# Optimizing irrigation with wireless network of soil moisture sensors

Project Report

ED17-307a

Aalborg University Esbjerg
Electronics and Computer Engineering

Electronics and IT
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

**Title:**
Optimizing irrigation with wireless network of soil moisture sensors

**Theme:**
Monitoring and programming

**Project Period:**
Fall Semester 2017

**Project Group:**
ED17-B307a

**Participant(s):**
Kamil Wojciech Mikołaj
Lars -Eric Ertlmeier
Sergio Hernandez
Michał Kujawski
Devrat Singh

**Supervisor(s):**
Torben Rosenørn
Dil Muhammad Akbar Hussain

**Copies:** 1

**Page Numbers:**

**Date of Completion:**
December 20, 2017

Water scarcity is a global issue with irrigation in agriculture especially in third-world countries being a great part of excessive and inefficient use of water. We developed a system that measures the soil moisture content, processes the sensor readings, transmits it to the main microcontroller which then processes the acquired data, displays useful information on an LCD and logs the data on an SD card for further analysis. We have built a small-scale, partially working prototype to test our system. This prototype includes two sensor nodes and one main node to which they send the data. Although we cannot give a sound conclusion, we can  assume that a system built on the same idea can minimize the excessive use of irrigation water in agriculture.

## Signature

Kamil Wojciech Mikołaj

Sergio Hernandez

_____________________

_____________________

Michał Kujawski

Devrat Singh

_____________________

_____________________

Lars-Eric Ertlmeier

_____________________

# Table of contents

## List of figures

# 1 Introduction

“What is water scarcity? When an individual does not have access to safe and affordable water to satisfy his or her needs for drinking, washing or their livelihood we call that person water insecure. When a large number of people in an area are water insecure for a significant amount of time, then we can call that area water scarce” [1]. 1.2 billion people of the world’s population live in water scarce areas and by 2025, half of the world’s population will be living in water stressed areas. A region can be water scarce mainly due to ‘Supply problem’ or ‘Demand problem’. Supply problem refers to the true physical scarcity of water i.e. there is no water supply while demand problem occurs when water is available but is not managed properly. Thomas Malthus ‘s essay postulated that “ population grows exponentially, and food production grow linearly, the former will eventually outpace the other” [2]. Stated simply, human population growth would outstrip the Earth’s food producing capabilities. It is obvious that water is extremely essential for food production and the use of water in agricultural purposes accounts to 70 percent of the water consumed including crops like rice which accounts for 40% of water use. The main causes for wasteful use of water in agriculture are (a) Leaky irrigation systems, (b) cultivation of water demanding crops not suited to the environment, (c) Low-efficiency irrigation systems, (d) Overwatering. Further inattention to this wasteful use of water in agriculture would eventually lead to water crises and consequently to food scarcity.

There is a need for better systems (in terms of affordability and usability) in farming industry which ensures a more efficient use of water. The scope of our report focuses on developing a solution which – in collaboration with precise irrigation systems, like drip irrigation – may make the agricultural process more efficient in terms of cutting down on water consumption during irrigation and can help the farmer in deciding when to irrigate the crops as well as the amount of water needed.

# 2 Initiating problem

Water is one of the fundamental elements of life as we know it: no macroscopic being alive can survive without it, independently of where they take that water from. For most of the living beings, the salinity of the water is also a concern. It is not enough to have water but freshwater. And, although water in general can be considered an unlimited resource, freshwater is scarce [3] (see Figure 1) and a large part of it is used in crop irrigation. This represents one of the most challenging problems for mankind in the XXI century.

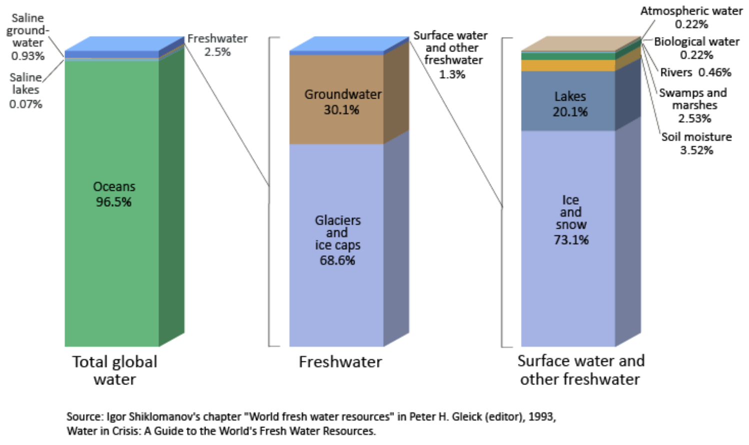


**Figure 1. World water resources. Courtesy of U.S. Geological Survey/Department of the Interior/USGS**

# 3 Problem Analysis

Water is the most vital natural resource on the planet. It is crucial for human survival and a critical input into our food, manufacturing, and energy systems. It is also responsible for sustaining ecosystem and climate. When we talk about water scarcity it is easy to get the impression that earth is running dry. But the truth is, there are land and human resources and water enough to grow food and provide drinking water for everyone. That doesn't mean, however, that the global water crisis is imaginary. Around the world there are already severe water issues. [4]

The issue is the quantity of water required for food production. According to United Nations: Department of Social and Economic Affairs, the current world population of 7.6 billion is expected to reach 9.8 billion in 2050. This growth of population will require more and more water for agriculture. Furthermore, as stated by the International Water Management Institute, agriculture accounts for about 70% of global water withdrawal and is constantly competing with industrial, domestic, and environmental uses for a scarce water supply. Inefficient ways of using water in agriculture during irrigation makes it one of the most significant contributors to ecosystem degradation and water scarcity. [5]

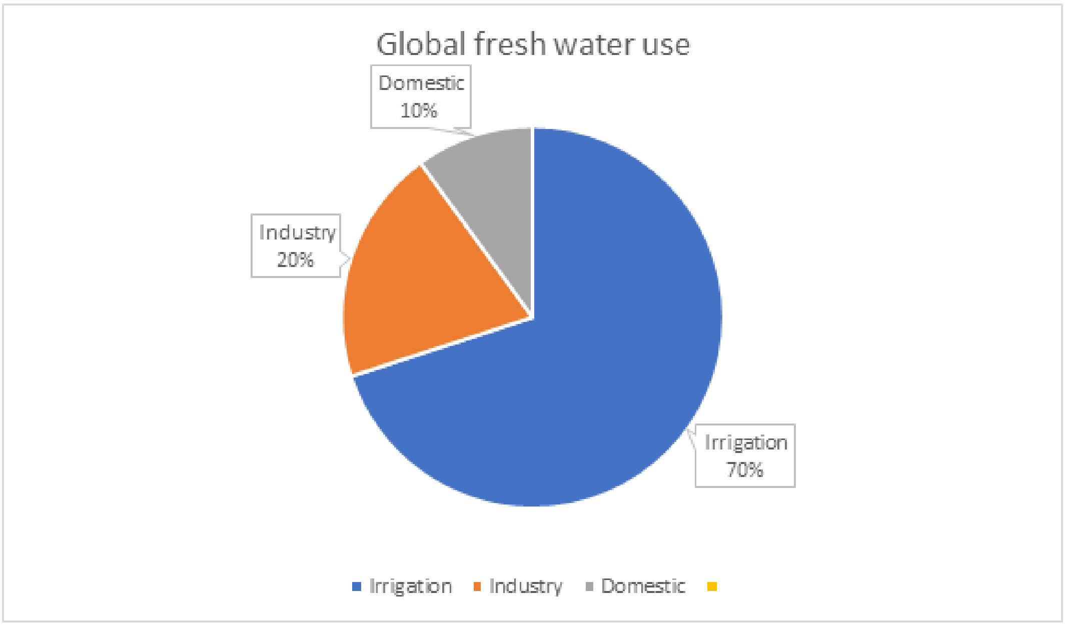


**Figure 2 Global Fresh water use**

## 3.1 Irrigation

The main reason for wastage of water in agriculture is due to the use of low efficiency irrigation systems. The most common irrigation methods are: spray irrigation, drip irrigation, and flood irrigation. Usually, the efficiency of spray irrigation is 65 percent of the water supplied but its efficiency can be increased to 90 percent by using hanging pipes. Drip irrigation is one of the most efficient types of irrigation systems. It delivers water directly to the plants or along the rows. One of the disadvantages of drip irrigation is that it requires clean water to prevent clogging in the nozzles. Even then, proper maintenance is required for cleaning the algae growth and mineral deposition. The efficiency of applied and lost water as well as meeting the crop water need, in drip irrigation, ranges from 80 to 90 percent. Flood irrigation loses about 50 percent water supplied and is the least efficient one. Out of the three types of irrigation systems, flood irrigation is the most common as it is a cheaper form of irrigation and is inexpensive to install. Leaky irrigation systems and overwatering are also factors contributing to the wastage of water during the irrigation process. [6] [7]

The usual irrigation methods involve following a schedule for timely irrigation of crops but deciding the appropriate time gap between two consecutive irrigations and the right amount of water for irrigation can prove to be very difficult as it differs for each crop and this very often leads to overwatering or underwatering of crops. The results of underwatering can be degraded crop quality or even ruined crops and in the case of overwatering, it is waste of water. The excess water due to overwatering either runs off or seeps into the depths of soil which are inaccessible by the plant roots

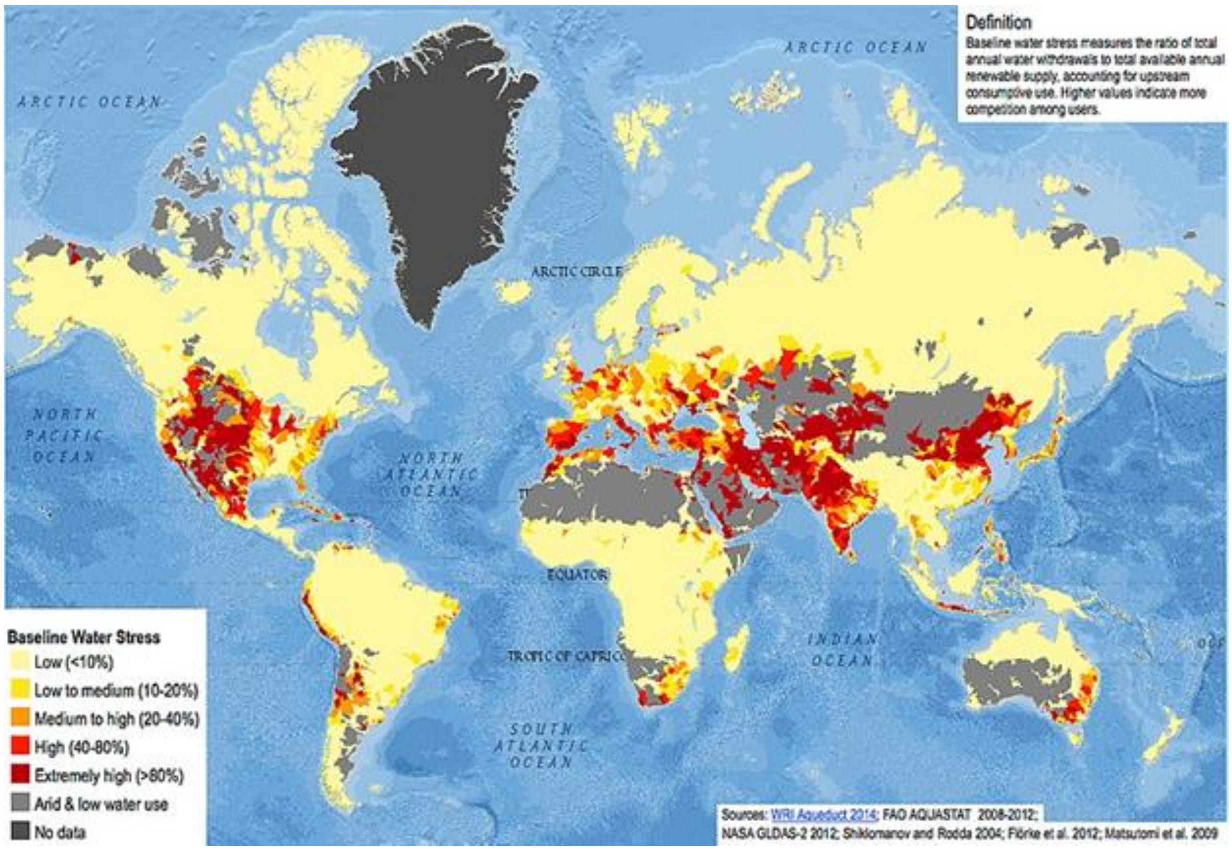

Definition
Baseline water stress measures the ratio of total annual water withdrawals to total available annual renewable supply, accounting for upstream consumptive use. Higher values indicate more competition among users.
ARCTIC OCEAN
ARCTIC OCEAN
ARCTIC CIRCLE
NORTH PACIFIC OCEAN
NORTH ATLANTIC OCEAN
EQUATOR
TROPIC OF CAPRICO
INDIAN OCEAN
SOUTH ATLANTIC OCEAN
Baseline Water Stress
Low (<10%)
Low to medium (10-20%)
Medium to high (20-40%)
High (40-80%)
Extremely high (>80%)
Arid & low water use
No data
Sources: WRI Aqueduct 2014; FAO AQUASTAT 2008-2012; NASA GLDAS-2 2012; Shiklomanov and Rodda 2004; Flörke et al. 2012; Matsutomi et al. 2009

**Figure 3 Baseline water stress**

## 3.2 Environmental, Economic, and Social effects

Many countries producing huge quantities of food like China, India, Australia, Spain, and US have reached or are very close to reaching their renewable water resource limits. For example, river basins, including important breadbaskets around the Colorado River in the United States, the Yellow River in China, the Jordan River in the Middle East, the Indus River in southern Asia and the Murray Darling River in Australia are stated as “closed” due to excess withdrawal of water, leaving just an inadequate trickle for the ecosystem.

About 1.2 billion people live in water scarce areas without enough water to satisfy demands, including water required to sustain the eco-system. Furthermore, 70% of the world's undernourished people live in rural areas and struggle to get access to enough food and water. Despite growing food, farmers constitute of world's 842 million people who suffer due to problems like poverty, disease, lack of sanitation and clean water. [4] [8]

Greater efficiency of irrigation practices in the agriculture sector would free up money for the government to provide clean water and food to the poverty-stricken regions. Thus, when water comes from improved and easily accessible sources, people spend less time and effort on collecting it, meaning they can be productive in other ways. Also, safe drinking water means people are less likely to fall ill and sustain medical costs, ensuring less expenditure on health and can remain economically productive. [9]

Increasing the efficiency of irrigation systems can save a substantial amount of water, at the same time efficiency has its limits in terms of sustainable water use because it can drive intensification which, in turn, can result in problems like poor soil health, soil erosion and over tilling. "Water Productivity" is a better measure for effective use of water as it is a combination of efficiency paired with sustainable practices like monitoring soil moisture, planting water-climate smart crops and keeping up with best irrigation systems. Thus, combining technology like soil moisture sensor with irrigation systems such as drip irrigation can provide a solution to water waste in agriculture.

## 3.3 Technology Assessment

If we do some insight into how is freshwater is used, we can see that in developing countries most of the water usage goes to agriculture. That group includes countries such as India, with a huge population and a highly agriculture-based economy, which translates in a very high volume of water every year, even higher than China (see Figure 2). Thus, we can conclude that the best way of minimizing water consumption is optimizing the irrigation systems used by the farming sector.

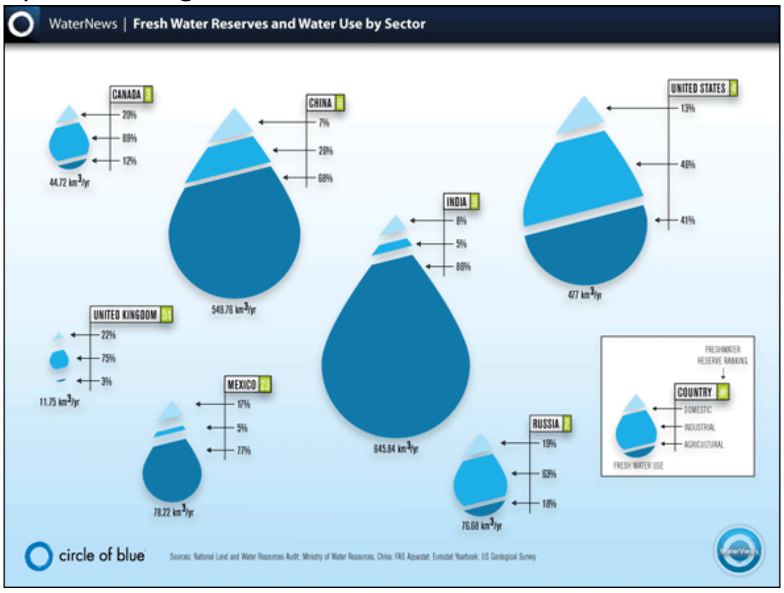


**Figure 4. Water usage in the world. Source: Natural Land and Water Resources Audit. Courtesy of http://www.circleofblue.org/2009/world/waterviews-infographics/**

Agriculture in places like India and other developing countries of its environment is still very poorly technified and highly inefficient. In regions like Punjab, where crops concentrate [10], the usage of water is clearly excessive, and the fertility of the soil decreases due to intensive farming. The weather does not help either: India has a variety of different climates, but in general we can state that rainfall is highly heterogeneous along the year, being much more abundant in spring and scarcer in autumn and winter. Besides, if we focus on India as a society, according to UNESCO, illiteracy affects around 29% of the adult population [11] and is higher in the countryside which leads the farmers to water their crops in more traditional ways that do not require any significant investment in infrastructure.

With a simple electronic watering system based on solar power, we can nearly guarantee that the system will be working 365 days a year, as its power consumption is so low that small solar panels are enough to power them in places with enough sunlight such as India. As we can monitor the moisture of the soil, we can exactly know when we should water the plants and with which amount, so we can make sure that the water wasted in the process is minimal. If it rains, the system can decrease the watering. If the weather is dry, it can increase the dosage. Keeping in mind that regular dripping irrigation can be between 15% and 30% more efficient than other irrigation systems such as surface or sprinkle irrigation, and that figure can be higher with our sensor feedback system. Once you have set a profile for amounts of water the system does not require any kind on maintenance so even illiterate users can take advantage from it. That way, we can improve people's lives making farming easier and more efficient.

## 3.4 Stakeholders analysis

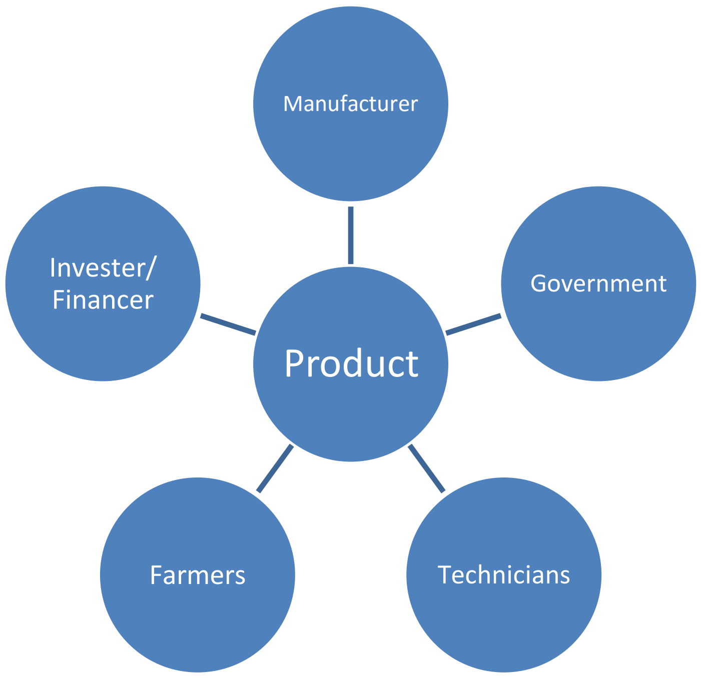


**Figure 5 Stakeholder diagram**

### Manufacturer:

Manufacturers are core stakeholders in this project as they are the ones who are going to implement the product for customers and are responsible for production, sales, and transportation of the product.

### Financer/Investor:

Insurance companies and financers provide the funds required for large-scale production of the product. They can also stop the production if there are no profits received in product sales.

### Government:

- The product should be authorized by the government for public use. The government also has the power to remove the product from the market if it causes any damage to the environment or to the people.
- Government can promote the implementation of the product in more and more farming lands by providing subsidy on it, as many farmers does not have the economic resources to install and use the product on their lands.

