FLORIVATE

JACO COETSEE
GRADE 11
REPORT FILE

Index:

Engineering Design Process:

Need for system:

Currently in South Africa there is a huge drought, and this puts pressure on our water reserves. If water is continually misused and wasted it can mean a dire situation for us as a country and the sector where most of the water is wasted is irrigation. So, a universal device that can minimalize wastage would mean huge relief on water reserves and can prevent similar situations from prevailing in other parts of the world.

Project:

This project will be an improvement on a project I made last year, the iGarden. Yet last year's project had many faults including unreliable programming, not being user friendly and a generally unappealing appearance. This year I hope to improve on all those categories. Thus, the project will consist of a device that uses modular sensors that can be removed and added on to collect data about the environment. This data will be used to determine the need for irrigation in a garden and if all the requirements are met it will activate the sprayers. All these systems will work together to prioritise the minimal use of water whilst also taking away all responsibility from the user. This project will also prioritise low-cost as to make it accessible to everyone and not just people of moderate to high wealth.

User:

The device is meant for any consumer with a garden or land in need of irrigation. Specifically meant for people who want to upkeep their gardens with little effort whilst also contributing towards water preservation. It also targets a user that wants a low-cost system that won't break their bank.

Function:

To enable users to passively save water via installing this once off system. It only needs to be setup and can be modified or replaced at any time. It will take over the duties of irrigation and determine the amount of water used and provide feedback to the user. Thus, it marries functionality and data collection for the ultimate residential irrigation system that will improve day to day life in one way or the other at the lowest cost possible.

Background Study:

Proof of need for water preservation in South Africa:

South Africa is currently going through a massive water crisis, this is because of but not limited to the following reasons:

- Very dry rainy seasons: the west coast of South Africa has experienced to dry winter seasons which is said to be so rare that it only happens once a millennium (https://qz.com/, 2017). Because of this Cape Town and various other cities cannot rely on natural ways to relief them of the crisis.

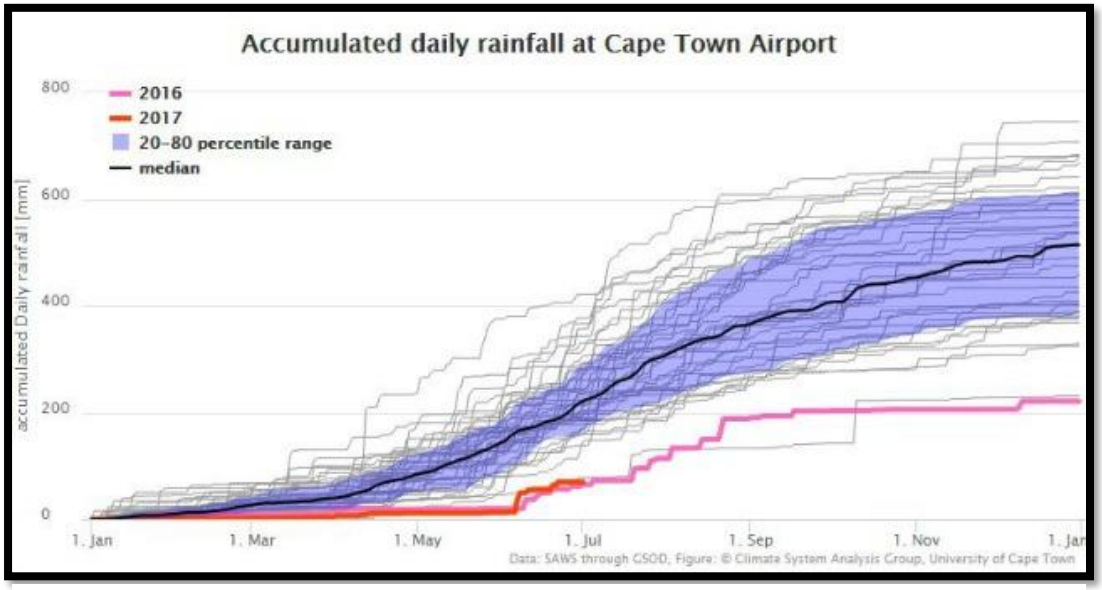


Figure 1: Daily Rainfall at Cape Town airport for 2 years

This graph (Figure 1) (https://albertonrecord.co.za/, 2017) clearly indicates these low rainfall numbers in 2016 and 2017.

