
- Dead band width: 5 µs

Position control

- "-90°"---- (1 ms pulse) is all the way to the left
- "0°" ------ (1.5 ms pulse) is the middle,
- "90°"----- (2 ms pulse) is all the way to the right

The MG90S servomotor angle is controlled through a 100 Hz PWM and a duty cycle that varies T_{ON} between 1 and 2 milliseconds.

After configured the timer used for this purposes (TIM2 channel 2), the duty cycle is changed during the operation by the following instruction:

```
htim2.Instance->CCR2=(PID_output);
```

which sets the TIM Compare and Capture Register value on runtime

DHT11 Humidity/Temperature Sensor.

The DHT11 is a low-cost, medium-accuracy humidity/temperature sensor. This sensor includes a component for humidity measurement and an NTC (Negative Temperature Coefficient) thermistor for temperature measurement.

Table 2: DHT11 Technical parameters

Parameter	DTH11
Feeding	3VDC ≤ VCC ≤ 5VDC
Output signal	Digital
Temperature measurement range	De 0 a 50 ºC
Accuracy Temperature	±2 ºC
Resolution Temperature	0.1 ºC
Humidity measurement range	De 20% a 90% RH
Accuracy Humidity	4% RH
Moisture Resolution	1% RH
Sensing Time	1s
Size	12 x 15.5 x 5.5 mm

Its supply voltage is 3.5 – 5 V and provides a digital data output. A great advantage of this module is the protocol it uses for data transfer. All sensor readings are sent using a single line thus reducing costs and extending distances. A communication process lasts about 4 ms. The data consists of decimal and integral parts. The sensor sends a high bit first and the entire data is 40 bits.

Data format:

8 bits	integral RHI	Relativity Humidity Integral Part
8 bits	decimal RHD	Relativity Humidity Decimal Part
8 bits	integral TI	Temperature Integral Part
8 bits	decimal TD	Temperature Decimal Part
8 bits	Check bit.	Sum of previous data

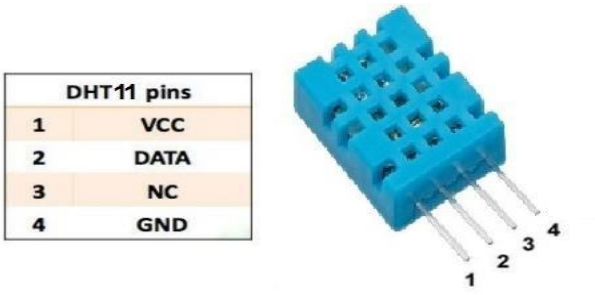


Figure 7: DHT11 Humidity/Temperature Sensor

Procedure for communication and data collection:

1. The "start" signal is sent to the device. While the data bus is free, a Vcc voltage is present. To start, the microcontroller must pull the voltage level to ground for at least 1 mS to ensure that the sensor has detected the signal. Next, wait for the response between 20 and 40 µS.

2. The DHT11 sends the response to the microcontroller. Once the start order has been detected, the sensor responds with a signal at a low voltage level for 80 µS and then at VCC for the same duration.

3. Finally, the sensor returns the measurement data to the MCU. Each transmitted bit starts with a low voltage level for 50 µS, then the duration of the high level pulse determines the value of the corresponding bit. As shown in Table 3.

Table 3: Bit codification in DHT11.

Pulse duration	Bit value
26-18 µS	'0'
70 µS	'1'

Four functions are involved in all this process of reading the data. They are grouped into a separated header file called `DHT11.c` (and `DHT11.h`).

```
void GetMeasurementsDHT11 (float *temperat, uint8_t *humid)
void microDelay (uint16_t delay)
uint8_t DHT11_Start (void)
uint8_t DHT11_Read (void)
```

Motor Shield (L298N).

This shield allows to drive one channel for stepper motors and two for DC motors. Use the L298N chip which delivers 2 A output current for each channel. Speed control is carried out through the conventional PWM. It is highly recommended to use an external power supply. The shield supports PWM and PLL (Phased Locked Loop) control modes.