### Technicians:

- The initial installation of the product on cultivation fields would require a technician, who is familiar with working of the product. These technicians can either be provided by the manufacturing company or can also be outsourced.
- As the product has the ability to collect data, a technician maybe required (depending on the farmer’s capabilities) to analyse this data, collected from each crop season. The information collected can be sent back to the manufacturing company or to the engineers responsible for the design of the product which can help in improving and modifying the product according to the conditions of a region.

### Farmer:

Farmers are the primary stakeholders as they are the ones who are purchasing and implementing the product on their fields. Feedback from them can be used to improve the product. These feedbacks involve the user-friendliness of the product or any other problems faced by the farmer while using it.

# 4 Problem Statement

A system with data collecting and logging capabilities, which can continue operating during power outages in remote areas, has been developed to provide users with tools to assess the current soil moisture content, salinity and temperature as well as the history for further analysis. To provide better insight into the whole system and to show connections between modules, diagrams are included below.

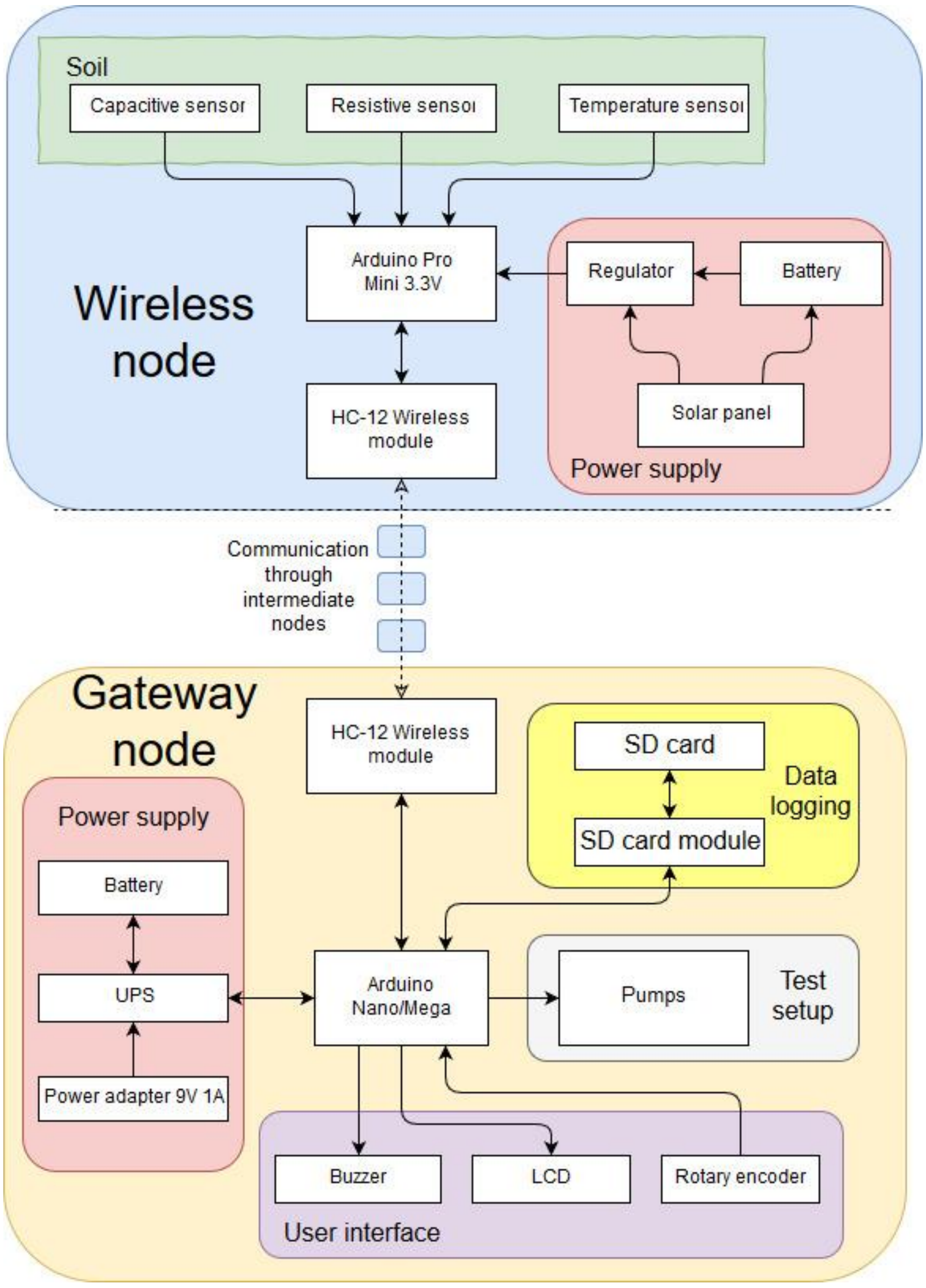


**Figure 6 System diagram**

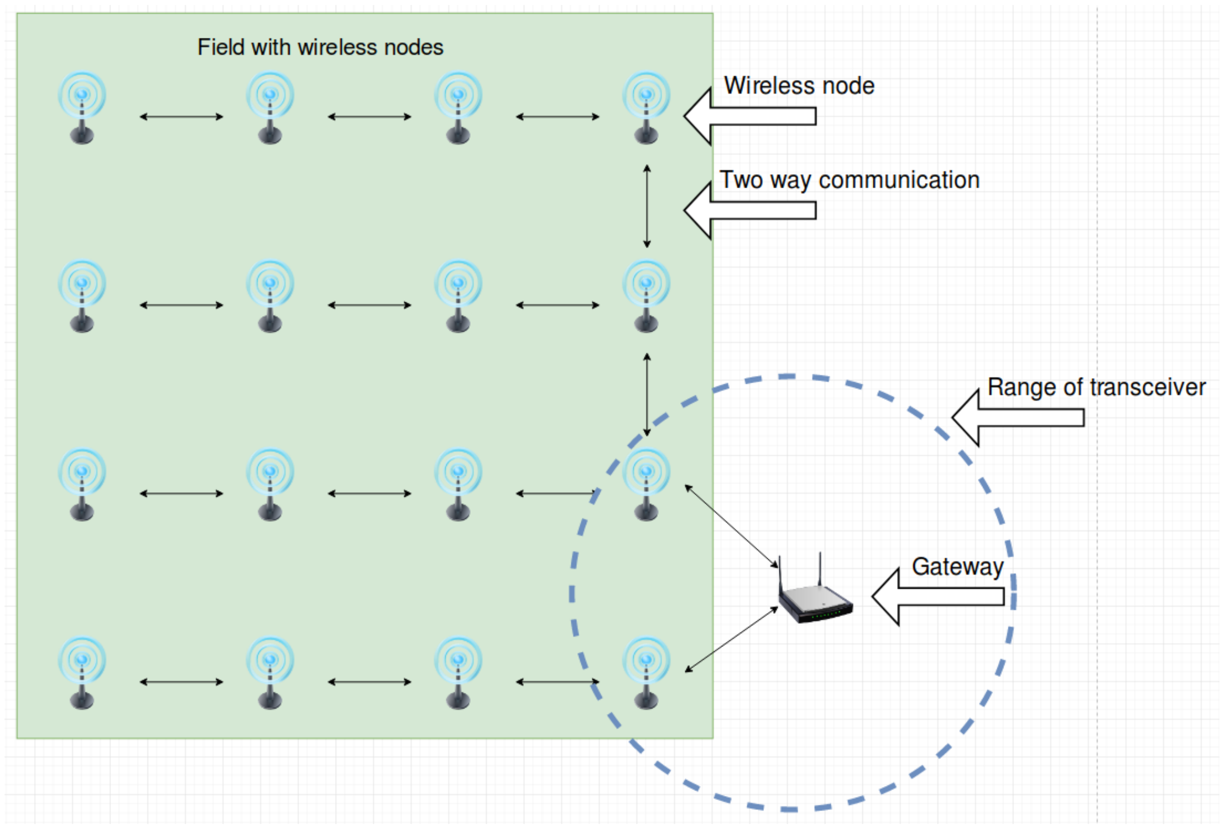


**Figure 7 Wireless sensor network**

The system consists of a gateway node which gathers, stores and displays information acquired from the rest of the system which consists of multiple wireless sensor nodes. Communication is relayed between nodes to overcome the transmitter/receiver range limitation. A detailed explanation of the whole system is included below.

# 5 Project delimitation

Due to scarce resources of our group, such as short period of the project work, small amount of experience and knowledge of group members in terms of electronics and group work, as well as environmental factors, for instance, Danish climate and conducting project at the end of autumn, following limitations have been made.

## "Real life" device:

Soil moisture monitoring cannot be implemented in the case of rice as its growing conditions involve flooding of fields from planting to harvest i.e. the soil is always moist during the whole cultivation season and the implementation of soil moisture sensor would be of no use.

Furthermore, pumps would not be included in final version of a gateway node as they would increase price of the system which might deter people from implementing solution in impoverished regions.

### Test setup:

Due to having only scarce funds and space in group room, only 2 wireless nodes were used in the project as there were only 2 pots that could monitored independently. Both of them were supposed to have direct wireless connection, as we would not be able to test communication between wireless nodes on time.

In addition to this, NIMH(nickel-metal hydride) batteries were implemented as a source of power in wireless nodes as charging them is simpler compared to other types of batteries and does not require any advanced circuit to control this process, which made it safer to test them and include them in final version of wireless nodes.

When it comes to main node, the code had to be limited on Arduino Nano because of small amount of memory of the microcontroller. Consequently, smaller SD card reading library was used to cut down on space used by program. Device would also not be able to handle bigger amount of wireless nodes unless some kind of optimization of code would free some memory.

As for testing our system, experiments with growing plants had to be done indoors due to low temperature and exposition to light that occur in late autumn.

# 6 Risk analysis

## 6.1 SLA/VRLA

Despite its robustness Sealed Lead Acid battery has its limitations and should not be charged in temperatures above 50°C [12]. This feature can be easily implemented in the system since charging voltage can be adjusted from microcontroller and temperature sensor will be used. Such a battery has the ability to supply a large amount of current due to its low internal resistance which can lead to fire risk if accidental short circuit is present so in line fuse with rating 0.5A should be implemented. It is also important that battery must not be placed in a sealed container due to the possibility of a hydrogen gas buildup which is highly flammable.

# 7 Microcontrollers

Arduino Nano (5V, 16MHz) and Pro Mini (3.3V, 8MHz) are used for this project. Both use ATmega328p chip and provide the same functionality. They differ just in clock speed and operating voltage. Arduino Pro Mini is used for wireless node because of its lower energy consumption.

During the final phase of the project it became obvious that 32kb of flash memory and 2kb of dynamic memory provided by the chip is not enough to run the main node program so for more complicated projects, a microcontroller with more memory is required. For that purpose, Arduino Mega (ATmega2560), which provides 256kb flash memory and 8kb SRAM, can be used. The comparison between Arduino Nano and Mega can be seen in the table below.

| Name | Arduino Mega | Arduino Nano |
|---|---|---|
| **Processor** | ATmega2560 | ATmega328P |
| **Operating/input voltage** | 5V/7-12V | 5V/7-12V |
| **CPU speed** | 16MHz | 16MHz |
| **Analog input pin** | 16 | 8 |
| **Digital Input/Output (PWM)** | 54/(14) | 14/(6) |
| **EEPROM - Non-volatile memory for variables** | 4kB | 1kB |
| **SRAM - Static Random-Access Memory. Used for variables** | 8kB | 2kB |
| **Flash Memory. Non-volatile memory used to store program** | 256kB | 32kB |
| **USB** | Regular | Micro |
| **UART** | 4 | 1 |

Table based on information from Arduino webpage [13]

# 8 Gateway node

The gateway node contains all modules necessary to provide data logging capabilities, user interface and notifications. It is connected to an HC-12 transmitter/receiver module to communicate with wireless nodes, an SD card module to allow data logging and to pumps so plants can be watered whenever it is required. It also has a user interface, consisting of LCD display, rotary encoder for selecting options and buzzer for notifications. It is also connected to the uninterruptible power supply to ensure that it can operate even during power outages. A temperature sensor will be implemented so that the charging voltage can be adjusted to extend battery life.

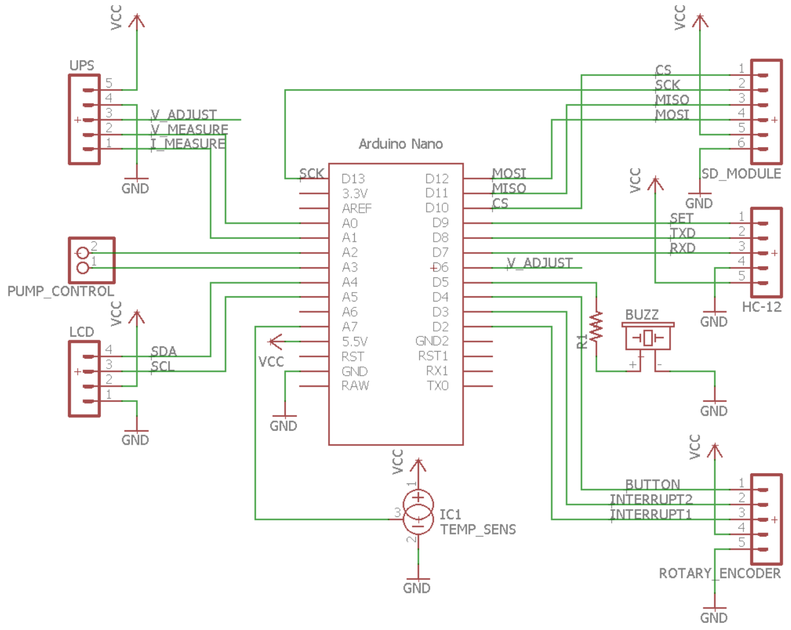


**Figure 8 Gateway node schematics**

## 8.1 Power Supply

### 8.1.1 SLA/VRLA Battery

A Sealed Lead Acid / Valve Regulated Lead Acid battery is used to provide uninterruptible power to the gateway node. This type of battery chemistry was chosen based on its robustness, low price and ability to be connected to the charging circuit indefinitely if proper float charging voltage is applied. The maximum current must also be limited to prevent damage to battery or charging circuit. Lead acid batteries consists of a positive electrode made from lead with additives like antimony or calcium to improve its properties, a negative electrode which is made from lead sponge. It also contains diluted sulfuric acid electrolyte and separator. One fully charged cell can provide 2.1V.

### 8.1.2 UPS (Uninterruptable power supply)

1. Requirements:
    a. Provide power without interruptions.
    b. Automatic switching between grid power and battery.
    c. Provide stable 5V for microcontroller.
    d. Proper battery charging voltage (6.9V for 3 cell battery SLA)
    e. Battery charging current must be limited to 0.1C (1/10 of total capacity).
    f. Battery voltage monitoring.
    g. (Optional) Voltage adjustment to improve battery life.
    h. (Optional) Charging current monitoring.

Since the Arduino onboard regulator requires at least 7V input to ensure stable 5V output, it was decided to include a more efficient chip with a minimum input of 6V so that 3 cell SLA battery and 9V power supply can be used to reduce the amount of energy that is wasted.

Based on those requirements, a first version of the circuit was developed. It consisted of two voltage regulators, an adjustable LM2941 which was intended to charge the battery and a fixed 5V output regulator (LM2940) to power the microcontroller. Unfortunately, both chips had to be replaced. Current limiting did not work with LM2941 while LM2940 was not able to output 5V when input voltage was less than 6.3V. A second version of the circuit was based on LM317 regulator and XL6009 step up/down converter module which was able to comply with requirements.

The current limiting circuit (Figure 10) was implemented to prevent damage to battery and circuit. When the voltage difference between emitter and base created by R2 reaches saturation voltage of 0.6V, the transistor starts conducting, lowering potential at regulator adjustment pin which in turn lowers output voltage of regulator and reduces current. A suitable resistor value can be calculated using the following formula:

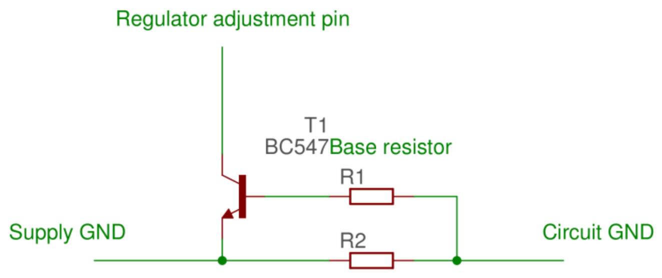


**Figure 10 Current limiting circuit**

$$Resistor\ Value = \frac{0.6V}{Current\ limit(A)}$$

After the main circuit was finished the optional voltage adjustment requirement was evaluated. It is beneficial to adjust the charging voltage based on the ambient temperature to prevent a premature failure of the battery. It should be adjusted from 2.35 to 2.22V per cell in temperature range from 0 °C to 40 °C [14]. Based on these two approaches, voltage adjustments were tested. First one was based on resistor ladder *(*Appendix 1 Resistor network) controlled by 3 digital output pins. Control was accomplished by connecting resistors L5, L6 and L7 either to supply line or ground which in turn changed reference voltage of the regulator. Eight different settings were possible, based on binary combination of 3 digital pins. Because of the high component count, difficulties to acquire precise output due to resistor tolerances and only 8 discrete output settings, a different approach was evaluated.

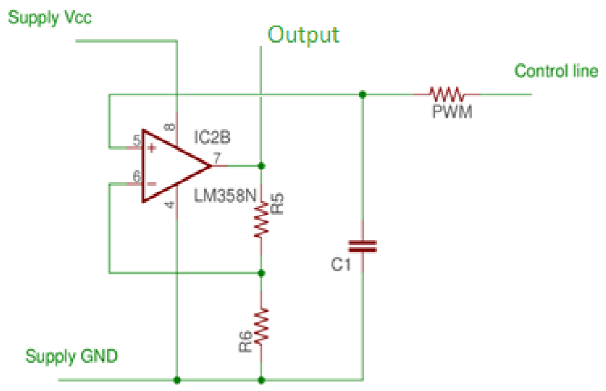


**Figure 11 PWM Digital to analog converter**

Digital to analog conversion (Figure 11) was accomplished by inputting the PWM signal from the Arduino through a low pass filter which produced 256 different analog levels based on the PWM duty cycle. Then the analog signal was amplified using an op-amp (operational amplifier) to provide a reference voltage for an adjustable voltage regulator.

The final circuit is built around an LM317 linear voltage regulator and features a microcontroller based voltage adjustment, a battery voltage sensing and a current limiter. Extra features like a connector for in line fuse and a current sensor have been added. Because the operational amplifier used for voltage control comes in dual package, the second one was used to amplify the voltage drop across R2 (in Figure 10 Current limiting circuit) which then can be measured by the microcontroller's internal analog to digital converter.

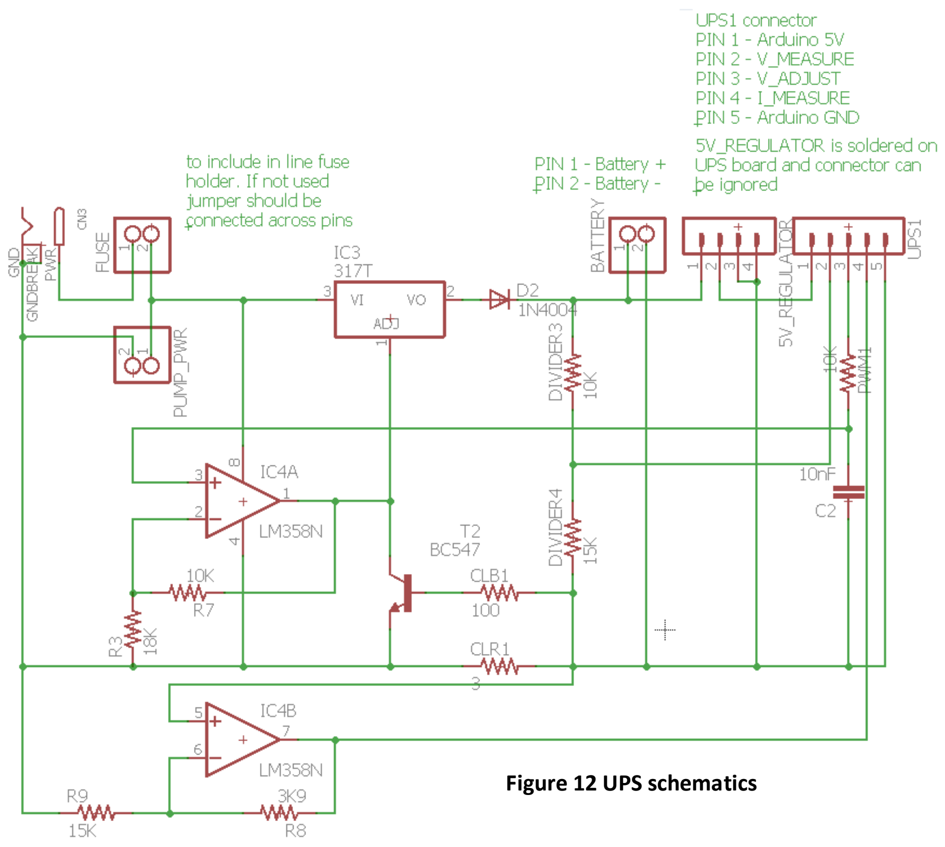


**Figure 12 UPS schematics**

## 8.2 Communication between Arduinos

Our project requires that data read from several different sensors is being transmitted to the master Arduino (previously referred to as "gateway node") which then logs this data on an SD card and displays the acquired information on an LCD. Since the transmissions are being sent through an HC-12 radio module, every Arduino in range of the module receives it. Thus, a communication protocol is needed to clarify who has to do what.

