GARDENING SYSTEM

Automatic system to keep plants always cool.

Version 1.0

Isaac Garcia Peveri

Contents

[Components used 3](#_Toc111390728)

[Assembling the circuit 4](#_Toc111390729)

[1. Understanding Arduino 4](#_Toc111390730)

[2. Connections (Full Schema) 5](#_Toc111390731)

[3. Connections (Simplified schema) 6](#_Toc111390732)

[Software part 7](#_Toc111390733)

[1. Arduino IDE 7](#_Toc111390734)

[2. Arduino Sketch 8](#_Toc111390735)

[3. C# .NET Project 10](#_Toc111390736)

# Components used

Following the list of components needed to complete this project:

* Arduino Uno board or equivalent clone. A smaller board like nano can be used too.

|  |  |
| --- | --- |
|  |  |

* Mini PC or Raspberry PI capable to run Linux or Windows with .NET

I used a Raspberry Pi 4 B with 8GB of Ram



* Some electronic components

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Buzzer (not necessary) | Two Leds diodes (not necessary), in different colors (one red and one green). | 1 DC 5V Relay | 1 DC 12V Adapter (500 mA / 1A max) | 12V Mini Water pump | Cables | Two 220 Ohms resistors  (not necessary) |

* Breadboard (not necessary but useful to add more complexity)

A picture containing text

Description automatically generated All components listed can be order from Amazon, OLX and allegro. There are also complete kits providing all the necessary.

# Assembling the circuit

## Understanding Arduino

The project is very simple and to work, no need to connect everything and/or having big electronic knowledges or competences.

First thing, the most important thing is to know how the microcontroller works. The board have many pins to connect different devices, let’s see most important of them which ones will be useful also for future references:

A close-up of a computer chip

Description automatically generated with medium confidence

* GND: As per name, this pin is simply the ground connection. There are 3 ground pins on the board.
* 13: This is a 5V normally open port, usually is used to perform some test or connect a LED diode. It’s a digital I/O.
* From 12 to 1: these are also all digital I/O N/O pins, providing +5V each one.

There are also 3 important pins in the POWER Section:

* 3.3 V: This pin (output) N/C simply provides 3.3 V to connect a device. It’s Normally Closed.
* 5 V: Like the other, this pin simply provides a tension of 5V. Normally Closed.
* Vin: This is a particular pin: this pin provides the same voltage used to power the board. Arduino can be powered by the USB SERIAL port or via DC Adapter up to 12V Max.

And the analog Ports:

* These pins are used to read / write data from analog sensors, but can be used also as Digital. The operation mode is decided via software.

## Connections (Full Schema)

Supposing to connect all the devices listed before, this is the connection schema I made:

Diagram

Description automatically generated

Let’s explain it:

- On pin 13 is connected the green led, placed on the breadboard, and powered by 5V DC. Because leds cannot operate at this voltage rate, a resistor is placed between the led and the ground.

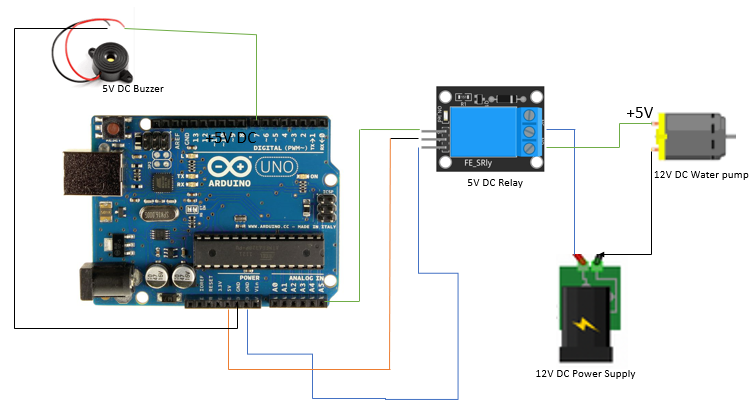
- On pin 7 is connected the buzzer, this pin provides +5V DC when is turned ON via software. The ground connection can be placed on GND arduino pins or directly to the ground connection we made on the breadboard.

- The relay connection is more complex: we are taking in this schema the +5V power from the breadboard +5V connection we made before by connecting +5V pin. Obviously, the positive pin on the relay can be connected directly to +5V pin on the Arduino board. Ground is connected to the Arduino GND pin, and the “relay switch signal” is connected to port A5 (we can use any port anyway).

To connect the water pump, connect its positive pin to the Normally Open connection to the relay as shown, the +12V DC must be connected to the “COMM” and the negative of the water pump must be connected to the GND of the power supply or the battery.