Specifications:

- Control logic voltage：5V
- Motor Drive Voltage： 4.8-35 V (from Arduino or External Source)
- Logical power supply Iss： ≤36mA
- Motor supply current Io： ≤2 A
- Maximum power consumption：25W（T=75℃）
- Speed control by PWM or PLL
- Control signal level:
  - High：2.3V≤Vin≤5V
  - Low：-0.3V ≤ Vin ≤ 1.5V

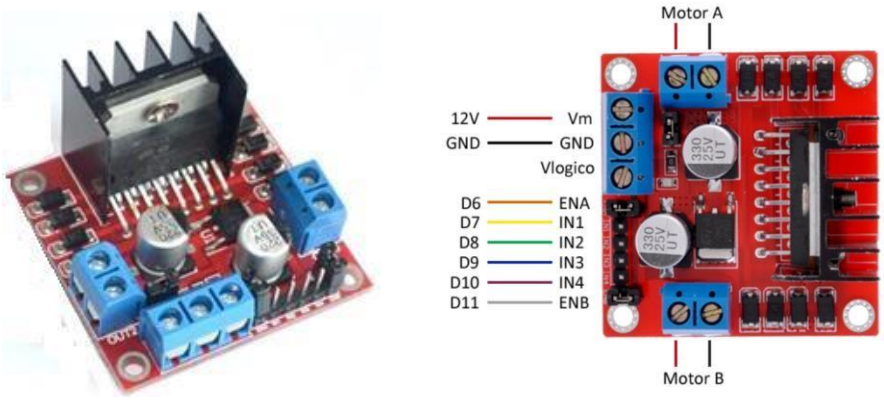


Fig. 8: L298N, DC and stepper motor control shield.

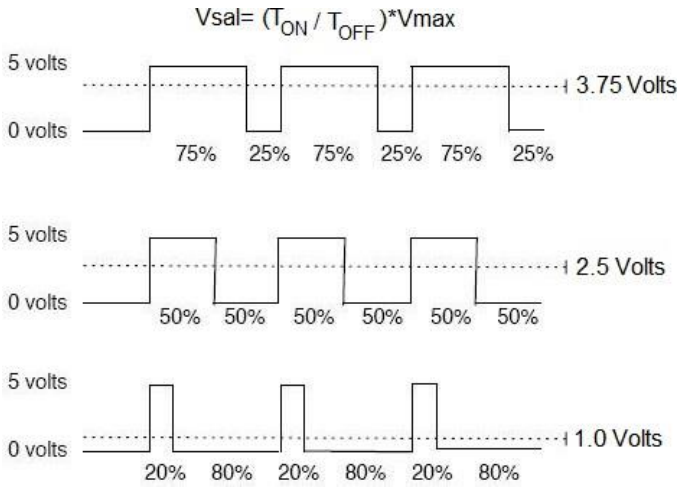


Fig 9: Speed control method using PWM.

The voltage input provides the voltage that will power the motors. The admissible input range is from 3V to 35V and is supplied through the 2 left terminals of the input connection terminal.
Lastly, we have the input pins that control the direction and speed of rotation.

- The IEA, IN1, and IN2 pins control output A.
- The IEB, IN3, and IN4 pins control output B.

The IN1, IN2, and IN3 and IN4 pins control the direction of rotation, respectively, of output A and B.

The PWM to the motor is generated by TIM3 channel 1 with a frequency of 10 KHz and during execution the duty cycle is changed by the following instruction:

```
htim3.Instance->CCR1=(PIDoutput);
```

DC Motor

- Motor size: 70mm x 22mm x 18mm
- Engine Weight: 50g
- Voltage between 3 V and 12 V (recommended 6 to 8 volt)
- Speed: 6000 RPM
- Current: 80-100mA

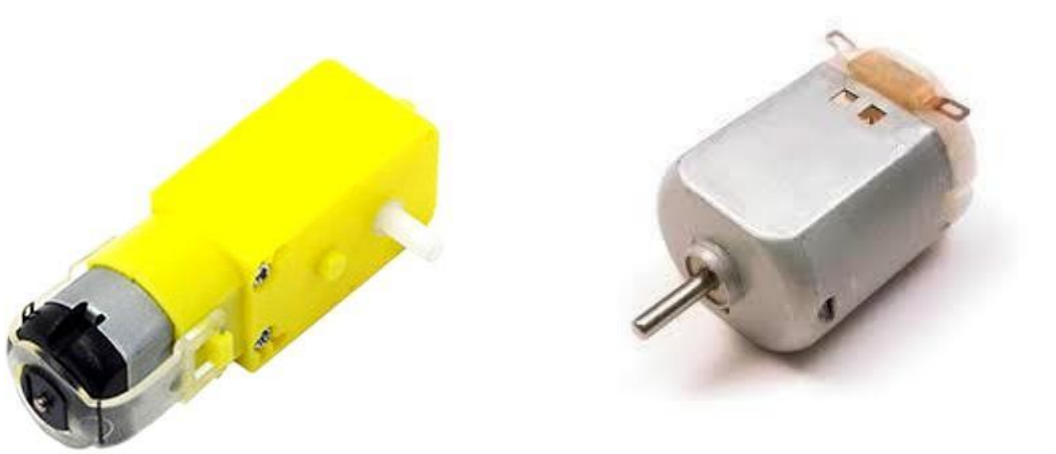


Fig 10. DC motor

Relay for irrigation Pump

Fig 11: Relay for activate the irrigation pump

The irrigation system consist in a series of sprinkles disperses for all the place. The water flows by the irrigation pump (activated by a relay). The output PA.5 is used for these purposes.