A communication protocol is a mutual agreement between sender and receiver about what the different bytes mean. [15]

In our case, it is a simple master-slave network, where the slaves (called "wireless sensor nodes") can only respond after the master has actively called them to minimize the risk of a corrupted message by only allowing one message to be transmitted at the same time and that guarantees that no device is sending data while it should receive. Due to the serial communication operating half-duplex, a device can only send or receive. Both at the same time is impossible.

A typical call from the master Arduino begins with an 8-bit system ID to identify the farm the system is operating on. Each node of one system has the same system ID. With this, it is possible to have two systems on neighboring farms with nodes of both systems in mutual range of HC-12 modules. The second byte in a message is occupied by the direct address to ensure that only one node can relay a message. This serves to prevent a message from duplicating itself and thus interfering with the original message. The third byte contains the number of bytes that are occupied by the address pathway of the node the master is calling. After the pathway, one byte is reserved for a function code to tell the slave what exactly it should execute. In a reply, the function code is being replaced by the data. The last byte in each message is a CRC8 checksum. It can detect a corrupted message in 255 of 256 occurrences. In a test I conducted, the communication protocol run for about 8 hours in an environment similar to our prototype without a single corrupted message being detected.

When a slave receives such a call and verifies that itself was meant, it either relays the call, if the final address corresponds to a different node, or it executes the function code. For our communication protocol, we utilize three function codes: one to request sensor data, one to order the slave to check for surrounding nodes and then return all addresses found. If a slave receives this function code, it calls every possible address (between 1 and 255) with the last function code that requests the address of the receiving node. After the execution has been finished, it reverses the pathway, erases the function code and the checksum and adds all the data to the end of the message. Then a new checksum is calculated and added to the end. Our communication protocol can handle any amount of data, as long as the message does not overflow the 64-byte serial buffer of the HC-12.

### 8.2.1 HC-12 radio module

The HC-12 is operating on a serial communication with a microcontroller (in our case, an Arduino). Thus, it only needs two digital pins of the Arduino for receiving and transmitting. These are connected to the RXD and TXD pins of the HC-12. By accessing the SET pin, it is possible to change the baud rate (transmission rate in bits per second). It's default operating frequency is 433.4 MHz but it is possible to change channels up to a frequency of 473.0 MHz. [16] The HC-12s in-built MCU automatically adjusts the baud rate in air to be higher than the baud rate between the Arduino and the module which prevents the in-built buffer from overflowing when sending. Unfortunately, this is not the case when receiving. A baud rate of 9,600 bps between the devices for example corresponds to a baud rate of 15,000 bps in air. A higher baud rate results in a faster transmission of data but it also increases energy consumption and it decreases range. [17] The transmission time is also limited by the timeout settings in the communication protocol. Since we need the full range of 1 byte (0 to 255) due to numerical data and an unpredictable checksum, it is not possible to work with start and stop bytes. Currently, a slave waits 80 milliseconds after receiving the first byte of a message to allow it to finish the transmission. This time buffer is well to long for our prototype but it has to work for bigger messages due to a longer pathway as well. Because of these timeouts, a direct transmission takes between 90 and 100 milliseconds (the master has a timeout of 100 milliseconds per required step). For our prototype, this means that all data can be acquired in 0.4 seconds. However, this time will rise exponentially with an increase in the number of nodes. For example, for 50 nodes and on average 4 nodes in between, it will take 40 seconds to complete.

### 8.2.2 Universal Asynchronous Receiver/Transmitter (UART)

An UART is a device that can convert parallel data to serial data to send it to another UART, which then converts it back to parallel data. Since there is no synchronizing clock signal, start and stop bits have to be added to a package. These mark the end and the beginning of a package for the receiving device. An additional requirement is the usage of the same data rate. The output pin is hold on a logical HIGH all the time until is switched to a logical LOW as the start bit. Aside from the start and stop bits a parity bit is added. It is 1 for an odd number of 1s in the data and 0 for an even number. The package contains 5 to 9 bits depending on the UART. [18] [19]Our Arduinos have an in-built UART at the digital pins 0 and 1. These pins are used for the communication with a computer or laptop but can be used for other devices as well.

## 8.3 SD Card

Our system is thought to be an assistance to the user to adjust his irrigation methods. Thus, it is useful for him to have an access to the sensor readings to analyze when he used to much water and when not enough. In our system we are utilizing a micro SD card and a Catalex MicroSD card adapter v1.0 11/01/2013 to automatically log the data in the file data_log.txt. Additionally, error notifications are logged along with the corresponding time stamp in sys_log.txt. Possible error notifications indicate corrupted messages, messages that are not meant for the master to receive, but also received sensor data at either 0 or 255 (minimum and maximum value of one byte). This could be due to extreme weather conditions (e.g. flooding or drought) but it is more likely to indicate a sensor failure. An SD card reader is connected to the Arduino through an SPI interface. The two pins for communication and the one for clock synchronizing (11, 12 and 13) are defined in the code for the SPI library and cannot be easily changed. The SS pin is variable, however the library requires the pin 10 (SS by default) to be set to output and to a logical high, if a different pin is used as SS. [20] Since these libraries are intended to be used on microcontrollers, they are limited in their functionality in order to reduce memory usage, so we are only able to write data on files in the 8-3 format. That means, the name is 8 letters or shorter and the extension is 3 letters long (like “data_log.txt”). [20] However, it is possible to copy the data into an excel file afterwards to create a graph.

### 8.3.1 Serial Peripheral Interface (SPI)

The SPI is a communication protocol designed for a network with multiple slaves and one master. Similarly to an UART it has one output pin and one input pin for the master, called MOSI (master output, slave input) and MISO (master input, slave output). Unlike an UART, is SPI a synchronous protocol and as such, it requires an internal clock. This clock synchronizes the slaves though the SCK pin. The second difference between UARTs and SPI is the number of devices in the system. A UART connects only two devices, of which none can be distinguished between master and slave, since both can send at any time. An SPI can connect a theoretically infinite, but practically limited number of slaves to one master, but one SS (slave select) pin is required for each slave. The master has to set the SS pin corresponding to the device it wants communicate with to a logical high and the other ones to a logical low, to prevent devices transmitting or receiving at the same time, since an SPI only utilizes one output pin and one input pin. [18] [19] An example of SPI in our system is the micro SD card reader, which uses the digital pins 10 to 13 with 10 as SS pin.

## 8.4 Interface

### 8.4.1 Selection of LCD

To avoid a lack of supervision over the system, an LCD (Liquid Crystal Display) was included as a part of the gateway node so as to display the status of the system that the user could check. To provide a reasonable amount of information, a display with a resolution of 20x4 symbols and 5x8 pixels for each symbol was chosen. Normally, using an LCD would significantly reduce the amount of available ports on the Arduino but the issue was solved by connecting the LCD through an PCA8574 I2C (Intern Integrated Circuit) port expander (Figure 12) that cut down the amount of required ports to 2.

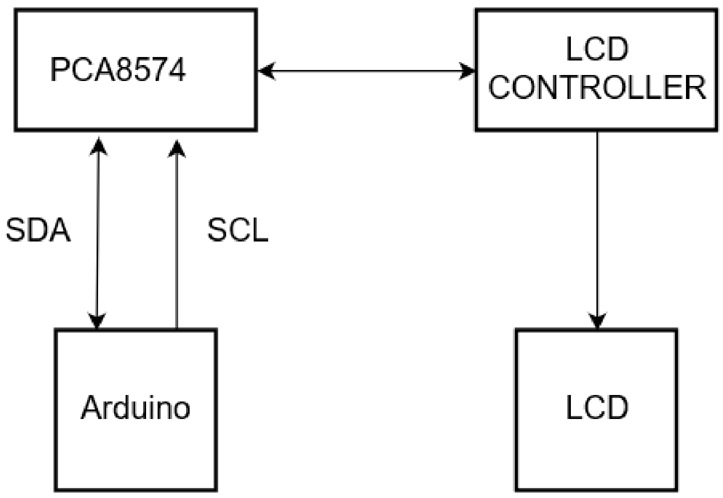


**Figure 13 Flow chart of LCD connection with Arduino**

### 8.4.2 I2C

An I2C bus is a very comfortable way of communication between electronical devices as it only requires two wires that conduct digital signals. SCL (Serial Clock) wire uses constant frequency signal, generated by the MASTER device, and can synchronize the communication by letting data to be read/transferred while its logic state is 1. If the communication has started, SDA (Serial Data) pin sends one bit of information while the state of SCL is HIGH and may change its state between LOW and HIGH when SCL is LOW. Apart from transmitting data, SDA line is used to start and stop the communication as well as to mark the end of a transferred byte (ACK condition) which is done by the SLAVE node by pulling SDA to LOW. (Figure 14) All devices in the bus have addresses and each time a communication starts, the MASTER has to send byte containing address of the chosen SLAVE device and type of transmission (read or write). [21]

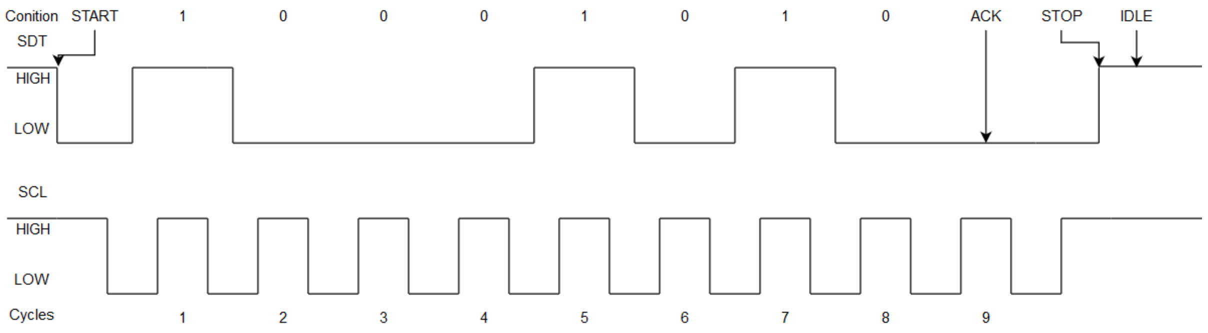


**Figure 14 Example of transmitting 1 byte, in this case 10001010. Waveforms are idealised.**

### 8.4.3 Generating text

An Arduino Liquid Crystal I2C library [22] was used to handle communication between Arduino and LCD. It provided functions to start an LCD and only basic functions to display text on screen (print and setCursor). To handle displaying various messages, additional string processing functions had to be written.

Option type was created to store four strings, that were supposed to be printed all at once to fill in all 4 rows of the LCD, as an array (interface.h). Functions such as createHeadline or createPageNum were formatting individual lines which were implemented later in template functions which filled all the strings in Option type that could be displayed on LCD to represent single option in the interface. The function initText took care of that task by printing subsequent strings in for loop. (textGenerators.cpp)

### 8.4.4 Rotary encoder with push button

In order to provide the user with the ability to navigate the interface, an input device had to be implemented in this project. A rotary encoder with a button seemed to be the most natural choice as the user can change his position in the interface or input values by rotating the encoder. As for the button, it was used for entering options in the interface.

When it comes to mechanical encoders, they can be used to measure rotation. The most important part of a simple mechanical encoder is a ring made of conductive material with gaps in it, which connects and disconnects two pins to the ground during its rotation (Figure 15). In our case, both pins are connected to the voltage supply through pullup resistors so the logic state on pins is always 1 when they are part of an opened circuit. Each time one of pins touches the rotor, it gets connected to the ground which closes the circuit and switches the pin's logic state from 1 to 0. Both pins are separated by a significant distance which causes one of them to switch its state before the second one (Figure 16)

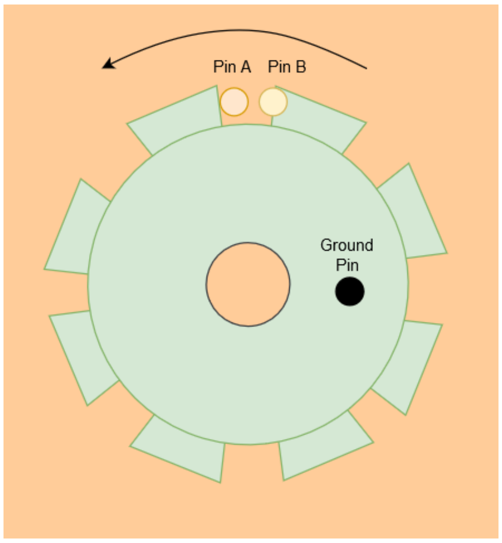


**Figure 15 Interior of mechanical encoder**

which can be used by a computer program to detect the direction of rotation. The biggest drawback of this type of encoders are noises that are produced while closing a pin's circuit which can be filtered by an RC (resistor-capacitor) circuit or adequate software.

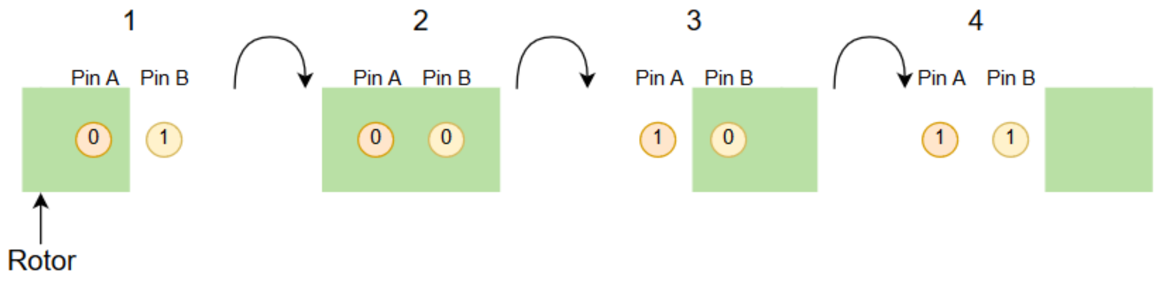


**Figure 16 Order of changing pins' logic states in clockwise rotation.**
**1. Rotor starts to spin and connects pin A to the ground. 2. The rotation is being continued and both pins are connected. 3. As rotor moves forward, gap reaches pin A first which returns to its previous state. 4. Rotor does not touch none of the pins which causes them to return to their idle state , if it continues to rotate, pin A will be the first pin to switch its logic state to 0.**

### 8.4.5 Selection of encoder

In case of this project, a KY-040 mechanical encoder with a button was chosen as it was cheaper than more accurate optical encoders. At the same time, high accuracy of encoder readings was not necessary in this solution and the frequency of rotation was expected to be low so issues such as noises or friction did not affect the accuracy on a large scale. Due to the fact that the sensitivity of the button and the encoder was easily managed by the code, it was used to ignore noises as well. Arduino digital pins 2 and 3 were used to read logic states on rotary encoder pins. Both Arduino and encoder in-built pullup resistors were used to avoid undefined logic states. As for the button, it was connected to the digital pin 4, including the Arduino pullup resistor.

### 8.4.6 Reading and debouncing input devices

One of the most important steps required to take advantage of the encoder's capabilities was writing a code that was able to analyze signals that were sent by this input device. We used interrupt pins on Arduino to handle pin's logic state detection efficiently thanks to ISR (Interrupt Service Routines). ISR are special functions that are called in parallel to code that is currently being executed so as to always detect input. They can alter output of functions that are called in the main code by changing values of global variables that are shared between an ISR and the main code. Any global variable that might be changed this way has to be declared as volatile so that the compiler is aware that it might be altered at any time. [23]

As for the code written for the purpose of this project (encoder.cpp), encoderRotDir was used as an ISR. Each time one of pins' state is being changed, it combines them as a single binary number which is added together with number that was read previously. Then, compareSum decides on direction of rotation based on that number (Figure 17) and increments global variable g_pinVal in case of clockwise rotation, and decrements it during counter-clockwise rotation.

Due to the fact that the smallest, reasonable rotation of encoder that user can perform corresponds to changing pins' state four times, global variable g_clickVal was introduced to decrease the sensitivity of the encoder. It is altered after each 4 pins' state changes in clockwise or counterclockwise rotation. This solution also serves as a debounce technique as noises that are mistakenly counted as rotation in clockwise and counterclockwise rotation cancel themselves out.

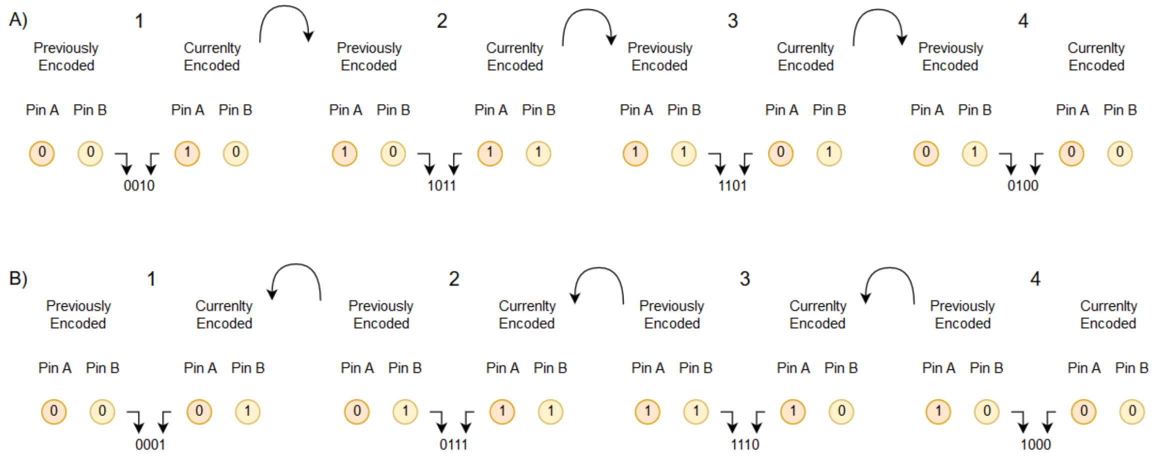


**Figure 17 All possible combinations of previously and currently encoded pins states in A) clockwise rotation B) counter-clockwise rotation**

In addition to this, function encoderValLim is called so as to set maximal and minimal values of g_clickVal. This piece of code is very important when we consider that there is given amount of options in menu and that g_clickVal may represent position it menu. [24]

Compared to reading rotary encoder, code that detects input from button is simple. (encoder.cpp) Function ButtonInput first ignores input that was read less than 50ms since previous input to avoid interpreting noises. If button was read, it returns true, otherwise false, which may be used as a condition in if-statement to call a function on demand.

### 8.4.7 Tracking position in menu and choosing output

Having written functions that were able to decode input from the rotary encoder and the button, it became possible to create an interface that could be controlled through those devices. The only necessary piece of code that was missing, were functions that could update the text that was displayed on the LCD according to received input.

Global variables g_menuIndex, g_pinVal, g_clickLimMax and g_clickLimMin (encoder.cpp) were declared to ensure that correct strings would be printed on the LCD. Function initMenus (menuInitializers.cpp) decides which "menu" function, containing right strings, should be called. Based on g_pinVal value, the "menu" function analyzes user's position in the menu and passes all the necessary strings to the appropriate text- generating function to display message on theLCD.

Furthermore, function menuOutput (menuOutputs.cpp) was created to manage outputs caused by pressing button. The function works analogically to initMenus but instead of altering text that is displayed on the screen, it changes values of global variables. MenuOutput might also modify variables that are passed to it but use of pointes is necessary as return values could represent completely different parameters due to wide range of options that can be chosen in the interface.

Most commonly used capability of menuOutput is switching menu by altering value of g_menuIndex. At the same time, g_clickLimMax and g_clickLimMin have to be adjusted to limit g_clickVal so as not to run out of possible options in newly chose menu. In addition, menuOuput can assign currently encoded g_clickVal value to a variable. For instance, this method is implemented in autoPowerMenuOuput to change time after which LCD will be automatically turned off.