- Overpopulation: The population of Cape town has grown with a margin of 55% (/www.news24.com/, 2017) since 1995 while dam storage has only increased with about 15%. This means that the population is consuming more water than can be stored and this also plays a role in the decline of water availability in South Africa.
- Global warming: Due to the increase of carbon dioxide in the air it has a very big impact on ecosystems in South Africa. One of the impacts it has is on temperature and if temperature rises so does pressure. Which means that this drought is likely a product of these changes (/www.news24.com/, 2017).
- The uneven distribution of water in South Africa: Because of the vastness of climates in South Africa it is a very diverse country in that view, but this can also be its downfall in the sense of the uneven distribution of rainfall in South Africa. When water evaporates in one area the wind carries the clouds it forms to another area. This means that water is being constantly lost from one area and only benefitting the next area. (www.ft.com, 2016).

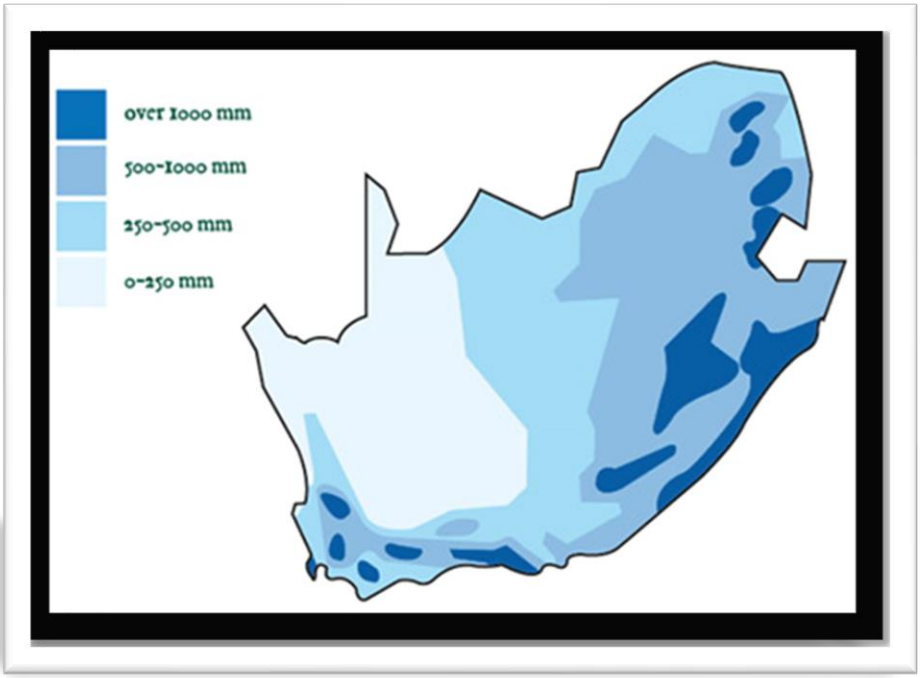


Figure 2: Rain Fall distribution Graph

In Figure 2 the distribution of rainfall can clearly be perceived. Effecting the Northern Cape and parts of the West Coast drastically.

So, water preservation needs to be prioritised to delay the day when we have no water left. To prove a need for a smart irrigation system the expenditure of water was inspected further.

Under the various categories of water uses irrigation uses the most of all. Irrigation is needed for agriculture when growing crops for food security. It is used in the upkeep of massive municipal gardens and corporation grounds. Thus, if a smart system were to be installed it would drastically reduce the pressure that irrigation puts on the water crisis.

Table 1: Water usage on Industrial Scale

Sector	**%**
Irrigation	59.0%
Urban use	25.1%
Mining and Bulk Industrial use	5.7%
Rural use	4.3%
Afforestation	3.7%
Power Generation	2.2%

As seen in table 1: irrigation accounts for more than 60% of water usage (businesstech.co.za, 2016).