PID Implementation

In order to effectively maintain the desired humidity through the fan velocity and the temperature by the angle of the dampers, is necessary the implementation of a Proportional-Integral-Derivative (PID) Controller.

The algorithm is as follows:

```
error=SP-Input;                          //error term
iTerm+=(ki*T*error);                     //Integral Term
deriv=kd*(Input-LastInput)/T;            //derivative term
Output =  kp*error + iTerm - deriv;      //controller output
LastInput = Input;                       //update last value
return Output;
```

where `Kp` is proportional constant, `Ki` is Integral constant and `Kd` is Derivative constant.

The functions operate on a single sample of data and each call to the function returns a single processed value.

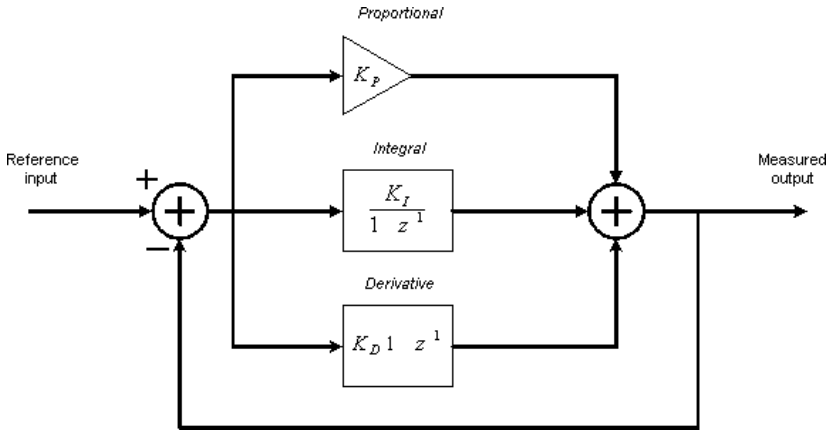


Fig 12: PID Structure

The PID controller calculates an "error" value as the difference between the measured output and the reference input. The controller attempts to minimize the error by adjusting the process control inputs. The proportional value determines the reaction to the current error, the integral value determines the reaction based on the sum of recent errors, and the derivative value determines the reaction based on the rate at which the error has been changing. An instance structure is defined for each PID Controller.

PID Tuning

Before the PID can be fully implemented, it is vital to correctly calibrate its parameters. In terms of simplicity and accuracy, the Ziegler-Nichols Sustained Oscillation Method is the most suitable approach in this case.

This method starts by zeroing the integral and differential gains and then raising the proportional gain until the system is unstable. The value of K_P at the point of instability is called K_{crit}; the frequency of oscillation is f_0. The method then backs off the proportional gain a predetermined amount and sets the integral and differential gains as a function of f_0. The P, I, and D gains are set accordingly, as shown in the following figure.

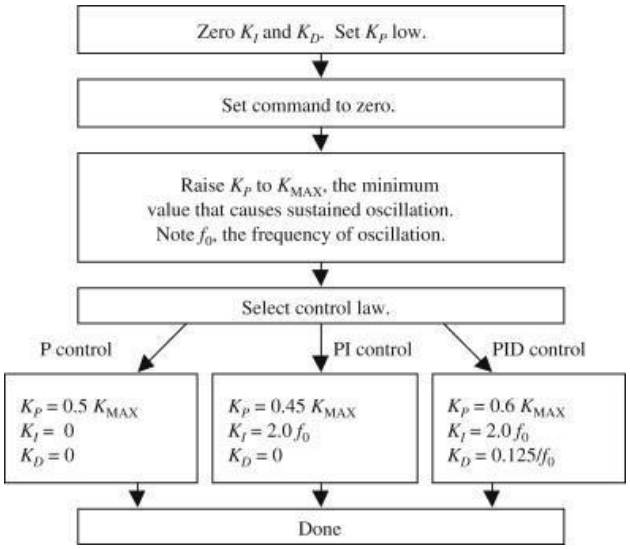


Fig 13: Ziegler-Nichols Sustained Oscillation Method

Program code

All the programming was done in the STM32CubeIDE platform. The configuration of devices and resources was done using the embedded graphical interface CubeMX Device Configuration Tool, which initializes and configures elements one by one in a very intuitive and easy mode, by using HAL functions, saving a considerable amount of time.

