Automatic Gardening System

Kristmund Ryggstein and Hergeir Winther Lognberg
University of the Faroe Islands

December 18, 2018

Abstract

We were asked to do an it project using an Arduino Uno. We chose to to do an automated gardening system with it, because that is relevant for some work projects that are currently in progress on Sandoy, where we are from. Our ardunio project entails an automated system that waters, measures moisture, heat and light exposure. That is done by customizing an Arduino Uno which has a moisture-, heat and light sensor that does all our measurements. All the the data is then collected by the Arduino, and then sent to a Web Service via. a esp-05 wifi module. The Web Service receives the data, and presents it to the user. The user is registered with the Web Service, and can track his plant data through his registered user account, where he also can customize water input for the plant/s.

Contents

Foreword

The foreword often consists of the author's personal opinion, the history of how the work emerged, how the work is structured, how the reader can choose to read it etc. It might also contain dedications, acknowledgements and similar things.

Acknowledgement

We want to thank The University of The Faroe Islands for providing us with the materials provided to us from the beginning, and to thank them for providing us with the funds to procure new materials. We also want to thank Benadikt Joensen for guiding and teaching us on the more electrical side of things in our project.

Chapter 1

Introduction

The purpose of this report is to document our project and the process of developing it.

We have chosen to work with automatic a gardening system because it has some tasks, what we believe, are tedious and easy to forget. There has been an increasing interest on Sandoy in trying to grow more vegetables on the islands, and to sell them. That grown interest in thanks to initiatives such as "Veltan" and "Eplafestivalurin". Trying to be self-sufficient with regards to food produce is, according us, a good idea, and that's were we think that our modest electronic gardening project fits well with that philosophy, and will be an interest for those, who want to try to be more self-sufficient.

Chapter 2

Setup

The Arduino Uno, TMP36GZ heat sensor, resistors and the cables which we use in this project, was provided by the one responsible by the course. We used at the start a FC28 moisture sensor, but it generally uses DC current which means that it corrodes quickly because of electrolysis in the soil and alters the soil composition which could potentially damage the plant. That would also eventually lead to tainted readings. We changed it to use an AC pulse that can be seen in figure, but we decided to opt for Capacitive Soil Moisture Sensor V1.2 that is Corrosion Resistant. The FC28 sensor can be seen in A.1 and the Capacitive sensor can be seen in figure A.2.

2.0.1 Arduino

Arduino is an open-source electronics platforms which is based on user-friendly software and hardware components.[1] It can read outputs from these components e.g. readings from sensors, and it can input these sensors to that start.

We worked on an Arduino Uno. It is a microcontroller board. It has 14 digital i/o pins. they can be located on the board right next to the text *DIGITAL*. Six of those pins can be used as Power Width Modulation, shortened as PWM, outputs. Those pins are the ones who have a next to their ascribed numbers. It has six analog pins. It then has five different kinds of power pins. We only use the GND and the 5V and 3.3V pins. The 5V and 3.3 pins output a 5V and 3.3V from the regulator on the board which can be used to power different hardware components. The 3.3V pins has a maximum current draw of 50mA. GND stands for ground, and serves as the common return path for current from the different hardware components in