Finally, function initLCD (encoder.cpp) was implemented to run entire interface code as a whole. It restarts interface in the main menu in "current time" option, turns on the LCD and calls checkEncoder and checkButton in while loop. The former function calls initMenus to update LCD and returns true when encoder is rotated. The latter function calls menuOptions to process choice after button was pushed, calls initMenu to refresh LCD and returns true. Return statement are used to determine user's presence. If both functions return false for the amount of time given by lastInput, the loop stops and LCD is being turned off. LCD can be also turned off if users chooses correct option in power menu.

# 9 Wireless Node

## 9.1 Power Supply

### 9.1.1 NiMH batteries

As we want our system to be as independent as possible and to have a minimum maintenance, the power source must be renewable, so we do not depend on external factors such as interruptions on the power line. One of the main issues of renewable energy sources such as solar power is that their input is not constant, so they are unable to run a 24h-working system ,such as ours, on their own. That made a battery system a must-have. We needed an affordable technology for the wireless nodes that could provide around 4 volts and at least 1000mAh of capacity.

**Figure 18. NiMH batteries in a charging station. Courtesy of Wikimedia Commons**

Nickel-metal hydride (NiMH) batteries were an ideal solution: cheaper than most of the rechargeable battery technologies and capable of powering the wireless nodes for months (2400mAh). Each one of the batteries supply theoretically 1.2V, but in real-world measurements we could see that 3 of them can supply 3.84V (nearly 1.3V per battery) which, after voltage losses in diodes, are ideal for the needs of a 3.3V Arduino Pro Mini. The only issue of this technology is the number of charging cycles until it becomes unusable (between 500 and 2000), less than heavier and more expensive technologies. [25]

### 9.1.2 Solar panels. The photoelectric effect

The photoelectric effect consists on the emission of electrons when certain radiation (infrared, visible light or ultraviolet) reaches a material. Both photoconductivity (variation in the conductivity of certain material depending on the incident electromagnetic radiation) and the photovoltaic effect (transformation of luminous energy into electric energy) are included in this phenomenon.

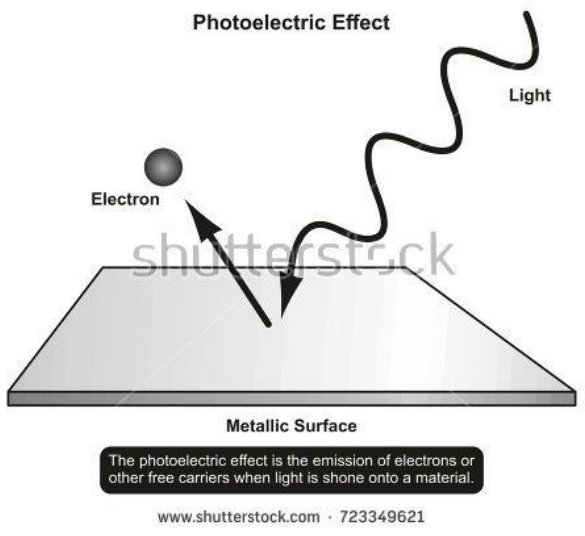

Photoelectric Effect
Light
Electron
shutterstock
Metallic Surface
The photoelectric effect is the emission of electrons or other free carriers when light is shone onto a material.
www.shutterstock.com · 723349621

**Figure 19. The photoelectric effect. Courtesy of udaix/shutterstock.com**

The principle of the photoelectric effect is tightly related to the characteristic energy of photons. That energy depends on the frequency of the light beam that the photon belongs to. When the photon hits an electron orbiting an atom of certain material, it transfers its energy to it. If that energy is higher than the energy needed to move an electron out of the material and its trajectory allows it, the electron abandons the atom. If the characteristic energy of the photon is lower than the electron binding energy, the electron stays attached to the atom. The maximum escape energy of an electron ($K_{max}$) is given by the formula:

$$K_{max} = h( f - f_0 )$$

where $h$ is the Planck constant (6.62607004 × $10^{-34}$ $m^2kg/s$), $f$ is the frequency of the light applied to the material and $f_0$ is the threshold frequency of the material.

Thus, the only factor that affects the energy of electrons is the frequency of the incoming electromagnetic radiation, regardless of the intensity of the light, that only affects the number of electrons ejected from the material.

Electrons are distributed in different energy bands within an atom. In order to understand conductivity, the crucial concepts are the valence band and the conduction band, besides Fermi's level. We can define Fermi's level as the necessary energy to get an electron out of its atom. The valence band is the closest (therefore, the furthest to the atom's core) state to Fermi's level where electrons are at 0 K temperature. On the other hand, the conduction band is the nearest range of energies to Fermi's level with no electrons on it at 0 K. If the material is a

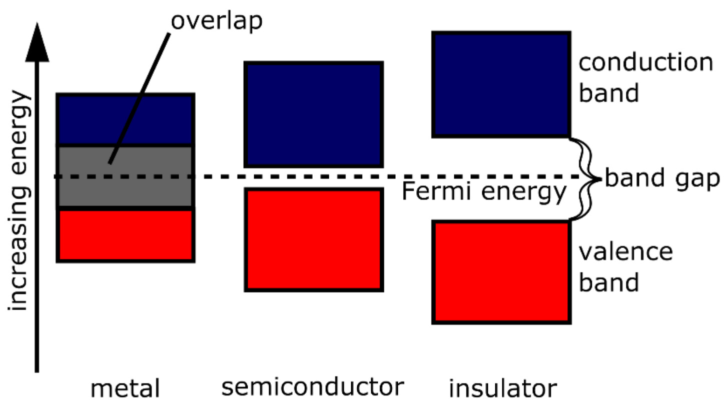


**Figure 20. Valence and conduction bands in different materials. Courtesy of commons.wikimedia.org**

semiconductor, both bands are close to each other in terms of energy, but they don't overlap. When any factor (light, temperature) increases the free energy of the electrons, it's easier than some electrons can get out of the valence band and move to the conduction band, which in "macroscopic" dimension we would call electric current.

Certain electromagnetic radiations behave as a temperature increase in semiconductors: they make them more conductive. But if we choose carefully the semiconductor used, adapting it to the type of light we want to get energy from (let's say, solar radiation) we can create an electron flux powerful enough to be used as a source of energy, a natural power supply. This is the principle behind solar panels.

**Figure 21. Solar Panels. Courtesy of www.powerdomainpictures.com/**

But in real life managing solar energy is not so easy. As power delivery is not constant throughout the day and the year, we need a device that is able to store power for its usage under demand, a battery. If we follow ideal circuit analysis, it should be enough to connect both the panel and the battery in parallel, so the battery acts as a load. When making the setup we realised that the batteries weren't charging at all, even though the short-circuit current of the panel was about 6 mA. After some experimentation we discovered that the panel's voltage must be substantially higher than the batteries to charge it (in our case, we needed a 7.5V solar panel to power a 3.8V battery pack.) Additionally, it was necessary to include a diode to avoid current going towards the panel, wasting the battery's energy at night.

Finally, we had to face the problem concerning sunlight in Denmark. As light hours are limited in this latitude and season, the power input of the panels is rather poor, and they are our only source of energy for the sensor nodes as they are not plugged to the line power. Fortunately, the system power consumption is so low (partly due to the choice of 3.3V Arduinos) that normally the overall power variation of the battery is negligible in a 2-month time span. [26] [27]

## 9.2 Sensors

### 9.2.1 Temperature sensor

Temperature readings are useful in our project for various reasons. One of them is purely biological: plants can only grow in a certain range of temperatures, and even if they can survive, they might not develop as expected. Measuring temperature allows us to know what kind of environment the plant has grown in, and gives the user a "research" tool that allows him to optimize his crop.

Another concern involving temperature is sensor readings. As our system is based on electronics, it is affected by temperature. This might lead to inaccurate readings that make the system misfunction. To avoid it, temperature sensors give us some additional information to compare readings and develop a correction algorithm to ensure that the system works in any environment.

One of the most affordable solutions available is TMP36, a simple temperature sensor based on the different voltage outputs of bipolar joint transistors (BJTs) as temperature changes. It is not especially precise (±2°C tolerance) but, as small temperature variations are negligible for both plants and electronics, it is an acceptable compromise. [28]

### 9.2.2 Capacitive moisture sensor

Capacitive moisture sensors use capacitance to distinguish soil with different amounts of water in it. Capacitance is the electrical property that defines how much charge can a body store. If we want to make a capacitor – a device able to store charge- the simplest way is putting 2 metal plates facing each other but separated by a certain distance and create a voltage differential between them. If the distance between the plates and their surface are adequate, the device will somewhat behave as a capacitor: it will store charge. But if we put a very electrically insulating material between the plates, it will store more charge, it will be a "better" capacitor. That means that the more dielectric (or insulating) the material between the plates is, the more charge we will be able to store.

Capacitive moisture sensors are a very reliable solution for accurately measuring water content in soil, but their cost is also higher. The capacitive sensors we are using are based on a square-signal oscillator connected to a probe made with 2 electrodes. The capacitance of the probe (that depends on the capacitance of the material it is surrounded by) is used as part of a low-pass filter that converts the square signal into a triangular one. Then, the triangular signal is rectified into a DC value that gives us the final reading. This way, when the material between its terminals is less conductive -dry soil- it will give us a higher voltage value (as its capacitance is higher) and when we add water the voltage output will decrease proportionally. As soil salinity influences capacitance very poorly, we can say that the readings are independent from salinity, which gives us much more precise data. Besides it is corrosion resistant, so its durability is better than other kinds of sensors. [29]

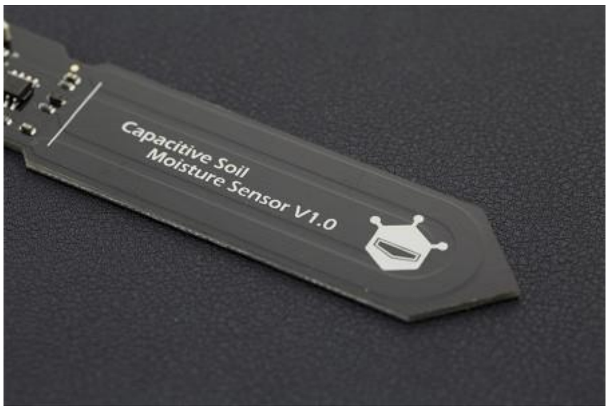


**Figure 23. Capacitive moisture sensor. Courtesy of http://minielektro.dk/**

### 9.2.3 Resistive moisture sensor/conductivity sensor

Resistive moisture sensors (or conductivity sensors as that is the property they measure) give a different value depending on how easily the material between their terminals conducts electric current. Conductivity sensors are quite simple: they have 2 electrically conductive terminals separated by a short distance with a voltage differential between them, designed in a fork-like shape. When introduced in soil - as mineralized water is a conductor itself – the conductivity of the aforementioned soil increases when water is added. One of the main problems of this solution for measuring moisture content is that soil conductivity does not depend solely on water content but also

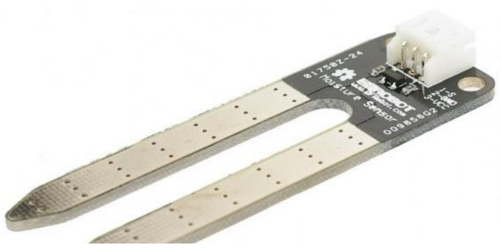


**Figure 24. Resistive moisture sensor. Courtesy of http://minielektro.dk/**

salinity and temperature. For that reason, we decided to use it as a salinity indicator. Another problem we face is corrosion, as both electrodes are exposed to external agents such as water, and their conductive properties deteriorate as the device is used. [30]

# 10Test Setup

## 10.1 Pumps

### 10.1.1 Peristaltic pumps

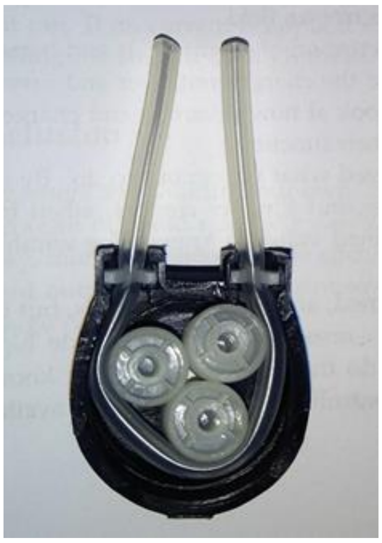


**Figure 25:Pump head and tube**

Peristaltic pumps are a type of positive displacement pump. They work on the principle of peristalsis, which is the alternative compression and relaxation of a hose or tube drawing the contents into the tube, similar to the working of our throat and intestine. The peristaltic pump can be broken down into three major components: 1. Pump tube, 2. Pump head and roller, 3. Motor. The product to be pumped is contained within the flexible tube or hose fitted in the pump casing. The pump head has a number of “rollers” which are rotated by a drive mechanism, normally an electric motor. These rollers pass along the length of the hose or tube completely compressing it to create a sealed pocket between the suction and discharge side of the pump eliminating any kind of product slip. As the rotor turns, this moves the sealing pressure along the tube or hose which forces the fluid to move away from the pump and into the discharge line. Upon decompression, the release of pressure creates a vacuum that draws the fluid into the suction side of the pump.

This combination of suction and discharge creates a powerful self-priming positive displacement action. [31] [32]

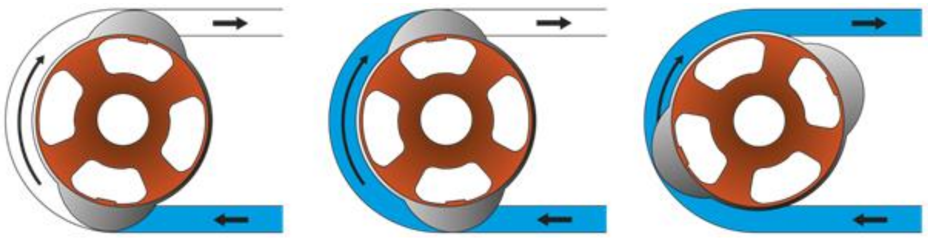


**Figure 26 Working mechanism of peristaltic pump (courtesy of www.verderliquids.com)**

**Advantages:**

- **Seal less design**

The lack of seals, valves, and glands in peristaltic pump makes it less prone to leakage as well as inexpensive to maintain.

- **Low maintenance cost**

Fewer the components, lower the maintenance cost. In the case of a peristaltic pump, the sole component that requires maintenance is the "tube" which is the only part of the pump that is in contact with the fluid and is a relatively low-cost item that can be changed in a short time.

- **Self-priming and Dry running**

Peristaltic pumps can draw the fluid into the tubing when starting dry, up to 8.8 m suction lift. Other pumps require the pump and the suction line to be filled with fluid before use which can be inconvenient and creates the potential for spills.

Concerning the project, if during irrigation the water source (15 L container) is empty then the dry running would not cost any downtime due to damage to the pump.

- **Gentle pumping action**

The gentle and low shear pumping action of the peristaltic pump makes it ideal for the project when it comes to watering the plants as it would prevent soil dispersion at the end of discharge line in the pot.

- **Reversible**

Peristaltic pumps are reversible as flow direction can be changed by switching the positive and negative connection and this can be used to empty lines and clear blockages in the tubing.

- **Non-Siphoning**

A peristaltic pump is non-siphoning i.e. they prevent backflow into the system. This promotes accuracy during dispensing.

- **Accurate Dosing**

Peristaltic pumps have a repeatability of $\underline{+}1$ % and metering capabilities of $\underline{+}5$ % and as the main concern of the project involves the efficient use of water this property comes into use when delivering the exact amount of water required, depending on the moisture level of the soil.

### 10.1.2 Pump control circuit

An Arduino microcontroller is not able to provide enough power to run pumps directly which requires over 200mA at 9v while the chip is only able to provide 40mA at 5V. To address this issue, an external transistor is required to switch the pumps. There are also fly back/snubber (D4 and D5) diodes which provide a discharge patch for a sudden voltage spike induced by a collapsing magnetic field when an inductive (electric motor) load is switched off which could damage the transistor.

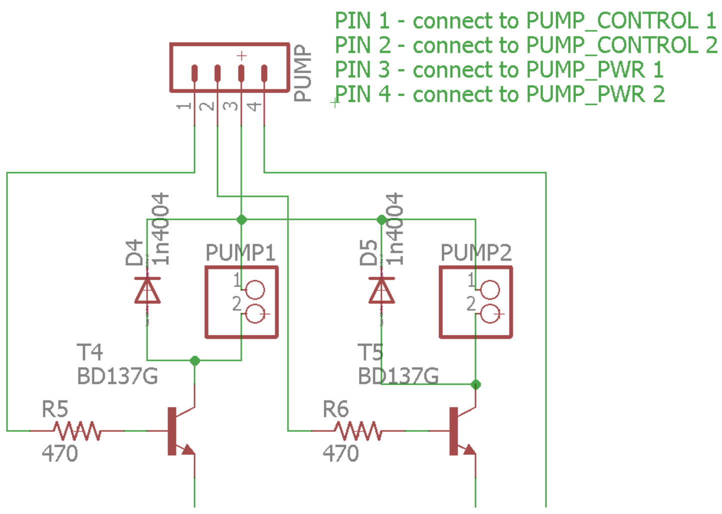

PIN 1 - connect to PUMP_CONTROL 1
PIN 2 - connect to PUMP_CONTROL 2
PIN 3 - connect to PUMP_PWR 1
PIN 4 - connect to PUMP_PWR 2
PUMP
1
2
3
4
D4
1n4004
PUMP1
D5
1n4004
PUMP2
T4
BD137G
T5
BD137G
R5
470
R6
470

**Figure 27 Pump control module**

## 10.2 Lights:

During the project, the plants are kept indoor and thus, are unable to get the adequate amount of sunlight required to grow. As a substitute, the plants are kept under the light of Led growth lamps for 12 hours per day. The growth lamps were powered through an external circuit and were controlled using an in-built timer.

## 10.3 Plants

The project setup consists of three sets of plants potted in two separate pots. This setup enables us to get the soil moisture readings and to deliver water to the plants, simulating the irrigation process. We decided to use Rosemary, as they are easily available and due to their ability to flourish in indoor environment. Furthermore, Rosemary plants require very low maintenance which makes them suitable for the 2-month test period of the project. [33]

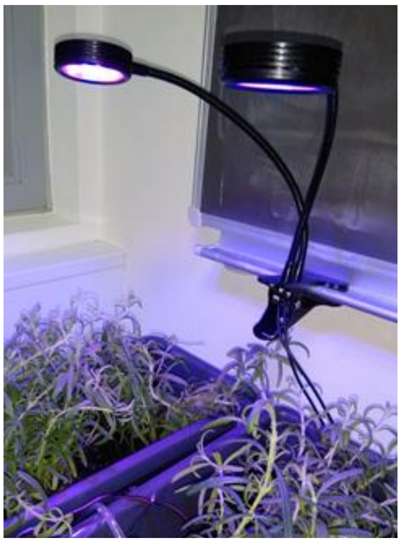


**Figure 28 Growth lamps and Plants**

## 10.4 Time-lapse

A Raspberry Pi zero with a camera was added to the test setup as an extra feature. It is taking photos of plants at regular intervals which will be used to create a time-lapse from our experiment. Because this single board computer is running Raspbian (operating system based on Debian Linux), it requires extra steps and a different programming method than an Arduino. First, an SD card needs to be formatted and flashed with an operating system. In this case, Raspbian LITE. Since Raspberry Pi will be used without display, keyboard and mouse it must be configured "headless" which means it cannot be accessed directly. To do so, details of the network which the RPi is supposed to connect to, must be input into a configuration file on the SD card (wpa_supplicant.conf). Then, it can be accessed through an SSH (secure shell) from a device connected to the same network. The RPi is programmed and configured through a command terminal. For the purpose of the experiment, time-lapse camera was enabled and a simple script was written (Appendix 16.3 Timelapse code). To ensure that the script is not terminated when SSH session is closed, program "Screen" was used. It simulates an open SSH session even after closing it which can be reassumed in the future. Photos will be combined to create a time-lapse after the plant growing experiment is finished.

# 11 Experiments

## 11.1 Sensor Experiment

In order to acquire reliable values from the sensor, it has been tested under different conditions and the readings had to be adjusted, based on the results to ensure the water content values data is as close to real value as possible.

First, the output of the sensor was checked, based on soil water content. Gravimetric and volumetric moisture content values were evaluated for two samples. Before the experiment, the soil sample was spread thinly and dried for three days in room temperature (21°C ±1°) in order to improve the quality of collected data. Then the soil was divided into multiple, smaller samples of 200 cm$^3$ and 50g which then have been used for further tests.

It was observed that sensor readings vary depending on various factors like soil compactness, contact between sensor and soil and how deep probes are inserted into the measured medium.

Test 1.

Based on this observation, two samples were prepared and evaluated. First one was additionally compressed to 145cm$^3$ and extra soil was added on top, resulting in a sample weighting 70g and having volume of 200cm$^3$. The second sample was 200cm$^3$ and 50g.