ADC

The Analog/Digital Converter (ADC1) uses channel IN0 (PA.0) and has been configured as follow:

ADCs_Common_Settings	
Mode	Independent mode
ADC_Settings	
Clock Prescaler	PCLK2 divided by 2
Resolution	8 bits (11 ADC Clock cycles)
Data Alignment	Right alignment
Scan Conversion Mode	Disabled
Continuous Conversion Mode	Disabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Disabled
End Of Conversion Selection	EOC flag at the end of single channel conversion

```
/** ===== ADC read========================================*/
uint8_t adcRead(void){
HAL_ADC_Start(&hadc1);                    // Start ADC Conversion
HAL_ADC_PollForConversion(&hadc1, 1);     // Poll ADC1 Peripheral & TimeOut = 1mSec
adc_value = HAL_ADC_GetValue(&hadc1);     // Read The ADC Conversion
return adc_value;}
```

Timers

As mentioned before, there are 4 timers used in this program for different purposes.

Timmer 1 is configured and used for operation of DHT11 sensor. Can generate signals with a frequency of f=1 MHz, with subsequent delays in the order of uSecs.

Prescaler (PSC - 16 bits value)	84-1
Counter Mode	Up
Counter Period (AutoReload Registe...	65535
Internal Clock Division (CKD)	No Division
Repetition Counter (RCR - 8 bits val...	0
auto-reload preload	Disable

Inside the function created for manage operation and communication with DHT11, several times is called the function:

```
microDelay(40);
```

And this function contains the procedure for generating delays with the desired micro-seconds

```
void microDelay (uint16_t delay){
  __HAL_TIM_SET_COUNTER(&htim1, 0);
  while (__HAL_TIM_GET_COUNTER(&htim1) < delay);}
```

Timer 2 generates the PWM signal to manage the servo angle, in this case, as stated before, the servo works using a frequency of 100 Hz (10 ms period of PWM), so the configuration is as follows:

$$f_{PWM} = \frac{f_{CLK}}{(ARR+1)(PSC+1)}$$

$$(ARR+1)(PSC+1) = \frac{f_{CLK}}{f_{PWM}}$$

$$(ARR+1)(PSC+1) = \frac{84\,000\,000\ Hz}{100\ Hz}$$

$$(ARR+1)(PSC+1) = 840\,000$$

Counter Settings	
Prescaler (PSC - 16 bits value)	840-1
Counter Mode	Up
Counter Period (AutoReload Regist...	1000-1
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable
Trigger Output (TRGO) Parameters	
PWM Generation Channel 2	
Mode	PWM mode 1
Pulse (32 bits value)	0
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High

Timer 3 has been set in PWM generation mode for controlling the speed of the fan motor trough the L298N motor driver. It's decided to use a working frequency of 10 kHz, to avoid noise, given the MCU clock frequency at 84 MHz, it's necessary to calculate the Auto-Reload Register (ARR) and the Pre-scaler (PSC), then:

$$f_{PWM} = \frac{f_{CLK}}{(ARR+1)(PSC+1)}$$

$$(ARR+1)(PSC+1) = \frac{f_{CLK}}{f_{PWM}}$$

Prescaler (PSC - 16 bits value)	840-1
Counter Mode	Up
Counter Period (AutoReload Regist...	1000-1
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable

$$(ARR+1)(PSC+1) = \frac{84\,000\,000\ Hz}{10\,000\ Hz}$$

$$(ARR+1)(PSC+1) = 8\,400$$

The **Timer 13** is an internal timer used for generate a Time-base for all operations that take place in main loop (sampling the sensor measurements, execute PID control algorithm and output to the motor and the servo respectively. Taken into account that the enviromental variables that we are monitoring has a very small response time (in the order of several minutes, even hours), there is no need for a high sampling time. Can be selected every 1 second. But we select a period of 0.2 s (5 Hz).

$$f_{event} = \frac{f_{CLK}}{(ARR+1)(PSC+1)}$$

$$(ARR+1)(PSC+1) = \frac{84\,000\,000\ Hz}{5\ Hz}$$

$$(ARR+1)(PSC+1) = 16\,800\,000$$

Prescaler (PSC - 16 bits value)	8400-1
Counter Mode	Up
Counter Period (AutoReload Registe...	2000-1
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable

This timer is used in interruption mode with a high priority to guarantee the correct sampling of the input/output signals of the system.

RTC

The board provides an additional tool, very convenient for our purposes. The Real-Time Clock (RTC) is an independent BCD timer/counter which provides a time-of-day clock/calendar, two programmable alarm interrupts, and a periodic programmable wakeup flag with interrupt capability. Two 32-bit registers contain the seconds, minutes, hours, day (day of week), date (day of month), month, and year, expressed in binary coded decimal format (BCD). As long as the supply voltage remains in the operating range, the RTC never stops, regardless of the device status.