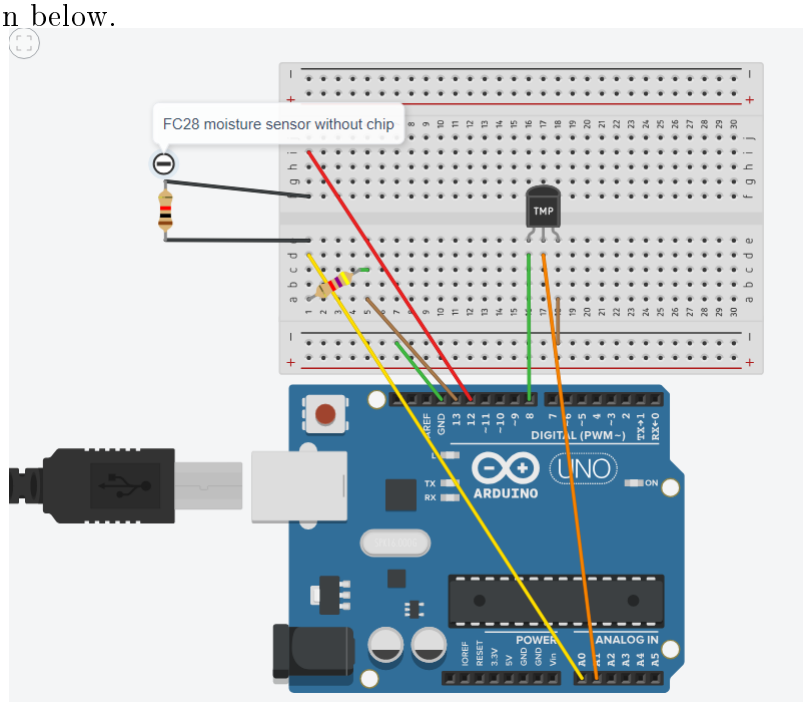
[1] https://www.arduino.cc/en/Guide/Introduction

our Arduino system.

2.1 Moisture Sensors

2.1.1 FC28

Our intention was to use the Arduino to induce an alternating current through the sensor. To simplify things, we circumvented the circuit of FC-28 such that we connected directly to the pins on the sensor. To alternate the current, we first emit 5V from pin 12 and set pin 13 to low (0V) for 1 ms then we take a reading through the analog pin A0. Then we flip the pin 12 and 13 (5V from pin 13) for the same amount of time. This should somewhat remedy the problems of electrolysis by shifting the ions back and forth between the moisture sensor pins, instead of just letting them congest at one pin. The chip had to be removed so that it could use AC. The setup can be seen below.



The resistor in the circuit helps increase the amount of current entering the analog sensor. Without it about half the current would go in the pin 13 when the measurement occurs. We are therefor able to steer the range of current entering the analog input. The left left leg is connected to 8 digital pin, the middle leg is connected to A1 pin, and the third leg is connected to -18 on the breadboard which is then connected to the ground pin on the Arduino.

2.1.2 Capacitive Soil Moisture Sensor v1.2

We looked at reviews, and this one seemed to be the most corrosion resistant of the ones we looked at. It is made by corrosion resistant material, so it has a higher lifetime than the other sensor. It measures moisture, or rather the ions in the soil, by reading the capacitance between two plates in the sensor. It has lesser range than the other one, as it measures air values as 555 and completely submerges as 280. We thought the minimized range wouldn't worsen our readings, and the benefit from increased lifetime far outweighed it.

```
1 uint8_t CapacitiveMoistureSensor::readPercent(uint16_t
      val)
  {
3       /* 555 is the value for moisture in the air
     // 288 is the value for when the sensor is
        submerged in water
5      then sett 555 to be 0%, 288 to be 100%, and map
         the percent between them */

7   val = constrain(val,280,555);
    double valPercent = map (val, 555, 280, 0, 100);
9
    return valPercent;
11 }
```

readPercent takes the read value from the analog read, and converts it to a percent that is mapped between air- and water value. Line 7 sets a constrain on val variable which says that it can't be more than 55 and not less than 280. Line 8 creates a new variable, which is val parameter mapped as a percentage between 555 and 280 where 555 is 0% and 280 is 100%.

```
  uint8_t CapacitiveMoistureSensor::read()
2 {
    digitalWrite(dPin,HIGH);
4   delay(1000);
      uint16_t val = analogRead(aPin);
6
      return readPercent(val);
8 }
```

This method reads the output from the capacitive sensor. digitalWrite(dPin, HIGH) starts the sensor. dPin is the the digital pin in which the sensor is connected to the Arduino. delay(1000) pauses the code at one second, then it initalized the val varaiable through analogRad(aPin), where aPin is the analog pin which the sensor is connected. Then the method calls and the above readPercent(val) method, which converts the value to a percentage, and the read() method returns that percentage to where the method was called from i.e. the main Arduino program.