An even distribution of water in the soil is important for the reliability of the experiment, so a liquid was dosed with a syringe in various points around the top surface of the sample. Then the sensor output was monitored for 10 minutes before adding more water. To ensure that the gravimetric moisture content is constant in both samples, different water volumes were added to the samples. Sample 1 received 3.5ml of water in each interval and sample 2 2.5ml.

Sample 1 received 87.5ml of water during a 4h period which equals to 125% gravimetric and 43.75% volumetric value.

Sample 2 received 62.5ml of water during 4h period which equals to 125% gravimetric and 31.25% volumetric value.

Test data was gathered from a capacitive, a resistive and a temperature sensor using an Arduino microcontroller and saved to an SD card every 10 seconds. The used code can be seen in the appendix (16.5 Data logging code)

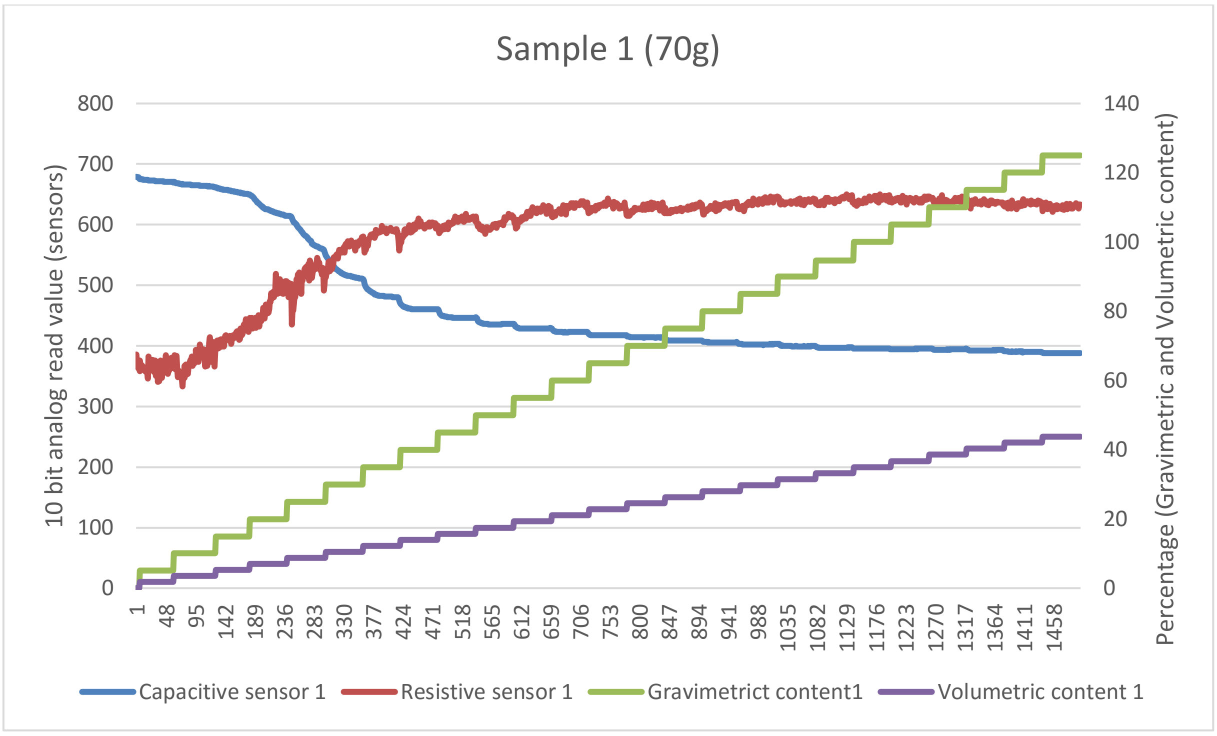


**Chart 1 Sample 1**

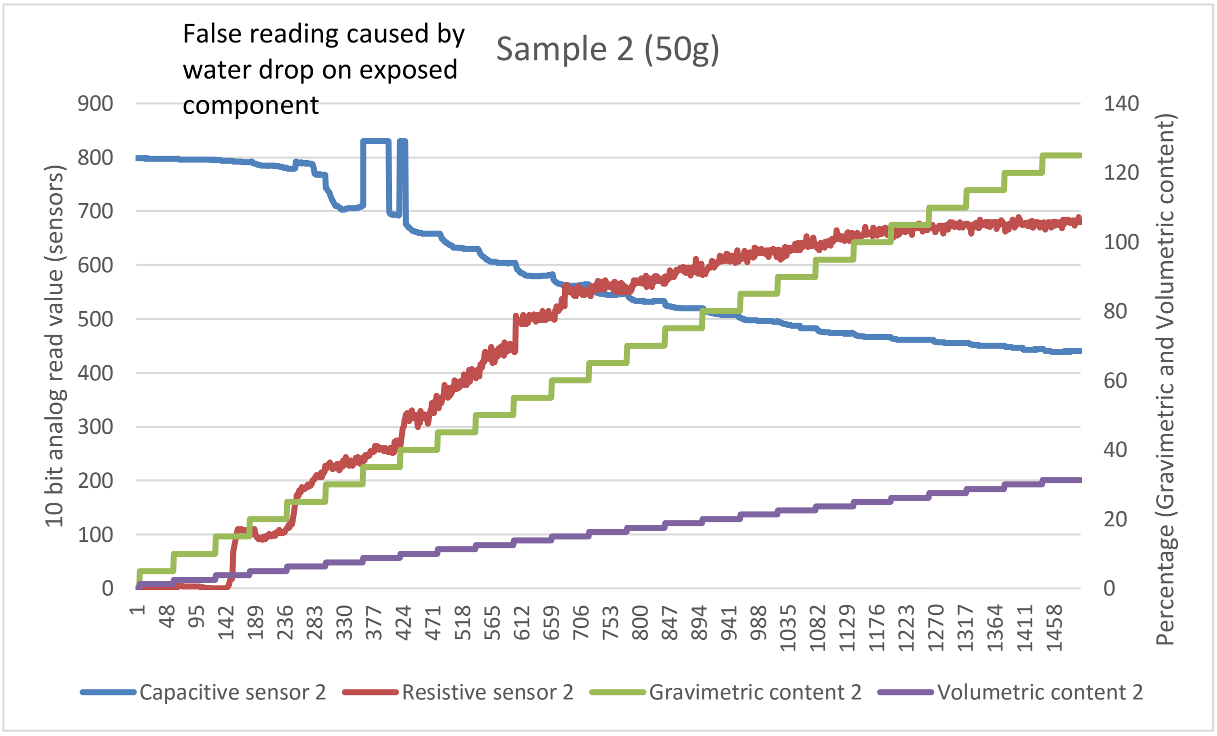


**Chart 2 Sample 2**

*To provide better readability temperature data was omitted. Temperature at the beginning of the test was 19°C and dropped to 17°C because of water temperature. (Both samples)*

Test 2.

The purpose of this experiment is to check variations of sensor output induced by a temperature difference. Two groups of three samples (15, 30, 45 ml) were prepared. The first group was placed on a heater while the second group was kept outside a window. After five hours, the sensors were placed in the samples to evaluate the output. The results have been compared in the table below.

| | Sample 1 (15ml) | | Sample 2 (30ml) | | Sample 3 (45ml) | |
|---|---|---|---|---|---|---|
| Temp | Capacitive | Resistive | Capacitive | Resistive | Capacitive | Resistive |
| 3°C | 638 | 161 | 635 | 486 | 605 | 621 |
| 17°C (Test 1) | 710 | 234 | 562 | 552 | 483 | 643 |
| 22°C | 799 | 0 | 670 | 164 | 585 | 656 |

It is not possible to draw a conclusion whether the temperature affects the sensor output, based on the table above. Variations in contact between sensor and soil affect the output significantly and another test must be performed in order to evaluate the relationship between temperature and reading error.

## 11.2 Watering test

In addition to constructing a device that could solve the problem,  we designed test to proof that our solution works properly and at a decent level of efficiency. The purpose of the experiment is to examine if there is a difference between irrigation based on schedule and relying on readings from sensors. To control environment in which experiment is conducted, saplings of plants of the same kind should be grown in one room, in pots placed next to each other. Soil in both pots should be watered automatically, using water pumps, to avoid human error. Even though only one set of plants would be watered according to moisture readings from sensors, pot storing another set should contain sensors as well so as to gather data that could be useful while comparing both methods of irrigation. If development of plants in both pots will be similar then it will proof that our device works properly. As mentioned above, rosemary was used for purpose of our experiment but unfortunately, we did not manage to start the it due to malfunctioning code and lack of time to debug it which will be discussed in next chapters.

# 12 Results

At the time of writing this report, the system is not completely operational and still requires debugging of the communication protocol, which worked during separate tests but malfunctions in the final version.  Also, the ATmega328p microcontroller does not have enough memory to run the program responsible for communication, data logging and interface at the same time without crashing. Especially the SD library requires 512 bytes of dynamic memory for a buffer. The program was successfully run on Arduino Mega (ATmega2560) which has 256kb flash memory, 8kb SRAM eight and four times more than the former microcontroller respectively.

The circuit boards for the main and the wireless nodes have not been made because the final layout has not been decided. Only the UPS circuit has been built and is operational but could still be improved by implementing fuses to protect the system from short circuits and replacing the resistor responsible for current limiting with a proper valued one. At this point, the current is set lower than it should be. The converter module could also be replaced by a linear regulator if its limitations can be addressed (See Discussion 13.1 UPS improvement). Since ambient temperature sensors have not been implemented, float charging voltage adjustment is set to 6.9V

As for the test setup, pumps are not able to water multiple plants at once because of the low flow rate. Only the plant closest to the pump receives a sufficient amount of water and only a fraction of this amount is delivered to further plants.

The sensor experiment provided useful data about the sensor output based on water content and variations caused by contact and compression of the soil. A better test to evaluate temperature differences should be performed and salt or fertilizer influence should be evaluated as well.

# 13Discussion

## 13.1 UPS improvement

The design could be simplified by replacing the converter module with a linear regulator and implementing solutions on the microcontroller side to mitigate problems associated with unstable input voltage. First problem caused by varying input voltage is constantly changing reference for analogRead functions which induces measurement error. To solve this problem external reference (5V, 3.3V, 2.5V depends on part used) or internal 1.1V reference can be used to ensure precise measurements. External 5V or 3.3V would be preferred since most sensors can operate in this range and output does not need to be scaled down to be measured. Second problem caused by unstable input voltage is variation in output voltage of digital pins which is not a problem in most cases but causes issues with PWM based voltage adjustment used to set float charging. Lowering input voltage causes output of digital to analog converter in UPS to drop for the same value of PWM. To solve this issue software solution could be implemented. Program would be required to monitor battery voltage and ambient temperature to adjust PWM signal accordingly.

## 13.2 Sensor experiment

Since output of the sensors is dependent on how well it maintains contact with soil placement procedure must be standardized to ensure comparable results. Also, capacitive sensor output seems to be stable, while resistive is noisy. During experiment, function was used to take 10 measurements, registered in 50 intervals, and calculate average value, which was supposed to reduce noise. Until solution to this problem is not found, resistive sensor should not be used for precise measurements. Further tests should be performed to provide better insight into variation in sensor output induced by difference  in temperature as well as salt content, which were not performed at all. In addition, sensors should be protected from water and dirt by conformal coating or other suitable method to prevent false readings.( similar to data seen in page 34 **Błąd! Nie można odnaleźć źródła odwołania.**)

### 13.3 Microcontroller choice for main node

Depending on system complexity and user requirements, different microcontrollers must be used to provide that functionality. Small systems with up to 50 nodes and a basic user interface can be run on boards based on Atmega328p (Arduino Uno, Nano, Pro Mini) or similar. Medium-sized systems with 100 to 150 nodes and a more advanced user interface will require a microcontroller that provides more SRAM(static random access memory) and flash memory like ATmega2560 (Arduino Mega). Aforementioned chip provides additional benefit of 3 additional hardware serial ports which can be used connect additional wireless modules operating at different frequencies to speed up data gathering process in medium systems. Bigger systems have not been taken into consideration at this stage of the project. Since the SD come with an in-built buffer of 512 bytes, it is the biggest single consumer of SRAM. Optionally, it is possible to remove the SD card reader to free up dynamic memory.

### 13.4 Interface

We have not done enough research on string processing capabilities of Arduino Nano. Unfortunately, it turned out that an interface that includes a lot of stings and editing them is too much for a microcontroller that has very limited amount of memory and is not designed for such an advanced task. Limiting options in menu could be a possible solution to this problem. In addition, part of string processing functions could be removed to free program memory. Another possible solution would be using Arduino Mega, that has more memory and could handle the interface.

### 13.5 Pumps

The water pressure and flowrate provided through peristaltic pumps are not enough for the uniform distribution of the water, all over the pot. Alternatively, any 12 V pump with a higher flow rate in combination with solenoid valve can be used which would ensure a uniform distribution of water but at the cost of reduced efficiency.

# 14 Conclusion

Since our prototype turned out to be only partially operational at this time, we cannot make a sound conclusion on the viability of our system. Despite testing the communication protocol and the LCD user interface extensively and proving them to work separately, the combined version of code contained bugs which caused troubles with starting the experiment. However, spending several more hours on debugging code should enable us to deal with that problem.

As we could not test the system, we cannot make any statement on how much of an assistance our system actually is. As mentioned before, sensor readings depend on many factors. Data acquired through experiments is only applicable under the exact same conditions. It requires extensive testing to create a system that is applicable in all conditions.

Although we cannot compare our system to the ones used in the industry, we can speculate that a system based on sensor data and correctly configured and optimized for its specific application can save resources that might not be so significant for small-sized crops but when implemented on a larger scale can lead to globally meaningful savings.

# 15 References

[1] F. R. Rijsberman, “Water scarcity: Fact or fiction?,” *Agricultural Water Management,* vol. 80, no. 1-3 SPEC. ISS., pp. 5-22, 2006.

[2] S. L. Postel, “Water for Food Production : Will There Be Enough in 2025 ? crucial to meeting future food needs,” *BioScience,* vol. 48, no. 8, pp. 629-637, 1998.

[3] “http://www.fewresources.org/water-scarcity-issues-were-running-out-of-water.html,” [Online].

[4] D. C. d. F. a. F. R. Molden, “Water Scarcity: The Food Factor,” 2007. [Online]. Available: http://issues.org/23-4/molden/. [Accessed 2017].

[5] “World population projected to reach 9.8 billion in 2050, and 11.2 billion in 2100,” 21 June 2017. [Online]. Available: https://www.un.org/development/desa/en/news/population/world-population-prospects-2017.html. [Accessed December 2017].

[6] . R. Madel, “Why Don't All Farmers Use Drip Irrigation,” 25 04 2016. [Online]. Available: http://www.gracelinks.org/blog/6965/why-don-t-all-farms-use-drip-irrigation. [Accessed 12 2017].

[7] T. Folnovic, “Drip Irrigation as the Most Efficient Irrigation System Type,” 06 06 2017. [Online]. Available: http://blog.agrivi.com/post/drip-irrigation-as-the-most-efficient-irrigation-system-type. [Accessed 12 2017].

[8] “Farming: Wasteful water use,” [Online]. Available: http://wwf.panda.org/what_we_do/footprint/agriculture/impacts/water_use/. [Accessed 12 2017].

[9] “Drinking-water,” 2017. [Online]. Available: http://www.who.int/mediacentre/factsheets/fs391/en/. [Accessed 12 2017].

[10] “Nations Encyclopedia- India - Agriculture,” [Online]. Available: http://www.nationsencyclopedia.com/economies/Asia-and-the-Pacific/India-AGRICULTURE.html.

[11] I. N. Census. [Online]. Available: http://censusindia.gov.in/2011-prov-results/data_files/india/Final_PPT_2011_chapter6.pdf.

[12] “Power Sonic,” [Online]. Available: http://www.power-sonic.com/images/powersonic/technical/1277751263_20100627-TechManual-Lo.pdf. [Accessed 2017 12 19].

[13] “Arduino.cc,” Arduino, [Online]. Available: https://www.arduino.cc/en/Products/Compare. [Accessed 17 12 2017].

[14] “https://www.powerstream.com/SLA.htm,” 5 12 2017. [Online].

[15] “www.techopedia.com,” [Online]. Available: https://www.techopedia.com/definition/25705/communication-protocol. [Accessed 15 12 2017].

[16] p. 2. “HC-12 Wireless Serial Port Communication Module User Manual V1.18”, “www.elecrow.com,” 10 2012. [Online]. Available: https://www.elecrow.com/download/HC-12.pdf. [Accessed 15 12 2017].

[17] p. 6. “HC-12 Wireless Serial Port Communication Module User Manual V1.18”, “www.elecrow.com,” 10 2012. [Online]. Available: https://www.elecrow.com/download/HC-12.pdf. [Accessed 15 12 2017].

[18] “www.circuitsbasics.com,” [Online]. Available: http://www.circuitbasics.com/basics-uart-communication/. [Accessed 15 12 2017].

[19] “www.rfwireless-world.com,” [Online]. Available: http://www.rfwireless-world.com/Terminology/UART-vs-SPI-vs-I2C.html. [Accessed 15 12 2017].

[20] “www.arduino.cc,” Arduino, [Online]. Available: https://www.arduino.cc/en/Reference/SD. [Accessed 15 12 2017].

[21] M. Mitescu and I. Susnea, “Using The I2C Bus,” in *Microcontrollers in Practice*, Springer, 2005, pp. 73-77.

[22] J. P. S. Frank de Brabander, “https://github.com/fdebrabander/,” 9 3 2017. [Online]. Available: https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library. [Accessed 28 11 17].

[23] Arduino, “https://www.arduino.cc/reference/en,” [Online]. Available: https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/. [Accessed 1 11 2017].

[24] bildr, “http://bildr.org,” 2 8 2012. [Online]. Available: http://bildr.org/2012/08/rotary-encoder-arduino/. [Accessed 2017 11 1].

[25] “EcuRed- Baterias NiMH (in Spanish),” [Online]. Available: http://www.ecured.cu/index.php/Bater%C3%ADas_de_Ni-MH.

[26] D. H. F. &. C. S. Skoog, Principles of Instrumental Analysis, 2007.

[27] C. Kittel, rinciples of Solid State Physics, 2005.

[28] “TMP36 Temperature Sensor,” [Online]. Available: https://learn.adafruit.com/tmp36-temperature-sensor/using-a-temp-sensor.

[29] W. C. Hughes, “Soil Moisture Sensor”. Patent US 20100277185 A1, 2010.

[30] R. Berberich, “Resistive moisture sensor”. USA Patent US 5739430 A, 1998.

[31] “WORKING PRINCIPLE PERISTALTIC PUMPS,” [Online]. Available: https://www.verderliquids.com/int/en/pumps-by-principle/how-do-peristaltic-pumps-work/working-principle-peristaltic-pumps/. [Accessed december 2017].

[32] “Peristaltic Pump Wear Factors,” [Online]. Available: http://blue-white.com/peristaltic-pump-wear-factors/. [Accessed december 2017].

[33] “OVERWINTERING ROSEMARY: GROWING ROSEMARY INDOORS,” 20 July 2017. [Online]. Available: https://www.almanac.com/blog/gardening/garden-journal/overwintering-rosemary-growing-rosemary-indoors. [Accessed december 2017].

[34] [Online]. Available: https://www.elecrow.com/download/HC-12.pdf, “HC-12 Wireless Serial Port Communication Module User Manual V1.18”, page 2, published in October 2012, accessed at 15.12.2017, 19:02. [Accessed 2017].

[35] “FARMS WASTE MUCH OF WORLD'S WATER,” 19 03 2006. [Online]. Available: https://www.wired.com/2006/03/farms-waste-much-of-worlds-water/. [Accessed 11 2017].

[36] “How do peristaltic dosing pumps work?,” [Online]. Available: https://www.verderflex.com/en/how-do-peristaltic-pumps-work/. [Accessed december 2017].