RTC has the possibility of trigger two alarms in interrupt mode completely configurable and we use this advantage for turn on the irrigation pump at one specific hour (10 am) every day and keep it turned on during a time (depending on selected crop).

```
//=========== Alarm A (Start Irrigation) 10:00 AM ==================
void HAL_RTC_AlarmAEventCallback(RTC_HandleTypeDef *hrtc){
	HAL_GPIO_WritePin (GPIOA, GPIO_PIN_5, 1); // turn on the irrigation pump
	irrigation=1;                             //irrigation starts
}
//=========== Alarm B (Stop Irrigation) ==================
void HAL_RTCEx_AlarmBEventCallback(RTC_HandleTypeDef *hrtc){
	RTC_DateTypeDef sDate = {0};
	HAL_GPIO_WritePin (GPIOA, GPIO_PIN_5, 0); // turn off the pump
	irrigation=0;                              //irrigation pump stops
	if (days_count==6){                        // update next weekday on RTC
		days_count=0;}
```

```
        else{days_count=days_count+1;}
        sDate.WeekDay = days[days_count];
        if (HAL_RTC_SetDate(&hrtc, &sDate, RTC_FORMAT_BCD) != HAL_OK)
        {Error_Handler();}
}
```

The configuration of the RTC is shown.

General	
Hour Format	Hourformat 24
Asynchronous Predivider value	127
Synchronous Predivider value	255
Calendar Time	
Data Format	BCD data format
Hours	9
Minutes	59
Seconds	40
Calendar Date	
Week Day	Tuesday
Month	May
Date	10
Year	22

Alarm A	
Hours	10
Minutes	0
Seconds	0
Alarm Date Week Day Sel	Weekday
Alarm Week Day	Tuesday
Alarm B	
Hours	12
Minutes	0
Seconds	0
Sub Seconds	0
Alarm Date Week Day Sel	Weekday
Alarm Week Day	Tuesday

Final assembled system

A small set was built for mount and connect the components and simulate the real systems conditions. Figure 15 shows the final project mounting with all the components

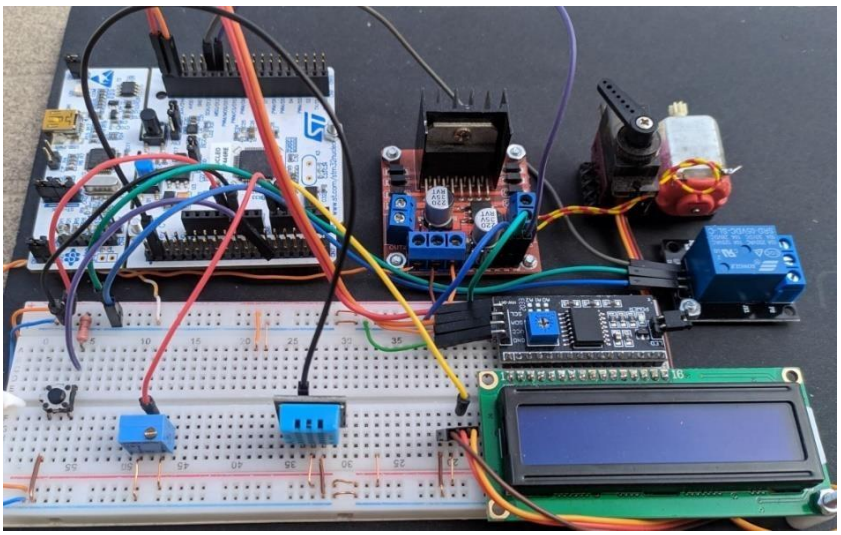


Fig15: Physical implementation of connections between components

Simulations shows that the implemented systems works as planned, sensors reading, peripheral communications, control algorithms, actuators operation.

Conclusions

After a deep study of fundamentals, work principles, operation modes and facilities of the STM32F446 MCU, it was successfully implemented a system for automate the control of environmental variables and general operation of a greenhouse for cultivate tomato, lettuce and cucumber. The system is scalable and can be extended for other types of crops. For these purposes, were used Board resources such as 4 timers, Real-time clock, Analog/Digital converters, PWM generators, interrupts, I2C communication, and standard PinOut operation. Several sensors and actuator were interfaced. Two PID controller were implemented, who effectively maintain the variables in desired operation point. With these, the initial goal of the project was successfully meet.

Main code and functions

Some considerations

In the main source file, the variables and structures that will be used are initialized, as well as the functions are declared and implemented.