2.2 Temperature Sensor

2.2.1 TMP Sensor

We used the TMP36 temperature sensor, as it was supplied to use by the course administrators and that it suits our purpose. The TMP36 is fairly simple. It can measure degrees from -40°C to °125C, where it has a precision of ±2°C[2]. It works as a diode so when it changes temperature, it then its voltage changes accordingly. The sensor measures the small change and outputs an analog output based on it, so its possible to calculate the temperature according to that output. The sensor's collector pin is attached to the 8 digital pin, the gate to the analog input A1 and the third leg is We use a readCelcius() method from the Temperature Class, and it can be seen below.

```
const int8_t TemperatureSensor::readCelsius()
{
    digitalWrite(triggerPin, HIGH);
    delayMicroseconds(200);
    uint16_t reading = analogRead(readPin);
    digitalWrite(triggerPin, LOW);
    // converting that reading to voltage, for 3.3v
       arduino use 3.3
    double voltage = reading * 5.0; // * 0.0009775171;
    voltage /= 1024.0;
    return round((voltage - 0.5) * 100);
}
```

digitalWrite(triggerPin, HIGH) sends a pulse to the sensor which starts it, then delayMicroseconds() delays the program, so that the sensor waits while

[2]https://www.bc-robotics.com/tutorials/using-a-tmp36-temperature-sensor-with-arduino/

the sensor is activated. Then reading = analogRead(readPind) reads the output from the sensor. The output from it, is converted to voltage by multiplying it with five. That is then divided by 1024 to find the percentage of the analog read, because the Arduino outputs between 0 and 1023, where 0 is not voltage and 1023 is 5V. Five is then subtracted from that, and then that result is multiplied with 100 to convert it from mV to degrees. Five is subtracted from the reading because we want it so that 0V corresponds to -50°C.

2.3 Wifi Module

The ESP8266-05 wifi module was the hardest to work with. We initially bought an ESP8266-01, but that was even tougher to work with, because it had 8 pins, and the distance between each pin on the module was so small that we couldn't properly solder each part without connecting it to the next one. We then bought the 05 version which was much easier to solder. A picture is attached of the module in A.3. We don't use the antennae nor the upper GND pin. We have a problem with the module in that the Arduino UNO 3V supply has inadequate current capabilities to power it. It still works but we get occasional garbage data from the packages sent sent from the module. Garbage transmissions seem to increase the longer the module is in use. It's possible to use an alternate 3.3V supply rather than the one from the Arduino but we haven't looked into it.

Chapter 3

Webserver

Webserver

We chose to work with Microsoft's ASP.NET Core which is an open-source cross-platform framework for building cloud-based and internet-connected applications. We chose this framework because we familiar with it through the course 5022.16 Web Applications: ASP.NET with C# which we had right before this course, and that it is something both collaborators in this project like to use.

3.0.1 Login

We found out that .Net Core has something called Identity Corewhich handles the login for an ASP.NET Core application. The user can create a membership for the application by using his or she's Facebook, Twitter, Microsoft, or Google login credentials.

```
services.AddAuthentication().AddFacebook(facebookOptions =>
{
    facebookOptions.AppId =
            Configuration["Authentication:Facebook:AppId"];
    facebookOptions.AppSecret =
            Configuration["Authentication:Facebook:AppSecret"];
});
```

This code block adds Facebook authentication to our login service. facebookOptions.AppId and facebookOptions.AppSecret are plugins from chrome, which had to be created in Chrome to support facebook login for our project. We found the tutorial for it on Microsoft's support page for identity on

3.0.2 Observer Pattern

We decided to use Gang Of Four's observer pattern. We think that it fit well in our implementation of the communication between the Arduino and our Webserver. Microsoft have their own implementation of the patternpattern

In short, the pattern is about having one subject that has a list of dependencies which it notifies when there's a change.