## Connections (Simplified schema)

Here I show a simplified schema, without the unnecessary components. I kept the buzzer even if it’s not needed (it’s quite horrible to listen to it).



# Software part

## Arduino IDE

We need to write a simple program into the controller to operate it from external. Every program inside the controller is called Sketch. The language used to write these sketches is a derived language from C/C++.

The sketch I made is a simple loop that checks what is coming from the Serial Port, and activates the corresponding pins on the controller.

To download and install Arduino IDE go to <https://www.arduino.cc/en/software> and download the version for your system and then connect the controller to the computer via the USB cable.

After the installation, usually an original Arduino controller on Windows is connected by default to the port COM3, on Linux system is /dev/ttyACM0 or /dev/ttyUSB0.

*Note: The USB port on the Arduino controller is also used to power it with +5V DC coming from your computer. If you power the Arduino controller via external power supply, the USB port is used only for Serial Communication and this +5V is automatically disabled, so you don’t have to worry.*

## Arduino Sketch

This is the sketch provided with the project. I took ideas from many projects taken from the internet to build it.

Let’s explain how it is working:

First of all, I define variables for the pins I need, and something to make the sound from the buzzer:

#define PIN\_BUZZER 7

#define PIN\_GREEN\_LED 13

/\*---------------------------------------

IGP Garden System sketch

Author: Isaac Garcia Peveri

Date Written: 06/07/2022

---------------------------------------

\*/

const int PIN\_RELAY\_1 = A5;

const int LowFrequencyByte = 80;

const int HiFrequencyByte = 100;

This variable is used to read data from the Serial Port (USB)

String val;

Initializing the controller and the pins. Arduino operates as Serial Communicator 9600, 8, N, 1

void setup()

{

Serial.begin(9600);

pinMode(PIN\_GREEN\_LED, OUTPUT);

pinMode(PIN\_BUZZER, OUTPUT);

pinMode(PIN\_RELAY\_1, OUTPUT);

}

These two methods are used to make the buzzer sound like a siren, but is quite horrible to listen, so I will change this into the future with small “clicks”. If you don’t want buzzer, you can remove all these methods or simply don’t connect it to the controller.

void First\_Frequency() {

for(int idx = 0; idx < LowFrequencyByte; idx++)

{

digitalWrite(PIN\_BUZZER,HIGH);

delay(10);

digitalWrite(PIN\_BUZZER,LOW);

delay(10);

}

}

void Second\_Frequency() {

for(int idx = 0; idx < HiFrequencyByte; idx++)

{

digitalWrite(PIN\_BUZZER, HIGH);

delay(5);

digitalWrite(PIN\_BUZZER, LOW);

delay(5);

}

}

Two new methods to start and stop the sound… If you don’t want buzzer, you can remove all these methods or simply don’t connect it to the controller.

void MakeSound()

{

First\_Frequency();

Second\_Frequency();

}

void StopSound()

{

noTone(PIN\_BUZZER);

}

Here the main loop, checking continuously what is coming from the serial communication, and raising or lowering the respective pins connected to the relay (and the non necessary components in this case).

void loop()

{

if (Serial.available())

{

val = Serial.readString();

}

Check\_Relay\_1();

}

void Check\_Relay\_1()

{

if (val == "ON\_1") //Cycle Start

{

MakeSound();

digitalWrite(PIN\_GREEN\_LED, HIGH);

digitalWrite(PIN\_RELAY\_1, HIGH);

}

else

{

if (val == "OFF\_1") //Cycle stop

{

StopSound();

digitalWrite(PIN\_GREEN\_LED, LOW);

digitalWrite(PIN\_RELAY\_1, LOW);

}

}

}

//END

## C# .NET Project

When I started to develop this project, I opted for a graphical WPF Windows desktop application, but as is working fine on Windows machines, there are some problems under Linux arm processors, and I don’t want to keep Windows on raspberry for two reasons:

1. It’s terribly slow
2. Writes a lot on the SD card (I don’t have SSD drive to connect to), making its lifespan shorter.

So, after successful testing on Windows systems and x86 Virtual Machine Linux systems, unfortunately was not the same on Arm32/64: wine on Arm processors doesn’t support Windows .NET applications. I tried also with official RaspiOS, but unfortunately there is no escape.

So, I decided to move to a Console Application to make it cross platform for Windows and Linux.

I know that console applications are not so attractive like graphical ones, but the TerminalGui free library (nuget package) can bring more attractive look to your console applications, and in this case it worths the price!

The whole project is well commented. All the application consists in one Main Program written in C#. The program shows also how to draw your screens and add Timers to the main loop, and sending requests to the controller by writing on the Serial Port.