Irrigation system

-RTC triggers alarm interrupt at 10 am every day
-Attention to interruption is to set irrigation pump (relay)
-After certain time or after a soil humidity overcome certain threshold (both depends on the type of crop), an interruption is triggered that resets the irrigate pump.
-Pressing the external button (PA.15) activates another interruption that increases the counter (crop type), changes the irrigation time, and shows info on LCD.

Crop type selection

Button that alternates between the 3 types of crops (tomato, lettuce, cabbage) every time it's pressed, attention to interruption increases the counter, changes the time that the irrigation pump will be stopped (AlarmB time) and the setpoint of temperature and humidity according to next table.

	Tomato	Lettuce	Cucumber
Air Humidity (%)	70	75	80
Temperature (°C)	25	18	23
Irrigation time	02:00 h	01:00 h	01:20 h
Soil humidity (%)	65	73	86

In main loop

Timer 13 controls the measurement and control. 5 times per second it triggers an interruption (at the end of count T=0.2 sec) which gives the order to obtain the DHT11 measurement. PID of both control loops are executed and its output are sent to the respective PWM of the servo and DC motor.

When irrigation enable, measure soil humidity until reach a predefined value, then Turn OFF the pump

If mode is 1, it means that the crop change button interruption has been activated, the related information is displayed for 2 seconds.

If mode is 0, normal operation, temperature and humidity measurement are updated and displayed on LCD. PIDs are executed. PID outputs must be scaled to enter PWM
- Duty Cycle to Timer 2 feeds the PWM servomotor (Min 60 -- Max 250)
-Duty Cycle to Timer 3 feeds the PWM for DC motor. (Min 0 -- Max 100)
At the end of the loop the timer 13 flag is reset, the rest of the time nothing is done

---

First the PIDs and other important variables are initialized:

```
uint8_t hum_SP_array[3]={70,75,80};            //setpoints array
uint8_t temp_SP_array[3]={25,18,23};
uint8_t soil_humid_array[3]={65,73,86};
struct PID {float kp, ki, kd, iTerm, LastInput, T;    //struct with PID parameters
            int outMin, outMax, iMin, iMax;};
struct PID PIDt = {.kp=-0.10, .ki=-0.4, .kd=-0.01, .iTerm=0, .LastInput=0, .T=0.2,
                   .outMin=-10, .outMax=100, .iMin=-10, .iMax=10}; //temp PID
struct PID PIDh = {.kp=-0.05, .ki=-0.1, .kd=-0.01, .iTerm=0, .LastInput=0, .T=0.2,
                   .outMin=-10, .outMax=100, .iMin=-10, .iMax=10}; //humidity PID
uint8_t days[7]={                 // array for update alarm day
                (uint8_t)0x01,    // Monday
                (uint8_t)0x02,    // Tuesday
                (uint8_t)0x03,    // Wednesday
                (uint8_t)0x04,    // Thursday
                (uint8_t)0x05,    // Friday
                (uint8_t)0x06,    // Saturday
                (uint8_t)0x07};   // Sunday
```

Function for ADC reading

```
/** =====ADC read=============================================*/
uint8_t adcRead(void){
      HAL_ADC_Start(&hadc1);                 // Start ADC Conversion
      HAL_ADC_PollForConversion(&hadc1, 1);  // Poll ADC1 Peripheral & TimeOut=1mSec
      adc_value = HAL_ADC_GetValue(&hadc1); // Read The ADC Conversion
      return adc_value;}
```

This are the two functions called every time AlarmA and AlarmB (from the RTC) are triggered

```
//=========== Alarm A (Start Irrigation) 10:00 AM ====================
void HAL_RTC_AlarmAEventCallback(RTC_HandleTypeDef *hrtc){
      HAL_GPIO_WritePin (GPIOA, GPIO_PIN_5, 1); // turn on the irrigation pump
      irrigation=1;                              //irrigation starts
//=========== Alarm B (Stop Irrigation)  ====================
void HAL_RTCEx_AlarmBEventCallback(RTC_HandleTypeDef *hrtc){
      RTC_DateTypeDef sDate = {0};
      HAL_GPIO_WritePin (GPIOA, GPIO_PIN_5, 0); // turn off the pump
      irrigation=0;                              //irrigation pump stops
      if (days_count==6){                        //update next weekday on RTC
            days_count=0;}
      else{days_count=days_count+1;}
      sDate.WeekDay = days[days_count];
      if (HAL_RTC_SetDate(&hrtc, &sDate, RTC_FORMAT_BCD) != HAL_OK)
      {Error_Handler();}
      }
```