**Abbreviations**

CRC8 – Cyclic Redundancy Check 8-bit  
*SLA – Sealed Lead Acid (also known as VRLA)*  
*VRLA – Valve Regulated Lead Acid*  
*UPS – Uninterruptible Power Supply*  
*NIMH – Nickel-Metal Hydride*  
*I/O - Input / Output*  
*SD – Secure Digital*  
*V – Voltage*  
*R- Resistance*  
*I – Current*  
*A - Ampere*  
*Op-amp – Operational Amplifier*  
SRAM – *Static Random Access Memory*  
*RPi -Raspberry Pi*  
UART – *Universal Asynchronous Receiver/Transmitter*  
MOSI – *Master Output, Slave Input*  
MISO – *Master Input, Slave Output*  
SPI – *Serial Periphal Interface*  
SCK – *Serial Clock*  
SS – *Slave Select*  
RXD – *Receive Exchanged Data*  
TXD – *Transmit Exchanged Data*  
SET – *Settings*  
bps – *bits per seconds*  
LCD – *Liquid Crystal Display*  
I2C – *Inter Integrated Circuit*  
SCL -*Serial Clock*  
SDA – *Serial Data*

# 16 Appendices

## 16.1 Gateway node code

### 16.1.1 Master.ino

```
#include "defs.h"
#include "interface.h"

extern SoftwareSerial comSerial;

using namespace std;

//Rotation and button detection code is based on
//http://bildr.org/2012/08/rotary-encoder-arduino/


void setup() {
    /** Sets up the Serial object and calls two
        distinct function that take care of the necessary
        set_up for the SD card and the interface as we wanted to keep
        this file as short as possible*/
    comSerial.begin(9600);
    interfaceSetup();
    card_setup();
}



void loop() {
    /** The program can find all nodes with the same system ID
        automtically, but it is possible to enter nodes manually.
        Please write:
            nodes.push_back(new Node(args))
        and replace args with the pointer to the next node in direction to the
        master and the address (e.g. the element of the vector).
        If the next node is the master, just write NULL)*/
    vector <Node*> nodes = find_nodes();

    /** Iterates until it one of the check functions return true.
        Then it starts the corresponding code and
        executes it to end after which
        it resumes iterating. This structure means that during the
        execution of one block of code, the other on cannot
        be executed. As long as the LCD is in use, the program will
        ignore the scheduling and vice versa while the program is executing
        the scheduled communication and the processing of the received data,
        the LCD will be irresponsive*/

    while (true){
        if (is_scheduled_now()){
            getData(nodes);
            delay(1000);
            pump(nodes);
        }
        if (buttonInput()){
            initLcd(dry_alert(nodes), wet_alert(nodes), F("NOT AVAILABLE"), F("NOT A
VAILABLE"));
        }
    }
}
```

### 16.1.2 A2A_master.cpp

```
#include <SoftwareSerial.h>
#include <ArduinoSTL.h>
#include <stdint.h>
#include "defs.h"

#define SYSID 222
#define ADDRESS 0

extern SoftwareSerial comSerial;

//the 3 function_codes
#define FC_SENSOR 1
#define FC_CHECK 2
#define FC_PRESENCE 3

using namespace std;

Node::Node(Node *p, uint8_t a){
    /** creates an instance of class node with the pointer to the next
        node in direction of the master (NULL if the next node is the master)
        and the address*/

    next = p;
    address = a;
}

void Node::get_pathway(vector <uint8_t> & p){
    /** A recursive function that calls itself (as the member function
        of all nodes in the pathway, until it reaches a node with next
        pointing to NULL, which then adds its own address and all function that were
        called before in the recursion add their own address until it finally reaches the
        original node which add its address and then terminates the recursion.*/

    if (next = NULL){
        p.push_back(0);
    }
    else{
        next->get_pathway(p);
    }
    p.push_back(address);
}

bool Node::call(uint8_t function_code){
    /** Creates and sends a complete message including
        the desired function code and then calls wait()*/
    vector <uint8_t> message, pathway;
    uint8_t checksum;

    message.push_back(SYSID);
    get_pathway(pathway);
    message.push_back(pathway[1]);
    message.push_back(pathway.size());
    for (int i=0; i<pathway.size(); i++){
        message.push_back(pathway[i]);
    }
    message.push_back(function_code);
    message.push_back(CRC8(&message[0], message.size()));
    for (uint8_t i=0; i<message.size(); i++){
        comSerial.write(message[i]);
    }
    return (wait(function_code, pathway.size()));
}

bool Node::wait(uint8_t function_code, uint8_t size){
    /** calculates how long the program has to wait for a response
        based on the pathway size. If the pathway size is greater than
        2, then there is a intermediate node. If that is the case, then
        the program stops waiting after 200 milliseconds, to empty the buffer
        so that the reply does not mix up with the relay of first node in the
        pathway.*/
    if (function_code == FC_PRESENCE || function_code == FC_SENSOR){
        if (size > 2){
            delay(200);
            while (comSerial.available() > 0){
                uint8_t buf = comSerial.read(); //empty the buffer
            }
            delay(90*2*(size-1) - 200);
        }
        else{
            delay(200);
        }
    }
    else if (function_code == FC_CHECK){
        if (size > 2){
            delay(200);
            while (comSerial.available() > 0){comSerial.read();}
            delay(90*2*(size-
1) - 200 + 180*254); //254 possible addresses are being called when executing FC_CHE
CK
        }
        else if (size = 2){
            delay(90 + 180*254);
            while (comSerial.available() > 0){comSerial.read();}
        }
    }
    if (comSerial.available() == 0){
        sys_log(String(F("#")) + address + String(F(" TIMEOUT")));
        return false;
    }
    return true;
}

bool Node::check_validity(vector <uint8_t> & message)
{
    /** Checks the validity of a received transmission. This
        includes the system ID, the direct address, the checksum,
        the pathway. Every received byte is being checked except for
        the data. If an error has been found, this function automatically
        logs it along with the corresponding time stamp and then returns false*/

    String error = F("");
    bool flag = false;
    bool valid = true;
    uint8_t *message_p = &message[0];
    if (CRC8(message_p, message.size()) != 0){
        valid = false;
        error = error + F("CHECKSUM_WRONG ");
    }
    if (message[0] != SYSID){
        valid = false;
        error = error + F("SYSID_WRONG ");
    }
    if (message[1] != 0){
        valid = false;
        error = error + F("DIRECT_ADDRESS_WRONG ");
    }

    vector <uint8_t> pathway;
    get_pathway(pathway);
    if (message[2] != pathway.size()){
        valid = false;
        error = error + F("PATHWAY_SIZE_WRONG ");
    }

    message.pop_back();
    reverse_elements(message, 3, 2+message[2]);

    for (int i=3; i<2+message[2]; i++){
        if (message[i] != pathway[i-3]){
            valid = false;
            if (flag == false){
                error = error + F("PATHWAY_WRONG ");
                flag = true;
            }
        }
    }

    if (valid == false){
        sys_log("#" + (String)address + " " + error);
    }
    return valid;
}

void Node::reverse_elements(vector <uint8_t> & vec, uint8_t first_e, uint8_t last_e)
{
    /** Reverses the elements of a vector with a given start and end element.
        This is necessary to verify the validity of the pathway of a received
        transmission.*/

    uint8_t i, k;
    k = last_e;
    uint8_t temp;
    for (i=first_e; i<=last_e; i++){
        temp = vec[i];  //first element is saved
        vec[i] = vec[k];//last element is copied to the beginning
        vec[k] = temp;  //first element is copied to the end
        k--;
        if ((k < i) || (k = i)){
            break;
        }
    }
}

vector <uint8_t> extract_data(vector <uint8_t> & message){
    /** Takes a message (unaltered aside from the reversed pathway
        and erased function_code/checksum) and returns the data.
        It can handle a varying amount of bytes in the data block*/

    vector <uint8_t> data;
    for (uint8_t i=2+message[2]+1; i<message.size(); i++){
        data.push_back(message[i]);
    }
    return data;
}

vector <Node*> find_nodes(){
    /** This function first calls all possible addresses directly with
        FC_PRESENCE as the function code, to get all slaves in direct range.
        Then it copies the pointers to all slaves found into the vector temp,
        which is then used to call all slaves found with FC_CHECK as function code.
        The called nodes will return addresses of all slaves they found in their neighborhood.
        These addresses are cleaned to remove all addresses that are already inside the vector nodes
        to prevent duplicates and added to nodes and temp2. After temp has been iterated to the end,
        all elements will be erased, and temp2 will be copied into temp1 and subsequently erased.
        This process is iterated until no new addresses are being found at one time.*/

    vector <Node*> nodes;
    vector <Node*> temp;
    vector <Node*> temp2;
    vector <uint8_t> addresses;
    vector <uint8_t> reply;

    for (uint8_t i=1; i<=255; i++){
        Node *node = new Node(NULL, i);
        if (node->call(FC_PRESENCE) == true){
            while (comSerial.available() > 0){
                reply.push_back(comSerial.read());
            }
            if (node->check_validity(reply) == true){
                nodes.push_back(new Node(NULL, i));
            }
            reply.clear();
        }
    }

    for (uint8_t i = 0; i<temp.size(); i++){
        if (temp[i]->call(FC_CHECK) == true){
            while (comSerial.available() > 0){
                reply.push_back(comSerial.read());
            }
            if (temp[i]->check_validity(reply) == true){
                addresses = extract_data(reply);
                clean_addresses(addresses, nodes);
                for (uint8_t k = 0; k<addresses.size(); k++){
                    nodes.push_back(new Node(temp[i], addresses[k]));
                    temp2.push_back(nodes[nodes.size() - 1]);
                }

                if (temp2.size() == 0){break;} //If no new nodes have been found this time, the loop stops.

                clean(temp);
                for (uint8_t k = 0; k<temp2.size(); k++){
                    temp.push_back(temp2[k]);
                }
                clean(temp2);
            }
        }
    }
    return nodes;
}

void clean_addresses(vector <uint8_t> & a, vector <Node*> & nodes){
    for (uint8_t i = 0; i<a.size(); i++){
        for (uint8_t k = 0; nodes.size(); k++){
            if (a[i] == nodes[k]->address){
                a.erase(i);
                break;
            }
        }
    }
```

```
}

void getData(vector <Node*> & nodes)
{
    /**Calls all nodes in the vector nodes and requests the sensor data.
        This data corresponds to (for moisture sensors) the concentration
        of water mass in relation to the total mass, which was being mapped
        to the full range of 1 byte (0 to 255). After acquiring the
        data, it logs all values in % with two digits after the comma
        along with the corresponding time stamp*/

    print_time_stamp();

    vector <uint8_t> reply;
    vector <uint8_t> data;

    for (int i=0; i<nodes.size(); i++){
        if (nodes[i]->call(FC_SENSOR) == true){
            while (comSerial.available() > 0){
                reply.push_back(comSerial.read());
            }
            if (nodes[i]->check_validity(reply) == true){
                data = extract_data(reply);
                check_sensor_data_validity(data, nodes[i]->address);
                nodes[i]->c_w_capac = data[0];                //capacity
                nodes[i]->c_w_resist = data[1];           //resistive
                //add more if necessary
                //nodes[i].temp = data[3];
                vector <double> convert = convert_data(data);
                log_data(convert, nodes[i]->address);
            }
        }
    }
}

void clean(vector <Node*> & n){
    /** Erases all elements of the vector that was passed by reference to this function*/
    while (n.size() > 0){n.pop_back();}
}

vector <double> convert_data(vector <uint8_t> & data){
    /** Converts incoming data to type double (with two digits behind the comma)*/
    vector <double> convert;
    double temp;
    for (uint8_t i=0; i<data.size(); i++){
        temp = (double)data[i]/2.55;
        temp *= 100;
        temp = (int)temp;
        temp /= 100;
        convert.push_back(temp);
    }
    return convert;
}

void check_sensor_data_validity(vector <uint8_t> & data, uint8_t address){
    /** If a returned byte of data is 0 or 255, this corresponds to 0%
        or 100% water concentration. This can be due to natural causes, but
        is likely to indicate a sensor failure.*/

    for (uint8_t k=0; k<data.size(); k++){
        if (data[k] == 0){
            sys_log(String(F("#")) + (String)address + String(F(" SENSOR_#")) + (String)k + String(F("_DATA_0")));
        }
        if (data[k] == 255){
            sys_log(String(F("#")) + (String)address + String(F(" SENSOR_#")) + (String)k + String(F("_DATA_255")));
        }
    }
}

//CRC-8 - based on the CRC8 formulas by Dallas/Maxim
//code released under the therms of the GNU GPL 3.0 license
byte CRC8(const byte *data, byte len) {
    byte crc = 0x00;
    while (len--) {
        byte extract = *data++;
        for (byte tempI = 8; tempI; tempI--) {
            byte sum = (crc ^ extract) & 0x01;
            crc >>= 1;
            if (sum) {
                crc ^= 0x8C;
            }
        extract >>= 1;
        }
    }
    return crc;
}
```

### 16.1.3 Setup.cpp

```
#include "defs.h"

/**
    This pin definitions can be changed by the user or the
    implementing professional in order to reflect the set-up
    this program is operating with*/

//pin definitions
const uint8_t TXD = 8;
const uint8_t RXD = 7;
const uint8_t V_ADJUST = 6;


//set up a new serial object
SoftwareSerial comSerial(TXD, RXD);

/**
    Change these variables to the current time before uploading the sketch.
    In case of the Arduino was/got reset, these variables have to be manually changed
    to reflect the current time.
    Please use only numbers between 0 and 99 and regarding the hours please use the 24h system
*/

uint8_t hour = 09;
uint8_t minute = 51;
uint8_t second = 0;

uint8_t day = 17;
uint8_t month = 12;
uint8_t year = 17;
```

### 16.1.4 Log.cpp

```
#include <SPI.h>
#include <Fat16.h>
#include <Fat16util.h>
```

```
#include "defs.h"

SdCard card;
Fat16 file;

const uint8_t SS_PIN = 10;


using namespace std;

void card_setup(){
    /** Sets up the SD card. Another digital pin can
        be used instead of 10 as SS_PIN, but 10 as to
        left open as an output*/

    card.begin(SS_PIN);
    Fat16::init(&card);
}

void print_time_stamp(){
    /** Prints the time stamp once inside data_log.txt*/
    file.open(F("data_log.txt"), O_WRITE);
    file.println();
    file.println(time_stamp());
    file.close();
}

void sys_log(String str){
    /** Prints an error notification combined with the current time stamp*/
    file.open(F("sys_log.txt"), O_WRITE);
    file.println(time_stamp());
    file.println(str);
    file.println();
    file.close();
}

void log_data(vector <double> & data, uint8_t address){
    /** Adds all numbers and the address together after typecasting them to strings*
/
    String d = (String)address;
    for (uint8_t i = 0; i<data.size(); i++){
        d += " " + (String)data[i];
    }
    file.open(F("data_log.txt"), O_WRITE);
    file.println(d);
    file.close();
}
```

### 16.1.5 Alert.cpp

```
#include <Arduino.h>
#include <ArduinoSTL.h>

//uses percentages for Moist
#define LOWER_LIMIT_MOIST 10
#define UPPER_LIMIT_MOIST 38
//#define LOWER_LIMIT_TEMP 100
//#define UPPER_LIMIT_TEMP 100

#define UPPER_LIMIT 1
#define LOWER_LIMIT 0

#include "defs.h"

String dry_alert(vector <Node> & nodes)
```

```
{
    String text;
    for (uint8_t i = 0; i<nodes.size(); i++)
    {
        check_value(((nodes[i].c_w_capac + nodes[i].c_w_resist) / 5.1), nodes[i].address, LOWER_LIMIT_MOIST, LOWER_LIMIT, text); //5.1 = 2 * 2.55
    }
}

String wet_alert(vector <Node> & nodes)
{
    String text;
    for (uint8_t i = 0; i<nodes.size(); i++)
    {
        check_value(((nodes[i].c_w_capac + nodes[i].c_w_resist) / 5.1), nodes[i].address, UPPER_LIMIT_MOIST, UPPER_LIMIT, text);
    }
}

//String cold_alert(vector <Node> & nodes)
//{
//  String text;
//  for (uint8_t i = 0; i<nodes.size(); i++)
//  {
//      check_value(nodes[i].temp, nodes[i].address, LOWER_LIMIT_TEMP, LOWER_LIMIT, text);
//  }
//}

//String hot_alert(vector <Node> & nodes)
//{
//  String text;
//  for (uint8_t i = 0; i<nodes.size(); i++)
//  {
//      check_value(nodes[i].temp, nodes[i].address, UPPER_LIMIT_TEMP, UPPER_LIMIT, text);
//  }
//}

String check_value(int16_t val, uint8_t address, int16_t limit, bool limit_type, String text)
{
    if ((val > limit && limit_type == UPPER_LIMIT) || val < limit && limit_type == LOWER_LIMIT)
    {
        if (text == F(""))
        {
            text = (String)(int)address;
        }
        else
        {
            text += String(F(", ")) + (String)(int)address;
        }
    }
    return text;
}
```

### 16.1.6 Pump.cpp

```
using namespace std;
#include "defs.h"

/** This file is supposed to test the system by managing a pump
    based on the received sensor data.
    It is of no value for the actual system.*/
```

```

#define SOIL_MASS 3000                      //needs to be rechecked
#define FLOW_RATE 45.87                     //milliliters per minute, pump running at 12V
#define DES_WATER_C 28                      //equal to 20ml water plus 50g soil
#define PUMP_START 24                       //starts pump only when water concentration is below...
#define PUMP_PIN 9                          //connected to the base of the transistor

unsigned long last_millis = 0;
unsigned long time_expired = 0;             //in milliseconds
uint16_t volume_used = 0;                   //volume already used in time peroid (in milliliters)
#define SAFEGUARD_VOLUME 500                //500 milliliters
#define SAFEGUARD_TIME 604800000            //7 days

//function use converted values
bool pump(vector <Node*> & nodes){
    /** This function manages the pump activity. It calls
        pump_run_time() to get the necessary runtime and then
        utilizes the safeguard to protect the plants. If either
        function returns 0 this function aborted on the spot.
        Otherwise it will run the pump until the given time has been reached
        after which it stops the pump*/

    uint16_t t = pump_run_time((double)nodes[0]->c_w_capac/2.55, (double)nodes[0]->c_w_resist/2.55);
    if (t == 0) {
        return false;
    }
    t = safeguard(t);
    if (t == 0) {
        return false;
    }
    unsigned long now_t = millis();
    digitalWrite(PUMP_PIN, HIGH);   //open pump
    unsigned long timeout = now_t + t;
    while (!(now_t == timeout || (now_t > timeout && now_t < timeout+100000))) {
        now_t = millis();
    }
    digitalWrite(PUMP_PIN, LOW);    //close pump
    return true;
}


//takes a concentration in % (between 0 and 100)
uint16_t pump_run_time(double m1, double m2){
    /** This function calculates the absolute volume
        of needed water as the difference of the given
        volume and the desired one. From there it calculates
        the necessary runtime by dividing by the flow rate and
        multiplying by 60,000 to get the runtime in
        milliseconds. If the concentration of water in the soil
        is above 24% it will not water.*/

    uint16_t average = (4*m1 + m2)/5;
    if (average >= PUMP_START) {
      return 0;
    }
    float volume_des = (DES_WATER_C/100.0) * SOIL_MASS;                 //desired volume of water in pot (in milliliters)
    float volume_real = (average/100.0) * SOIL_MASS;                    //actual volume of water in pot (in milliliters)
    return (uint16_t)(((volume_des - volume_real)/FLOW_RATE)*60000.0);  //necessary pump runtime in milliseconds
}

uint16_t safeguard(uint16_t t){
    /** This function allows the pump to only comsume a
        specified amount of water(500 ml)
        in a specfied time period (7 days) to
        prevent the pump from fatally overwatering
        due to corrupted data, improperly set/wired
        sensors or a mistake in the conversion of
        sensor data/calculation of pump runtime.
        It returns 0 if the limit is already met in
        said time period.*/

    unsigned long now_t = millis();
    unsigned long diff = now_t - last_millis;
    if (now_t < last_millis){
        unsigned long max = ~0;
        diff = (max - last_millis) + now_t;
    }
    time_expired += diff;
    while (time_expired > SAFEGUARD_TIME) {
        time_expired -= SAFEGUARD_TIME;
    }
    uint16_t volume = ((float)t/60000.0)*FLOW_RATE; //t/60000.0 because FLOW_RATE is in ml/min
    if (volume_used >= SAFEGUARD_VOLUME) {
        return 0;
    }
    if ((volume_used + volume) > SAFEGUARD_VOLUME){
        volume = SAFEGUARD_VOLUME - volume_used;
        volume_used = SAFEGUARD_VOLUME;
        return ((float)volume/FLOW_RATE)*60000.0;           //time in milliseconds until SAFEGUARD_VOLUME is reached;
    }
    return t;
}
```