Observer

Our provider is called Observer and implements the IObserver with ArduinoData in the template brackets. That it implements the interface means that the class implements its own version of the methods

- OnCompletet()
- OnError(Exception Error)
- OnNext(T Value)

These can be called event handlers as they get called, when those events happen. The first method indicates that the provider is finished with transferring data. Second method informs the observer that an error occurred. The third method supplies the observer with current or new data. The class has one member variable called _unsubscriber, which is an IDisposable object, and it has two properties of type ArduinoData called Data and type String which is the Observer's name. IDisposable is a C# interface that provides a mechanism for releasing unmanaged resources by its method Dispose(). ArduinoData is one we made, and it is the container for the data in which the arduino transfers to the webserver. It's properties can be seen in listings B.2

Observrable

Appendix A

Graphics of sensors and other components

The following are pictures of our sensors and other components.

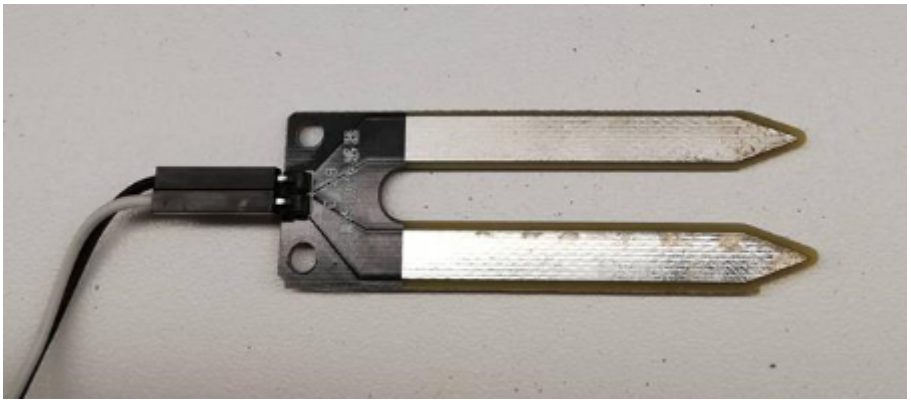


Figure A.1: Corroded fc-28 after moderate use.