Interrupt handle for button PA.15 (change crop)

```
//=========== Change crop button ====================

void EXTI15_10_IRQHandler(void){
  /* USER CODE BEGIN EXTI15_10_IRQn 0 */
extern int num_crop,mode,IrrigationTime,TimeBase[2],temp_SP_array[3],hum_SP_array[3],
           temp_SetPoint,hum_SetPoint;
  /* USER CODE END EXTI15_10_IRQn 0 */
```

```
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_15);
  /* USER CODE BEGIN EXTI15_10_IRQn 1 */
  mode=1;                                  //mode=1 during 2 seg for show the information
  if (num_crop==2)  num_crop=0;            // crop-type counter increase
  else             {num_crop=num_crop+1;}
  IrrigationTime=TimeBase[num_crop];       //update irrigation time depending of crop
  HAL_TIM_Base_Start_IT(&htim6);           //initiate timer used for show information
  temp_SetPoint=temp_SP_array[num_crop]; //update temperature
  hum_SetPoint=hum_SP_array[num_crop];     //update humidity setpoint
  Alarm_Update(hrtc, num_crop); }          //update irrigation end time (by alarm)
  /* USER CODE END EXTI15_10_IRQn 1 */

//======= Update end of irrigation time depending of crop type =============
void Alarm_Update(RTC_HandleTypeDef *hrtc, int cult){
RTC_AlarmTypeDef sAlarm={0};
if (cult==0){                          //Tomato (Irrigation 02:00 h, finish at 12:00)
  sAlarm.AlarmTime.Hours   = 0x12;
  sAlarm.AlarmTime.Minutes = 0x00;}
else if (cult==1){                      //Lettuce (Irrigation 01:00 h, finish at 11:00)
  sAlarm.AlarmTime.Hours   = 0x11;
  sAlarm.AlarmTime.Minutes = 0x00;}
else if (cult==2){                     //Cucumber (Irrigation 01:20 h, finish at 11:20)
  sAlarm.AlarmTime.Hours   = 0x11;
  sAlarm.AlarmTime.Minutes = 0x20;}
sAlarm.Alarm = RTC_ALARM_B;            // Alarm B is 2nd alarm (end of irrigation)
HAL_RTC_SetAlarm(&hrtc, &sAlarm, RTC_FORMAT_BCD);} //update alarm time in RTC
```

Interrupt handle for timer end of counting

```
//======execute when timers finish counting (only attend Timer13)=======
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
if(htim->Instance == TIM13){               //timer used for operations in main loop
NVIC_ClearPendingIRQ(TIM8_UP_TIM13_IRQn); //Clear the interruption on the line
timmer13=1;                                //activate flag
if (mode==1) {                             //if crop button is pressed
  if (contador==12){                       //count for get aprox 2 seg for show info
       contador=1;
       mode=0;}
   else contador=contador+1;}}
}
```

Functions for display information in LCD display (crop type and measurements)

```
//========= show crop in lcd ================================
void display_crop(void){
      lcd_send_cmd(0x80);                          //put cursor in 1st row
      lcd_send_string(crop[num_crop]);             //show crop type and irrigation time
      lcd_send_cmd(0xc0);                          //put cursor in 2nd row
      lcd_send_string(SPcrop[num_crop]);}          //show set points for this crop

//========= display DHT11 data in LCD  ========================
void display_variables(float temperat,int humid){
      char buf[18];                                 //temporal chars
      sprintf(buf,"Temp:   %3.1f C   ",temperat);  //concatenate temperature arrays
      lcd_send_cmd(0x80);                           //put cursor in 1st row
```

```
      lcd_send_string(buf);                          //show temperature value
      sprintf(buf,"Humidity: %d /.   ",humid);       //concatenate humidity arrays
      lcd_send_cmd(0xc0);                            //put cursor in 2nd row
      lcd_send_string(buf);}                         //show humidity value --------------
```

PID algorithm implementation

```
//========= PID function ======================================================
float PIDcontrol(struct PID *cPID, uint8_t SP, float Input){
float error=SP-Input;                                        //error term
cPID->iTerm+=(cPID->ki*cPID->T*error);
//iTermT=iTermT+(ki*T*error);  Integral Term
if(cPID->iTerm>cPID->iMax) cPID->iTerm= cPID->iMax;          //antiwindup integral term
else if(cPID->iTerm<cPID->iMin) cPID->iTerm=cPID->iMin;
float deriv=cPID->kd*(Input-cPID->LastInput)/(cPID->T);      //derivative term
float Output = cPID->kp*error+cPID->iTerm-deriv;             //controller output
if(Output>cPID->outMax)    Output = cPID->outMax;            //antiwindup output
else if(Output < cPID->outMin) Output =cPID->outMin;
cPID->LastInput = Input;                                     //update last value
return Output;}
```