### 16.1.7 Defs.h

```
/** This file includes the declarations of the class Node
    and all functions defined in the files mentioned below.
    For more information please refer to the function definition.*/

#include <SoftwareSerial.h>
#include <ArduinoSTL.h>
#include <Arduino.h>
using namespace std;
#include <stdint.h>

/** The member functions of class Node are defined in A2A_master.cpp*/
class Node{
    public:
        uint8_t address;
        Node *next;
        //8bit values
        uint8_t c_w_capac;   //capacitive
        uint8_t c_w_resist;  //resistive
        //uint8_t temp;
    public:
        Node(Node *p, uint8_t a);
        bool call(uint8_t function_code);
        bool wait(uint8_t function_code, uint8_t size);
        bool check_validity(vector <uint8_t> & message);
        void reverse_elements(vector <uint8_t> & vec, uint8_t first_e, uint8_t last_e);
        void get_pathway(vector <uint8_t> & p);
};

/**Defined in A2A_master.cpp*/
vector <Node*> find_nodes();
byte CRC8(const byte *data, byte len);
vector <uint8_t> extract_data(vector <uint8_t> & message);
void check_sensor_data_validity(vector <uint8_t> & data, uint8_t address);
void getData(vector <Node*> & nodes);
vector <double> convert_data(vector <uint8_t> & data);
void clean(vector <Node*> n);
void clean_addresses(vector <uint8_t> & a, vector <Node*> & nodes);

/**Defined in log.cpp*/
void sys_log(String str);
void log_data(vector <double> & data, uint8_t address);
void card_setup();
void print_time_stamp();

/**Defined in time.cpp*/
String double_digit(uint8_t t);
void update_time();
bool is_scheduled_now();
String time_stamp();

/**Defined in pump.cpp*/
bool pump(vector <Node*> & nodes);
uint16_t pump_run_time(double m1, double m2);
uint16_t safeguard(uint16_t t);

/**Defined in encoder.cpp*/
void encoderRotDir();
bool buttonInput();
void initLcd(String alert1 ,String alert2 ,String alert3, String alert4);
bool checkEncoder(String alert1 ,String alert2 ,String alert3, String alert4);
bool checkButton(bool *userPresence, String alert1 ,String alert2 ,String alert3, St
ring alert4);
void compareSum(byte sum);
void encoderValLim();

/**Defined in alert.cpp*/
String check_value(int16_t val, uint8_t address, int16_t limit, bool limit_type, Str
ing text);
String dry_alert(vector <Node*> & nodes);
String wet_alert(vector <Node*> & nodes);
//String cold_alert(vector <Node> & nodes);
//String hot_alert(vector <Node> & nodes);
```

### 16.1.8 Interface.h

```
#ifndef INC_2_INTERFACE_H
#define INC_2_INTERFACE_H

#endif //INC_2_INTERFACE_H

#include <SoftwareSerial.h>
#include <SPI.h>
#include <Fat16.h>
#include <Fat16util.h>


#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Arduino.h>

```

```
#define ENCODER_PIN_CLK 2
#define ENCODER_PIN_DT 3
#define BUTTON_PIN 4

struct Option {
  String text[4];
};

void interfaceSetup();
void encoderRotDir();
void compareSum(byte sum);
void encoderValLim();
bool buttonInput();
bool checkButton(bool *userPresence, String alert1 , String alert2 , String alert3, String alert4);
bool checkEncoder(String alert1 , String alert2 , String alert3, String alert4);
void initLcd(String alert1 , String alert2 , String alert3, String alert4);

void initMenus(int clickVal, int clickLimMax, int menuIndex, String alert1 , String alert2 , String alert3, String alert4);
void initMainMenu(int clickVal, int clickLimMax);
void initAlertsMenu(int clickVal, int clickLimMax, String alert1 , String alert2 , String alert3, String alert4);
void initNodesMenu(int clickVal, int clickLimMax);
void initSettingsMenu(int clickVal, int clickLimMax);
void initPowerMenu(int clickVal, int clickLimMax);
void initAutoPowerMenu(int clickVal, int clickLimMax);

void menuOutput(bool *userPresence);
void mainMenuOutput();
void alertsMenuOutput();
void nodesMenuOutput();
void settingsMenuOutput();
void powerMenuOutput(bool *userPresence);
void autoPowerMenuOutput();

void initText(Option option);
void isEmpty(String *subject, String confirmation);
String insertDash(int i);
String insertSpace(int i);
String createHeadline(String line);
String createLine(String line);
String createPageNum(String line, int clickVal, int clickLimMax);

void templateLeft(String menuName, String line1, String line2, String pageName, int clickVal, int clickLimMax);
void templateRight(String menuName, String line1, String line2, String pageName, int clickVal, int clickLimMax);
void templateMiddle(String menuName, String line1, String line2, String pageName, int clickVal, int clickLimMax);
void templateChoice(String line1, String line2, String line3, String instruction);
```

### 16.1.9 Encoder.cpp

```
//Rotation and button detection code is based on
//http://bildr.org/2012/08/rotary-encoder-arduino/

#include "interface.h"

// Set the LCD address to 0x27 for a 16 chars and 2 line display
LiquidCrystal_I2C lcd(0x27, 20, 4, LCD_5x8DOTS);

// store rotation in pins state changes
int g_pinVal = 0;

// store rotation in clicks
```

```
volatile int8_t g_clickVal = 0;
int8_t g_prevClickVal = -1;

// store rotation reading limit to avoid
// exceeding number of options in menu
int8_t g_clickLimMax = 3;
int8_t g_clickLimMin = 0;

// track current position in the interface
int8_t g_menuIndex = 0;

// store time since last button state switch
unsigned long g_lastDebounce = 0;

unsigned long lastMillis = 0;

// store lcd standby time in minutes
unsigned long g_lcdTimeOut = 60000;

// setup required hardware
void interfaceSetup()
{
  // set encoder and button pins as input
  // turn on pull-up resistor
  pinMode(ENCODER_PIN_CLK, INPUT_PULLUP);
  pinMode(ENCODER_PIN_DT, INPUT_PULLUP);
  pinMode(BUTTON_PIN, INPUT_PULLUP);

  // set encoder pins as interrupt
  attachInterrupt(0, encoderRotDir, CHANGE);
  attachInterrupt(1, encoderRotDir, CHANGE);

  // start lcd and turn the screen off imidiately
  lcd.begin();
  lcd.noBacklight();
  lcd.noDisplay();
}

void encoderRotDir()
{
  // store previous readings from encoder
  static int prevState = 0;
  int MSB = digitalRead(ENCODER_PIN_CLK); //MSB = most significant bit
  int LSB = digitalRead(ENCODER_PIN_DT); //LSB = least significant bit
  // convert 2 pin value to single number
  int encoded = (MSB << 1) | LSB;
  // add it to the previously encoded value
  int sum = (prevState << 2) | encoded;
  compareSum(sum);
  encoderValLim();
  prevState = encoded; //store this value for next time
}

/**Check current pin states with previous pin states to determine
   direction of rotation. Threre are 4 combinations for both rigth
   and left. Decrement or increment encoderValue
*/

void compareSum(byte sum)
{
  // counterclockwise rotation
  if (sum == 0b1101 || sum == 0b0100 || sum == 0b0010 || sum == 0b1011)
  {
    g_pinVal--;
    /**One physical "click" is equivalent to 4 pin state changes
       The statement below ensures that programm will respond to
```

```
        physical "click" instead of each pin change. It also works as
        a debounce.
      */
      if ((g_pinVal % 4) == 0)
      {
        g_clickVal--;
      }
    }
    // clockwise rotation
    else if (sum == 0b1110 || sum == 0b0111 || sum == 0b0001 || sum == 0b1000)
    {
      g_pinVal++;
      if ((g_pinVal % 4) == 0)
      {
        g_clickVal++;
      }
    }
}

// limit clickVal so as not to exceed number of pages in menu
void encoderValLim()
{
  if (g_clickVal > g_clickLimMax)
  {
    g_clickVal = g_clickLimMax;
  }
  else if (g_clickVal < g_clickLimMin)
  {
    g_clickVal = g_clickLimMin;
  }
}

// check if button was read
bool buttonInput()
{
  // ignore input if it was read 50ms
  // after previous one to cancel noises
  if (millis() - g_lastDebounce > 50)
  {
    g_lastDebounce = millis();
    if (digitalRead(BUTTON_PIN) == 0)
    {
      return true;
    }
  }
  return false;
}

// update LCD only when encoder reads new value
bool checkEncoder(String alert1 , String alert2 , String alert3, String alert4)
{
  if (g_prevClickVal != g_clickVal)
  {
    initMenus(g_clickVal, g_clickLimMax, g_menuIndex, alert1, alert2, alert3, alert4
);
    g_prevClickVal = g_clickVal;
    return true;
  }
  // refresh each second if user looks at time and date page
  if (g_menuIndex == 0 && g_clickVal == 0 && (millis() - lastMillis >= 1000))
  {
    lastMillis = millis();
    initMenus(g_clickVal, g_clickLimMax, g_menuIndex, alert1, alert2, alert3, alert4
);
    return false;
  }
  else
  {
    return false;
  }
}

// choose option in menu if button input was detected
bool checkButton(bool *userPresence, String alert1 , String alert2 , String alert3, String alert4)
{
  if (buttonInput() == true)
  {
    menuOutput(userPresence);
    initMenus(g_clickVal, g_clickLimMax, g_menuIndex, alert1, alert2, alert3, alert4);
    return true;
  }
  else
  {
    return false;
  }
}

// create interface
void initLcd(String alert1 , String alert2 , String alert3, String alert4)
{
  // reset menu postion to time and date
  g_clickVal = 0;
  g_menuIndex = 0;
  g_clickLimMax = 3;
  // turn lcd on
  lcd.backlight();
  lcd.display();
  // set variable to track if user wnats to turn off the screen
  bool userPresence = true;
  // store last time user input something
  unsigned long lastInput = millis();
  // keep LCD turned as long as user has not turned it off and he was not
  // idle for too long
  while (userPresence == true && (millis() - lastInput) < g_lcdTimeOut)
  {
    bool input1 = checkEncoder(alert1, alert2, alert3, alert4);
    bool input2 = checkButton(&userPresence, alert1, alert2, alert3, alert4);
    if (input1 == true || input2 == true)
    {
      lastInput = millis();
    }
  }
  // turn lcd off
  lcd.clear();
  lcd.noBacklight();
  lcd.noDisplay();
}
```

### 16.1.10 textGenerators.cpp

```
#include "interface.h"

extern LiquidCrystal_I2C lcd;
// print given lines on lcd
void initText(Option option)
{
  int i;
  lcd.clear();
  for (i = 0; i < 4; i++)
  {
    lcd.setCursor(0, i);
    lcd.print(option.text[i]);
  }
}

// check if string is empty, if so, alter string
void isEmpty(String *subject, String confirmation)
{
  if (subject->length() == 0)
  {
    *subject = confirmation;
  }
}

String insertDash(int i)
{
    String text;
    int j;
    for(j=0;j<i;j++)
    {
        text +=String(F("-"));
    }
    return text;
}

String insertSpace(int i)
{
    String text;
    int j;
    for(j=0;j<i;j++)
    {
        text +=F(" ");
    }
    return text;
}

String createHeadline(String line)
{
    String text;
    int m,dashes1,dashes2;
    m = line.length();
    if ((m % 2)==0)
    {
        dashes2 = (18-m)/2;
        dashes1=dashes2;
    }
    else
    {
        dashes2 = (17-m)/2;
        dashes1=dashes2 + 1;
    }
    text = insertDash(dashes1) + line + insertDash(dashes2);
    return text;
}

String createLine(String line)
{
    String text;
    int m,spaces1,spaces2;
    m = line.length();
    if ((m % 2)==0)
    {
        spaces2 = (18-m)/2;
        spaces1=spaces2;
    }
    else
    {
        spaces2 = (17-m)/2;
        spaces1 = spaces2 + 1;
    }
    text = insertSpace(spaces1) + line + insertSpace(spaces2);
    return text;
}

String createPageNum(String line, int clickVal, int clickLimMax)
{
    String halfText, text;
    int m,dashes1,dashes2;
    halfText = String(F("[")) + line + String(F(" ")) + String(clickVal+1) + String(F("/")) + String(clickLimMax+1) + String(F("]"));
    m = halfText.length();
    if ((m % 2)==0)
    {
        dashes2 = (18-m)/2;
        dashes1=dashes2;
    }
    else
    {
        dashes2 = (17-m)/2;
        dashes1=dashes2 + 1;
    }
    text = insertDash(dashes1) + halfText + insertDash(dashes2);
    return text;
}

void templateLeft(String menuName, String line1, String line2, String pageName, int clickVal, int clickLimMax)
{
    Option option;
    String text0, text1, text2, text3;
    text0 = String(F("|")) + createHeadline(menuName) + String(F(">"));
    text1 = String(F("|")) + createLine(line1) + String(F(" "));
    text2 = String(F("|")) + createLine(line2) + String(F(" "));
    text3 = String(F("|")) + createPageNum(pageName,clickVal, clickLimMax) + String(F(">"));
    option.text[0] = text0;
    option.text[1] = text1;
    option.text[2] = text2;
    option.text[3] = text3;
    initText(option);
}

void templateRight(String menuName, String line1, String line2, String pageName, int clickVal, int clickLimMax)
{
    Option option;
    String text0, text1, text2, text3;
    text0 = String(F("<")) + createHeadline(menuName) + String(F("|"));
    text1 = String(F(" ")) + createLine(line1) + String(F("|"));
    text2 = String(F(" ")) + createLine(line2) + String(F("|"));
    text3 = String(F("<")) + createPageNum(pageName,clickVal, clickLimMax) + String(F("|"));
    option.text[0] = text0;
    option.text[1] = text1;
    option.text[2] = text2;
    option.text[3] = text3;
    initText(option);
}

```

```
void templateMiddle(String menuName, String line1, String line2, String pageName, int clickVal, int clickLimMax)
{
    Option option;
    String text0, text1, text2, text3;
    text0 = String(F("<")) + createHeadline(menuName) + String(F(">"));
    text1 = String(F(" ")) + createLine(line1) + String(F(" "));
    text2 = String(F(" ")) + createLine(line2) + String(F(" "));
    text3 = String(F("<")) + createPageNum(pageName,clickVal, clickLimMax) + String(F(">"));
    option.text[0] = text0;
    option.text[1] = text1;
    option.text[2] = text2;
    option.text[3] = text3;
    initText(option);
}

void templateChoice(String line1, String line2, String line3, String instruction)
{
    Option option;
    String text0, text1, text2, text3;
    text0 = String(F("|")) + createLine(line1) + String(F("|"));
    text1 = String(F("|")) + createLine(line2) + String(F("|"));
    text2 = String(F("|")) + createLine(line3) + String(F("|"));
    text3 = String(F("|")) + createLine(instruction) + String(F("|"));
    option.text[0] = text0;
    option.text[1] = text1;
    option.text[2] = text2;
    option.text[3] = text3;
    initText(option);
}
```

### 16.1.11 menuInitializers.cpp

```
#include "interface.h"

#include "interface.h"
#include "time.h"

// decide which menu should be displayed based on menuIndex
void initMenus(int clickVal, int clickLimMax, int menuIndex, String alert1 , String alert2 , String alert3, String alert4)
{
  switch (menuIndex)
  {
    case 0:
      initMainMenu(clickVal, clickLimMax);
      break;
    case 1:
      initAlertsMenu(clickVal, clickLimMax, alert1, alert2 , alert3, alert4);
      break;
    case 2:
      initSettingsMenu(clickVal, clickLimMax);
      break;
    case 3:
      initPowerMenu(clickVal, clickLimMax);
      break;
    case 22:
      initAutoPowerMenu(clickVal, clickLimMax);
      break;
  }
}

// decide which text should be displayd based on clickVal
void initMainMenu(int clickVal, int clickLimMax)
```

```
{
  switch (clickVal)
  {
    case 0:
      templateLeft(F("FarmID"), F("Time"), time_stamp(), F("Page"), clickVal, clickLimMax);
      break;
    case 1:
      templateMiddle(F("FarmID"), F("Alerts"), F(""), F("Page"), clickVal, clickLimMax);
      break;
    case 2:
      templateMiddle(F("FarmID"), F("Settings"), F(""), F("Page"), clickVal, clickLimMax);
      break;
    case 3:
      templateRight(F("FarmID"), F("Power"), F(""), F("Page"), clickVal, clickLimMax);
      break;
  }
}

void initAlertsMenu(int clickVal, int clickLimMax, String alert1 , String alert2 , String alert3, String alert4)
{
  switch (clickVal)
  {
    case 0:
      isEmpty(&alert1, F("None"));
      templateLeft(F("Alerts"), F("Dehydrated nodes:"), alert1, F("Page"), clickVal, clickLimMax);
      break;
    case 1:
      isEmpty(&alert2, F("None"));
      templateMiddle(F("Alerts"), F("Overwatered nodes:"), alert2, F("Page"), clickVal, clickLimMax);
      break;
    case 2:
      isEmpty(&alert3, F("None"));
      templateMiddle(F("Alerts"), F("Overheated nodes:"), alert3, F("Page"), clickVal, clickLimMax);
      break;
    case 3:
      isEmpty(&alert4, F("None"));
      templateMiddle(F("Alerts"), F("Cold nodes:"), alert4, F("Page"), clickVal, clickLimMax);
      break;
    case 4:
      templateRight(F("Alerts"), F("Back to main menu"), F(""), F("Page"), clickVal, clickLimMax);
      break;
  }
}

void initSettingsMenu(int clickVal, int clickLimMax)
{
  switch (clickVal)
  {
    case 0:
      templateLeft(F("Settings"), F("Date&Time"), F(""), F("Page"), clickVal, clickLimMax);
      break;
    case 1:
      templateMiddle(F("Settings"), F("Auto Power Off"), F(""), F("Page"), clickVal, clickLimMax);
      break;
    case 2:
      templateRight(F("Settings"), F("Back to Main Menu"), F(""), F("Page"), clickVal, clickLimMax);
      break;
  }
}

void initPowerMenu(int clickVal, int clickLimMax)
{
  switch (clickVal)
  {
    case 0:
      templateChoice(F("Do you want to"), F("quit interface?"), F("*Yes"), F("[Rotate to switch]"));
      break;
    case 1:
      templateChoice(F("Do you want to"), F("quit interface?"), F("*No "), F("[Rotate to switch]"));
      break;
  }
}

void initAutoPowerMenu(int clickVal, int clickLimMax)
{
  templateChoice(F("Set the time of "), F("LCD standby"), (String(clickVal)+F("/")+String(clickLimMax)+F(" min")), F("[Rotate to switch]"));
}
```

### 16.1.12 menuOutputs.cpp

```
#include "interface.h"
extern int8_t g_clickVal, g_clickLimMax, g_clickLimMin, g_menuIndex, g_lcdTimeOut;

// decide which output function should be called based on g_menuIndex
void menuOutput(bool *userPresence)
{
  *userPresence = true;
  switch (g_menuIndex)
  {
    case 0:
      mainMenuOutput();
      break;
    case 1:
      alertsMenuOutput();
      break;
    case 2:
      settingsMenuOutput();
      break;
    case 3:
      powerMenuOutput(userPresence);
      break;
    case 22:
      autoPowerMenuOutput();
      break;
  }
}
// decide which output should be give based on g_clickVal
void mainMenuOutput()
{
  switch (g_clickVal)
  {
    case 0:
      break;
    case 1:
      // switch to alerts menu
      g_menuIndex = 1;
      g_clickLimMax = 4;
      g_clickVal = 0;
      break;
    case 2:
      // switch to settings menu
      g_menuIndex = 2;
      g_clickLimMax = 2;
      g_clickVal = 0;
      break;
    case 3:
      // switch to power menu
      g_menuIndex = 3;
      g_clickLimMax = 1;
      g_clickVal = 0;
      break;
  }
}

void alertsMenuOutput()
{
  switch (g_clickVal)
  {
    case 4:
      // switch to main menu
      // page alerts
      g_menuIndex = 0;
      g_clickVal = 1;
      g_clickLimMax = 3;
  }
}

void settingsMenuOutput()
{
  switch (g_clickVal)
  {
    case 0:
      break;
    case 1:
      // switch to auto turn off menu
      g_menuIndex = 22;
      g_clickLimMax = 10;
      g_clickLimMin = 1;
      break;
    case 2:
      // switch back to main menu
      // setting page
      g_menuIndex = 0;
      g_clickLimMax = 3;
      g_clickVal = 2;
      break;
  }
}

void powerMenuOutput(bool *userPresence)
{
  switch (g_clickVal)
  {
    case 0:
      // turn lcd off
      *userPresence = false;
      g_clickVal = 0;
      g_clickLimMax = 3;
      break;
    case 1:
      // switch to main menu
```

```
        // page power
        g_clickLimMax = 3;
        g_clickVal = 3;
        break;
    }
    g_menuIndex = 0;
}

// change Lcd stanby time
// switch to settings menu
// page out power off
void autoPowerMenuOutput()
{
    g_lcdTimeOut = g_clickVal * 60000;
    g_menuIndex = 2;
    g_clickLimMax = 2;
    g_clickLimMin = 0;
    g_clickVal = 1;
}
```

## 16.2 Wireless node code

### 16.2.1 Slave.ino

```
using namespace std;
#include "defs.h";

#define SET_SENSORS 8

// set up a new serial object
SoftwareSerial comSerial (4, 5);

void setup() {
    pinMode(SET_SENSORS, OUTPUT);
    comSerial.begin(9600);
    //empty comSerial buffer
    while (comSerial.available() > 0){
        comSerial.read();

    }
}

void loop() {
    wait();
}
```