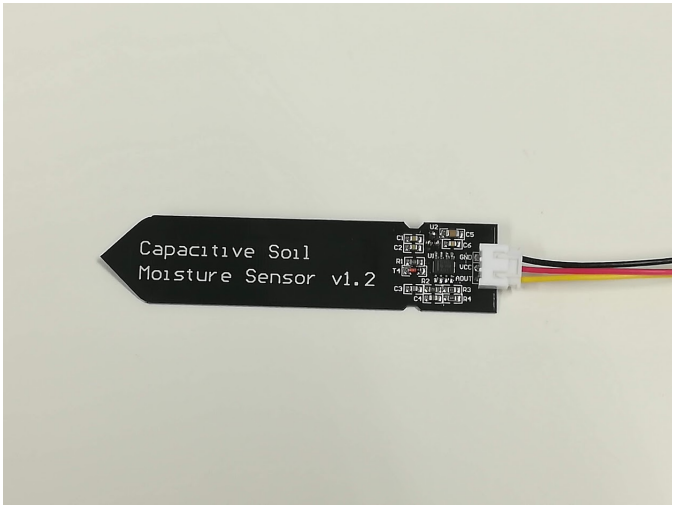


Figure A.2: Capacitive Soil Moisture Sensor v1.2

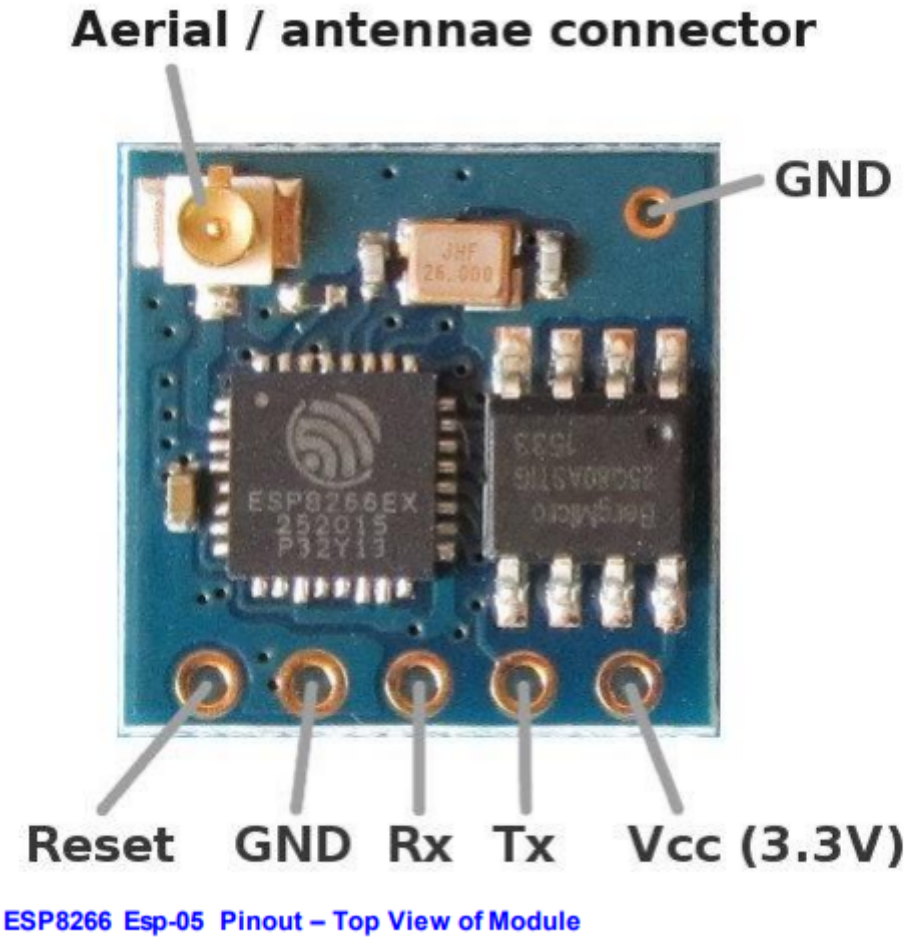


Figure A.3: Descriptive picture of ESP8266-05 module

Appendix B

Latex Listings

The code highlighting in this tex document was taken from trihedral's ArduinoLatexListing Github repository

Listing B.1: Observer Class

```
public class Observer : IObserver<ArduinoData>
{
    private IDisposable _unsubscriber;
    public EFArduinoDataRepository ArduinoDataRepository { get; set; }


    public void Subscribe(IObservable<ArduinoData> provider)
    {
        if (provider != null)
        _unsubscriber = provider.Subscribe(this);
    }

    public virtual void OnCompleted()
    {
    Console.WriteLine($"connection to observer closed.");
    this.Unsubscribe();
    }

    public virtual void OnError(Exception e)
    {
    Console.WriteLine($"Observer: Error in connection or transmission of data.");
    }

    // This runs when data from arduino is received.
    public virtual void OnNext(ArduinoData value)
    {
        ArduinoDataRepository.SaveData(value); // save the read data into the data
    }
```

```
    public virtual void Unsubscribe()
    {
        _unsubscriber.Dispose();
    }
}
```

Listing B.2: Struct of ArduinoData

```
public struct ArduinoData
{
    public int Temperature { get; set; }
    public uint Moisture { get; set; }
    public User User { get; set; }
    [Key]
    public uint PlantId { get; set; }
    public int Light { get;  set; }
    public int Water { get;  set; }
}
```

Bibliography