Some timers are then initialized before running the main loop.

```
HAL_TIM_Base_Start(&htim1);                //Start Timmer 1 (1 uSec timebase for DTH11)
HAL_TIM_Base_Start_IT(&htim13);            //Start Timmer 13 (Timebase of 0.2 sec)
lcd_init();                                //Init LCD
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2); //Start Timmer for PWM for servo
HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);  //Start Timmer for PWM for DC motor
```

There are some functions involved in get data from DHT11

```
//=========functions related to DHT11 working ===========
void GetMeasurementsDHT11 (float *temperat,uint8_t *humid){
      if(DHT11_Start()){              //get DHT11 measurements
            *humid = DHT11_Read();    // Relative humidity integral
            RHD = DHT11_Read();       // Relative humidity decimal
            TCI = DHT11_Read();       // Celsius integral
            TCD = DHT11_Read();       // Celsius decimal
            SUM = DHT11_Read();       // Check sum
            *temperat = (float)TCI + (float)(TCD/10.0);}} // temperature in celsius

//-------- 1 uSec delay used for internal operation ----------
void microDelay (uint16_t delay){
  __HAL_TIM_SET_COUNTER(&htim1, 0);
  while (__HAL_TIM_GET_COUNTER(&htim1) < delay);}

//------------ DHT11 start measurement ---------------------------------------
uint8_t DHT11_Start (void){
  uint8_t Response = 0;
  GPIO_InitTypeDef GPIO_InitStructPrivate = {0};
  GPIO_InitStructPrivate.Pin = DHT11_PIN;
  GPIO_InitStructPrivate.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStructPrivate.Speed = GPIO_SPEED_FREQ_LOW;
  GPIO_InitStructPrivate.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(DHT11_PORT, &GPIO_InitStructPrivate); // set the pin as output
  HAL_GPIO_WritePin (DHT11_PORT, DHT11_PIN, 0);       // pull the pin low
  HAL_Delay(20);                                      // wait for 20ms
  HAL_GPIO_WritePin (DHT11_PORT, DHT11_PIN, 1);       // pull the pin high
  microDelay (30);                                    // wait for 30us
```

```
  GPIO_InitStructPrivate.Mode = GPIO_MODE_INPUT;
  GPIO_InitStructPrivate.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(DHT11_PORT, &GPIO_InitStructPrivate); // set the pin as input
  microDelay (40);
  if (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)))
  {
    microDelay (80);
    if ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN))) Response = 1;
  }
  pMillis = HAL_GetTick();
  cMillis = HAL_GetTick();
  while ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)) && pMillis + 2 > cMillis)
  {
    cMillis = HAL_GetTick();
  }
  return Response;
}

//-------------- DHT11 reading operation I2C ----------------------------------------------
uint8_t DHT11_Read (void){
  uint8_t a,b;
  for (a=0;a<8;a++){
    pMillis = HAL_GetTick();
    cMillis = HAL_GetTick();
    while (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)) && pMillis + 2 > cMillis)
    {                                                  // wait for the pin to go high
      cMillis = HAL_GetTick();    }
    microDelay (40);                                     // wait for 40 us
    if (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)))   // if the pin is low
      b&= ~(1<<(7-a));
    else
      b|= (1<<(7-a));
    pMillis = HAL_GetTick();
    cMillis = HAL_GetTick();
    while ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)) && pMillis + 2 > cMillis)
    {                                             // wait for the pin to go low
      cMillis = HAL_GetTick();}
  }
  return b;
}
```

---

main loop.

```
while (1)
  {
if(timmer13){                                 //Check timer13 flag ( 5 times per second)
//============== Soil humidity checking ==============================
if (soil_humid>soil_humid_array[num_crop]){            //if soil_humidity>setpoint
HAL_RTCEx_AlarmBEventCallback(&hrtc);}                 //turn off irrigation pump
//============== Check if crop button is pressed ==============================
if (mode==1){                                          //when crop button is pressed
   display_crop();}                                    //show crop in LCD
//============== Normal operation ==============================
else{
   display_variables(temperat,humid); }              //display measurements in LCD
//==============Read data from DHT11 ==============================
GetMeasurementsDHT11(&temperat, &humid);             //reading sensor DTH11
```

```
//============== Temperature PID ================================
OutT=PIDcontrol(&PIDt,temp_SetPoint,temperat);      //temperature PID control
htim2.Instance->CCR2=(OutT*9.5+155);                //PWM output to servo
//============== Humidity PID ===================================
OutH=PIDcontrol(&PIDh,hum_SetPoint,humid);         //humidity PID control
htim3.Instance->CCR1=(OutH*5+50);                  //PWM output to DC motor
timmer13=0;}                                       //reset timmer13 flag
}
  /* USER CODE END 3 */
}
```