### 16.2.2 A2A_slave.cpp

```
#include "defs.h"

using namespace std;

extern SoftwareSerial comSerial;

#define SYSID 222
#define ADDRESS 1

#define FC_SENSOR 1
#define FC_CHECK 2
#define FC_PRESENCE 3

```

```

void wait(){
    /** Iterates until one or more bytes are detected in the serial buffer.
        It waits 80 milliseconds more to allow the complete message to be transmitted.
        It then transfers the whole content of the serial buffer to message and passes it on
        to check()*/

    if (comSerial.available() > 0){
        delay(80); //wait 80 milliseconds
        vector <uint8_t> message;
        //reads received transmission
        while (comSerial.available() > 0){
             message.push_back((uint8_t)comSerial.read());
        }
        check(message);
    }
}

void check(vector <uint8_t> & message){
    /** Checks the validity of the received message.
        If this node was meant to execute the function code,
        it erases the last two elements of message (checksum, function code)
        and adds the required data (depending on the function code,
        so either, its own address, the addresses of all
        nodes in range or a series of sensor readings*/

    uint8_t *message_p = &message[0];
    if (message[0] == SYSID && message[1] == ADDRESS && CRC8(message_p, message.size()) == 0 ){
        //result: node was meant
        if (ADDRESS == message[message.size()-3]){      //third last element in message: element preceding function_code and checksum
            switch (message[message.size()-2]){         //function code
                case FC_SENSOR:
                    message.pop_back();
                    message.pop_back();
                    reply(message);                     //this node is the last receiver (last address in pathway)
                    break;
                case FC_CHECK:
                    message.pop_back();
                    message.pop_back();
                    run_check(message);
                    break;
                case FC_PRESENCE:
                    message.pop_back();
                    message.pop_back();
                    acknowledge(message);
                    break;
            }
        }
        else{
            //it only erases the checksum here because it does not execute the function code
            message.pop_back();
            relay(message);
        }
        //add checksum
        uint8_t *message_p = &message[0];
        uint8_t checksum = CRC8(message_p, message.size());
        message.push_back(checksum);
        write_serial(message);
    }
```

```
}

void relay(vector <uint8_t> & message){
    /** It iterates the pathway of the receives message,
        starting at 3 (first address) until pathway length
        plus 2 (last address) until it finds its own address,
        and takes the next address after its own in the pathway
        and copies into the second element (message[1]) as
        the direct address of the message that is meant to be
        transmitted*/

    for (uint8_t i=3; i<message[2]+2; i++){
        if (message[i] == ADDRESS){
            message[1] = message[i+1];
            break;
        }
    }
    delay(10);
}

void reply(vector <uint8_t> & message){
    /** Copies the address preceding the last one (the address before
        its own address) into the second element and reverses the
        pathway, because the reply is going "backwards".
        Then it gets the sensor data (already converted and mapped to
        the full range of one byte) and adds it to the message*/

    message[1] = message[message.size()-2];
    reverse_elements(message, 3, message.size()-1);
    vector <uint8_t> data = readSensors();

    for (uint8_t i=0; i<data.size(); i++){
        message.push_back(data[i]);
    }
}

void run_check(vector <uint8_t> & message){
    /** Calls every possible address (except 0) under the assumption
        that the corresponding slave is in direct range. Afterwards it adds
        all addresses whose owners replied and acknowledged their presence and
        add them to the message*/

    vector <uint8_t> addresses;

    for (uint8_t i=1; i<256; i++){
        vector <uint8_t> m;
        m.push_back(SYSID);
        m.push_back(i);
        m.push_back(2);
        m.push_back(ADDRESS);
        m.push_back(i);
        m.push_back(FC_PRESENCE);
        m.push_back(CRC8(&m[0], m.size());

        write_serial(m);
        delay(200);
        if (comSerial.available() > 0){
            vector <uint8_t> reply;
            for (uint8_t k=0; k<comSerial.available(); k++){
                reply.push_back(comSerial.read());
            }
            uint8_t *reply_p = &reply[0];
            if (reply[0] == SYSID && reply[1] == ADDRESS && reply[2] == 2 && reply[3
] == i && reply[4] == ADDRESS && CRC8(reply_p, reply.size()) == 0){ //message is val
id
                addresses.push_back(i);
            }
        }
    }
    reverse_elements(message, 3, message.size()-1);
    message[1] = message[message.size()-1]; //the second address in the pathway (reversed order), referring to the last node before the targeted node

    //Add the addresses
    for (uint8_t i=0; i<addresses.size(); i++){
        message.push_back(addresses[i]);
    }
}

void acknowledge(vector <uint8_t> & message){
    /** Acknowledge its presence by returning its address.
        This is only necessary for the pathfinding phase.*/

    message[1] = message[message.size()-2];         //copy second last address in pathway in direct address
    reverse_elements(message, 3, message.size()-1);
    message.push_back(ADDRESS);                     //returns own address
    delay(10);
}

void reverse_elements(vector <uint8_t> & vec, uint8_t first_e, uint8_t last_e){
    /** Reverses the elements of a vector, utilizing a given start and end point*/

    uint8_t i, k;
    k = last_e;
    uint8_t temp;
    for (i=first_e; i<=last_e; i++){
        temp = vec[i];  //first element is saved
        vec[i] = vec[k];//last element is copied to the beginning
        vec[k] = temp;  //first element is copied to the end
        k--;
        if ((k < i) || (k = i)){
            break;
        }
    }
}

void write_serial(vector <uint8_t> & message){
    /** This function is just for purposes of convenience (and debugging)*/

    for (uint8_t i=0; i<message.size(); i++){
        comSerial.write(message[i]);
    }
}

//CRC-8 - based on the CRC8 formulas by Dallas/Maxim
//code released under the therms of the GNU GPL 3.0 license
byte CRC8(const byte *data, byte len) {
  byte crc = 0x00;
  while (len--) {
    byte extract = *data++;
    for (byte tempI = 8; tempI; tempI--) {
      byte sum = (crc ^ extract) & 0x01;
      crc >>= 1;
      if (sum) {
        crc ^= 0x8C;
      }
      extract >>= 1;
    }
  }
  return crc;
```

```
}
```

### 16.2.3 Sensor.cpp

```
#include "defs.h"
#include "math.h"
#define SET_SENSORS 8
#define CAPACITIVE_PIN A0
#define RESISTIVE_PIN A1

vector <uint8_t> readSensors(){
    /**The sensors are powered by the digital pin 8, defined as the
        constant SET_SENSORS. Pin 8 is set to a logical 1 before reading
        the sensors to reduce energy consumption. After seeting it to 1,
        the function waits 10 milliseconds so the sensors get some time to start.
        After reading the sensors, pin 8 is set to a logical 0.

        The communication protocols in the code for both master and slave
        allow for a varying number of transmitted sensor readings.
        If you want to add a sensor, please write:
            data.push_back(<conversion_function())
        below at //add sensor here. Please include a conversion
        function that converts the reading to a normal value and the map
        it from [minimum value, maximum value] to [0, 255].*/

    digitalWrite(SET_SENSORS, HIGH);
    delay(10);
    vector <uint8_t> data;
    data.push_back(getCapacitive());
    data.push_back(getResistive());
    //add sensor here
    digitalWrite(SET_SENSORS, LOW);
    return data;
}

uint8_t getCapacitive(){
    /** The formula for the concentration c was plotted in
        accordance with the results of the experiment with the
        capacitive sensor, conducted by Devrat and Sergio.
        The result is then mapped to the full range of one byte
        (0 to 255) in order to lose the least possible amount of
        presicion. This was done after we noticed that reading the
        analog signal sending them directly (as 8bit numbers) resulted
        in a range of 10 to 20 numbers (for the tested concentrations)*/

    double A = (double)analogRead(CAPACITIVE_PIN);
    double c = -0.00766*(A-518.3);
    c = map(c, 0, 1, 0, 255);
    c = limit(c, 0, 255);
}

uint8_t getResistive(){
    /** See the comment on the function above*/

    double A = (double)analogRead(RESISTIVE_PIN);
    double c = -0.1015*log(7.194*(767.949/A - 1.0));
    c = map(c, 0, 1, 0, 255);
    c = limit(c, 0, 255);
}

uint8_t limit(double c, uint8_t valMin, uint8_t valMax){
    /** Takes a number and limits it to [valMin, valMax], with both numbers
        8-bit integers. It is returned as an 8-bit integer*/

    if (c < valMin){
        c = valMin;
    }
    if (c > valMax){
        c = valMax;
    }
    return (uint8_t)c;
}
```

### 16.2.4 Defs.h

```
#include <ArduinoSTL.h>
#include <Arduino.h>
#include <stdint.h>
#include <SoftwareSerial.h>
using namespace std;

vector <uint8_t> readSensors();
uint8_t conductivityTest();
uint8_t getCapacitive();
uint8_t getResistive();
uint8_t limit(double c, uint8_t valMin, uint8_t valMax);

void wait();
void check(vector <uint8_t> & message);
void reply(vector <uint8_t> & message);
void relay(vector <uint8_t> & message);
void run_check(vector <uint8_t> & message);
void acknowledge(vector <uint8_t> & message);
void reverse_elements(vector <uint8_t> & arr, uint8_t first_e, uint8_t last_e);
void write_serial(vector <uint8_t> & message);
byte CRC8(const byte *data, byte len);
```

## 16.3 Timelapse code

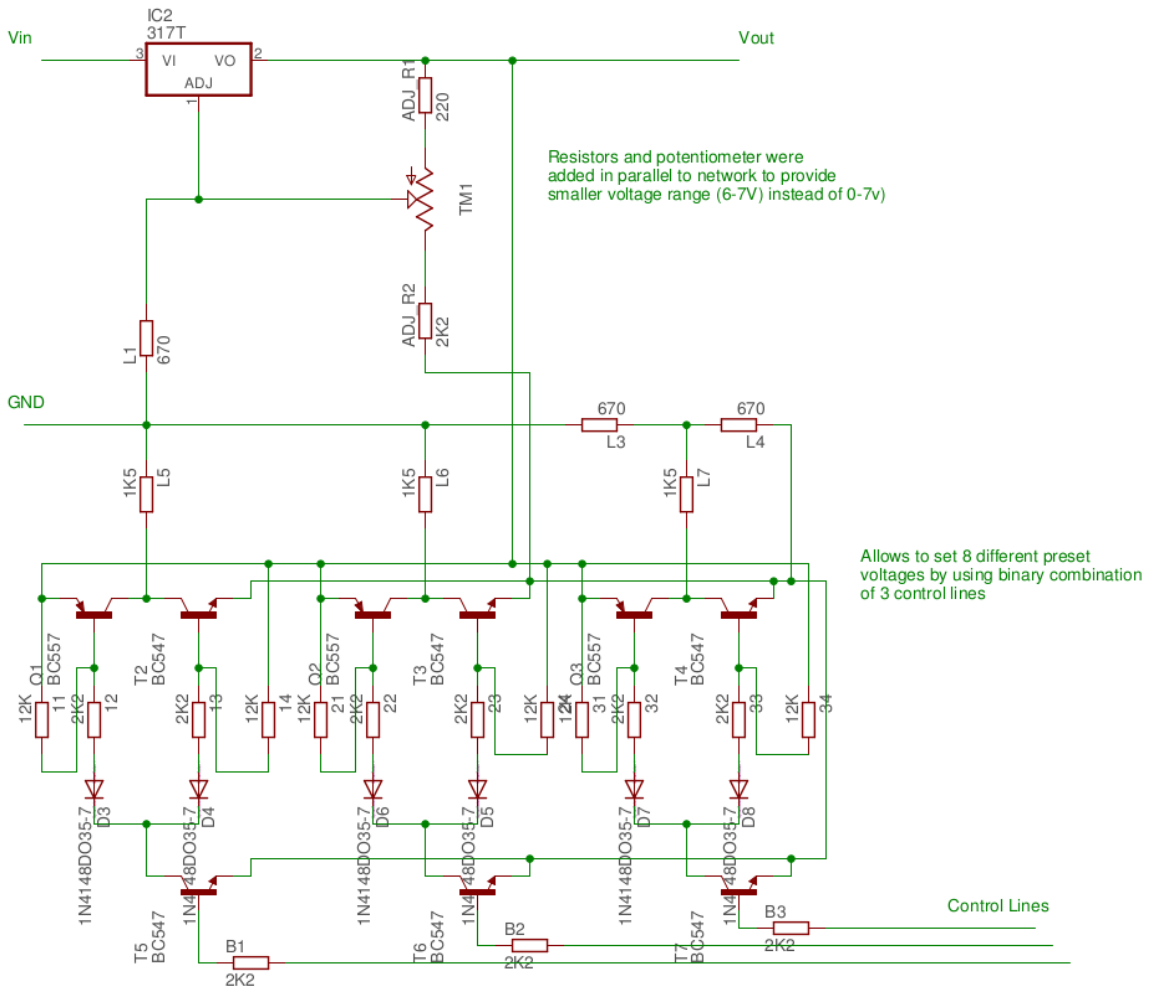
```python
 #Simple script to take photos at various times during the day and save it to different
#folders to allow us to create time lapse of growing plants. Due to time constrains
#it has been written to provide bare minimum functionality to fulfill this task

#Import required libraries
import os
import time
import datetime
#set flags to mark if photos have been taken today. Different numbers
#represent different times
flag1 = 1
flag2 = 1
flag3 = 1
#set starting number for photo numbering. in case program was interrupted
number1 = 0
number2 = 0
number3 = 0
#confirms that script has started
print('Picture taking script started!')
try:
    #infinite loop that can be stopped with keyboard interrupt(ctr+c)
    while True:
        #takes photo if it was not taken today at this time and saves it to
        #photos(1/2/3). Photos are numbered.
        if flag1 == 0 and datetime.datetime.now().hour == 9:
            os.system("raspistill -o photos1/%s.jpg"%(number1))
            flag1 = 1
            number1 = number1 + 1

        if flag2 == 0 and datetime.datetime.now().hour == 12:
            os.system("raspistill -o photos2/%s.jpg"%(number2))
            flag2 = 1
            number2 = number2 + 1

        if flag3 == 0 and datetime.datetime.now().hour == 15:
            os.system("raspistill -o photos3/%s.jpg"%(number3))
            flag3 = 1
            number3 = number3 + 1
        #flags are reset at the end of the day so loop can continue to take photos
        if datetime.datetime.now().hour  == 23:
            flag1 = 0
            flag2 = 0
            flag3 = 0
        #short delay to conserve power when photos are not taken
        else:
            time.sleep(1)
#terminates script when ctr+c is pressed
except KeyboardInterrupt:
   print('Picture taking script terminated!')
```

## 16.4 Resistor network


Vin
IC2
317T
VI
VO
ADJ
Vout
ADJ_R1
220
TM1
Resistors and potentiometer were added in parallel to network to provide smaller voltage range (6-7V) instead of 0-7v)
ADJ_R2
2K2
L1
670
GND
670
L3
670
L4
1K5
L5
1K5
L6
1K5
L7
Allows to set 8 different preset voltages by using binary combination of 3 control lines
Q1
BC557
T2
BC547
Q2
BC557
T3
BC547
Q3
BC557
T4
BC547
1N4148DO35-7
D3
D4
D6
D5
D7
D8
T5
BC547
T6
BC547
T7
BC547
B1
2K2
B2
2K2
B3
2K2
Control Lines

**Appendix 1 Resistor network**

## 16.5 Data logging code

```
//pin definitions 1st setup
const int cap_sens1 = A0;
const int res_sens1 = A1;
const int temp_sens1 = A8;
// pin definitions 2nd setp
const int cap_sens2 = A3;
const int res_sens2 = A4;
const int temp_sens2 = A9;
//chipSelect for SD card
const int chipSelect = 53;
//flags
bool scheulde_flag = false;
//unclude libraries
#include <SPI.h>
#include <SD.h>
#include <TimeLib.h>

```

```

// functions
void log_data();
void print_values(int value_cap1,int value_res1,double value_temp1,int value_cap2,int value_res2,double value_temp2);
void write_SD(int value_cap1,int value_res1,double value_temp1 ,int value_cap2,int value_res2,double value_temp2);
int average_sensor_value(int sensor, int reading);
double average_temp_value(int temp_pin, int readings);

void setup() {
// set time every time Arduino is restarted
  setTime(00,00,00,14,12,2017); //hour, min, sec, day, month, year

// Open serial communications and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect.
  }
  Serial.print("SD card setup wait\n");
  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect)) {
    Serial.println("Something is wrong!");
    while (1);
  }
  Serial.println("SD card OK !.\n");
}

void loop() {
  delay(100);
  //below function logs data every 10 seconds
  if (second() %10 == 0 && scheulde_flag==false){
    log_data();
    scheulde_flag=true;
  }
  if (second() %10 != 0){
    scheulde_flag=false;
  }
}
//data gathering and logging function
void log_data(){
  //stores output from averaging functions in variables
  int value_cap1 = average_sensor_value(cap_sens1, 10);
  int value_cap2 = average_sensor_value(cap_sens2, 10);
  int value_res1 = average_sensor_value(res_sens1, 10);
  int value_res2 = average_sensor_value(res_sens2, 10);
  int value_temp1 = average_temp_value(temp_sens1, 10);
  int value_temp2 = average_temp_value(temp_sens2, 10);
  //print values to serial
  print_values(value_cap1, value_res1, value_temp1, value_cap2, value_res2, value_temp2);
  //write values to SD card
  write_SD(value_cap1, value_res1, value_temp1, value_cap2, value_res2, value_temp2);
}
//write to sd card
void write_SD(int value_cap1,int value_res1,double value_temp1 ,int value_cap2,int value_res2,double value_temp2){
  //open datalog.txt file
  File dataFile = SD.open("datalog.txt", FILE_WRITE);
  //write to file
  if (dataFile) {
    //prints line of data to file
    dataFile.print(day());
    dataFile.print(", ");
    dataFile.print(month());
    dataFile.print(", ");
    dataFile.print(hour());
    dataFile.print(", ");
    if(minute()<10)
      dataFile.print('0');
    dataFile.print(minute());
    dataFile.print(", ");
    if(second()<10)
      dataFile.print('0');
    dataFile.print(second());
    dataFile.print(", Plant 1, ");
    dataFile.print(value_cap1);
    dataFile.print(", ");
    dataFile.print(value_res1);
    dataFile.print(", ");
    dataFile.print(value_temp1);
    dataFile.print(", Plant 2, ");
    dataFile.print(value_cap2);
    dataFile.print(", ");
    dataFile.print(value_res2);
    dataFile.print(", ");
    dataFile.print(value_temp2);
    dataFile.println(" ");
    // closes file. IMPORTANT!
    dataFile.close();
}
  // error if something goes wrong
  else {
    Serial.println("cannot open file");
  }
}
//function that prints values to serial
void print_values(int value_cap1,int value_res1,double value_temp1,int value_cap2,int value_res2,double value_temp2){
  Serial.print(hour());
  printDigits(minute());
  printDigits(second());
  Serial.print(" // ");
  Serial.print("cap1 = ");
  Serial.print(value_cap1);
  Serial.print(", ");
  Serial.print("res1 = ");
  Serial.print(value_res1);
  Serial.print(", ");
  Serial.print("temp1 = ");
  Serial.print(value_temp1);
  Serial.print(", // ");
  Serial.print("cap2 = ");
  Serial.print(value_cap2);
  Serial.print(", ");
  Serial.print("res2 = ");
  Serial.print(value_res2);
  Serial.print(", ");
  Serial.print("temp2 = ");
  Serial.print(value_temp2);
  Serial.print("\n");
}

//function that reads sensor multiple times and averages the value
int average_sensor_value(int sensor, int readings){
  int temp =0;
  int i;
  delay(50);
  for (i = 0; i < readings; i++){
    temp = temp + analogRead(sensor);
    delay(50);
  }
  return temp/readings;
```

```
}

//function that reads temperature sensor multiple times and averages value
//value is then converted to degree C
double average_temp_value(int temp_pin, int readings)
{
  const double invBeta = 1.00 / 3380.00;   // replace "Beta" with beta of thermistor
  const  double adcMax = 1023.00;
  const double invT0 = 1.00 / 298.15;   // room temp in Kelvin
  double temp, K, C;
  int i;
  temp = 0;
  for (i = 0; i < readings; i++)
   {
    temp = temp + analogRead(temp_pin);
    delay(100);
   }
  temp = temp/readings;
  K = 1.00 / (invT0 + invBeta*(log ( adcMax / (double) temp - 1.00)));
  C = K - 273.15;                      // convert to Celsius
  return C;
}

void printDigits(int digits){
  // utility function for digital clock display: prints preceding colon and leading 0
  Serial.print(":");
  if(digits < 10)
    Serial.print('0');
  Serial.print(digits);
}
```