From all the water used in our country about 12% is used by residents and for purely domestic use. Yet even though a smaller amount is used by residents it is once again shocking to see how much of it is dedicated to irrigation.

If looking at the table 2 below, the expenditure in houses without gardens is largely based around toilet expenditure but progress has already been made reducing that amount with things like toilet bricks.

Table 2: Domestic Use of Water

Domestic Use	Rural	Urban
Toilets	73%	37%
Bath and Shower	19%	32%
Washing Machine	N/A	17%
Other (Cooking, Cleaning, Washing Dishes, Drinking, etc)	8%	14%

Yet if we look at houses with gardens (Table 3) show that irrigation takes the largest expenditure. Yet no effort has been made to make gardening more water efficient thus the need for an irrigation that prioritises that is crucial.

Table 3: Use of water with Homes with Water

Homes with Gardens	%
Gardening	46%
Other (See above)	54%

All tables sourced from (businesstech.co.za, 2016)

Idea originality:

This idea has already been executed by me in 2017. This is when I created the first prototype of the gardening system and worked out the main principles of the project. Yet all the ideas were very rough and the coding and hardware extremely unreliable with more bugs than variable names. This project also had the major flaw of requiring trained personal to install and repair system while the new project will require this only in severe cases. Therefore, this year I will improve on last year's mistakes and expand on the idea itself. And because it is my idea it is in my rights to "reuse" it.

Furthermore, research was conducted to see if similar systems already exists. Firstly, a DIY project was found (www.electronicshub.org, 2015) that shows a primitive version of my idea that was quickly executed out of need for the system. This specific system also lacks the needed sensors to ensure that water is not wasted in the process of irrigation. For example, the lack of rain sensors will cause an over expenditure in water due to irrigation taking place while or directly after it rains. The programming surrounding the system is also very primitive since it doesn't dissect water usage and times of irrigation to be able to monitor the effectivity of the system.

When looking at commercial projects a similar system was found called the RainMachine Touch HD-12 (www.rainmachine.com, 2018). This system is flawed in the sense that it doesn't use the direct environment to determine irrigation time, but it uses predictions from the internet to determine irrigation times. This means that if predictions are wrong it will lead to waste of water and if the direct environment was change (e.g. an obstruction prevented the water being sprayed on lawn) the

system won't be aware of it and keep spraying the water again and again essentially wasting it. Whereas my system will be able to detect this and eliminate the possibility of wastage.

The RainMachine (Figure 3) is also completely dependent on WIFI and will not operate properly when it is not connected. This is not ideal for people with lower income who cannot afford WIFI or the have a limited amount of data each month that must be used for other necessities. My system will be different in the sense that it marries technology with nature and isn't dependant on being connected to the web.

Figure 3: A Photo of the Rain machine

Finally, after a review of various other products on the market and their different functions a shocking conclusion was made. Besides only having limited functionality in comparison with my design it is extremely expensive. Just a bare bones controller without any sensors or added piping like the RainMachine equates to $239 or approximately 3000 rand (www.rainmachine.com, 2018). This is a very unrealistic amount for the average South African citizen and is the reason none of these systems have become mainstream and has aided in the saving of water in our country. My design will aim to be as cheap as possible while giving the best result and least consumption of water.

3D Printing Background Information:

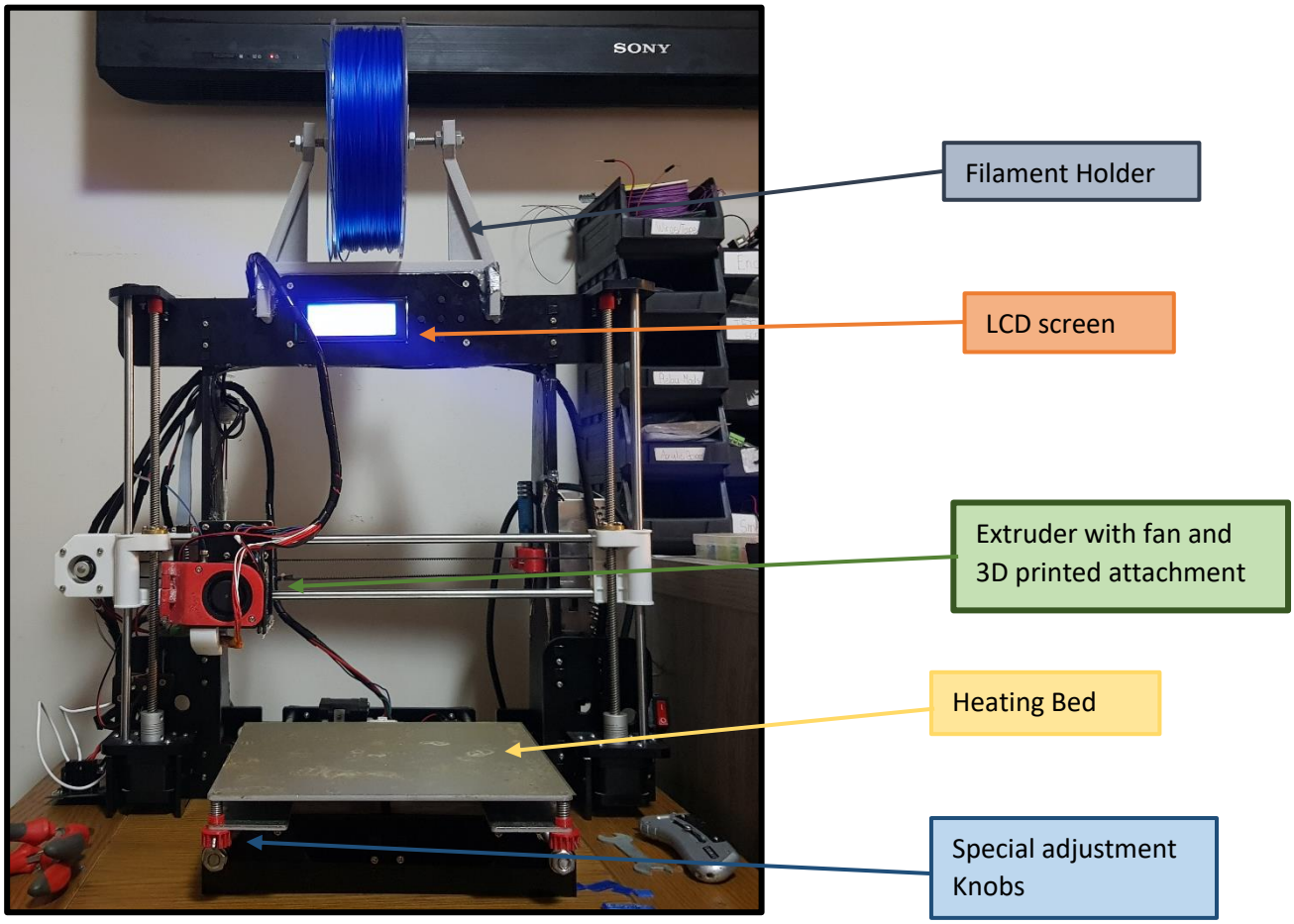
For this project many of the housing schemes for the sensors will be created using 3D printing. This will enable the sensors protection against the elements. The main hub will also be constructed using 3D printing going for an aesthetic look to make it more appealing for the user. The 3D printer I will be using is the Anet A8, which was sourced from Banggood.com a couple of years ago. Since then many improvements have been added (rather printed) onto it so it isn't quite the same as the stock printer. Therefore, if any of the design are to be recreated with another printer the accuracy will not be guaranteed. The printer settings are determined by the code it is fed therefore the only setting of worth on the printer to be noted is when the levelling is considered. Before any print the level is tested across the whole surface to ensure it is equal.

The following adjustments were made to the printer:

- A holder for the filament roll so it doesn't get caught up between other wires or itself. It also keeps the tension in the filament the same, so it doesn't snap or loosen.
- A structure was printed that holds the fan that keeps the extruder cool. This structure can fold out revealing the internals of the extruder making it more accessible when filament is being swapped for example.

- Special knobs were printed that attach to the screws on the heating bed. This makes it easy to adjust and level before a print.
- Lastly a tool holder was printed that attaches to inside of the printer. This doesn't really affect the print quality but makes it easier to operate the printer.

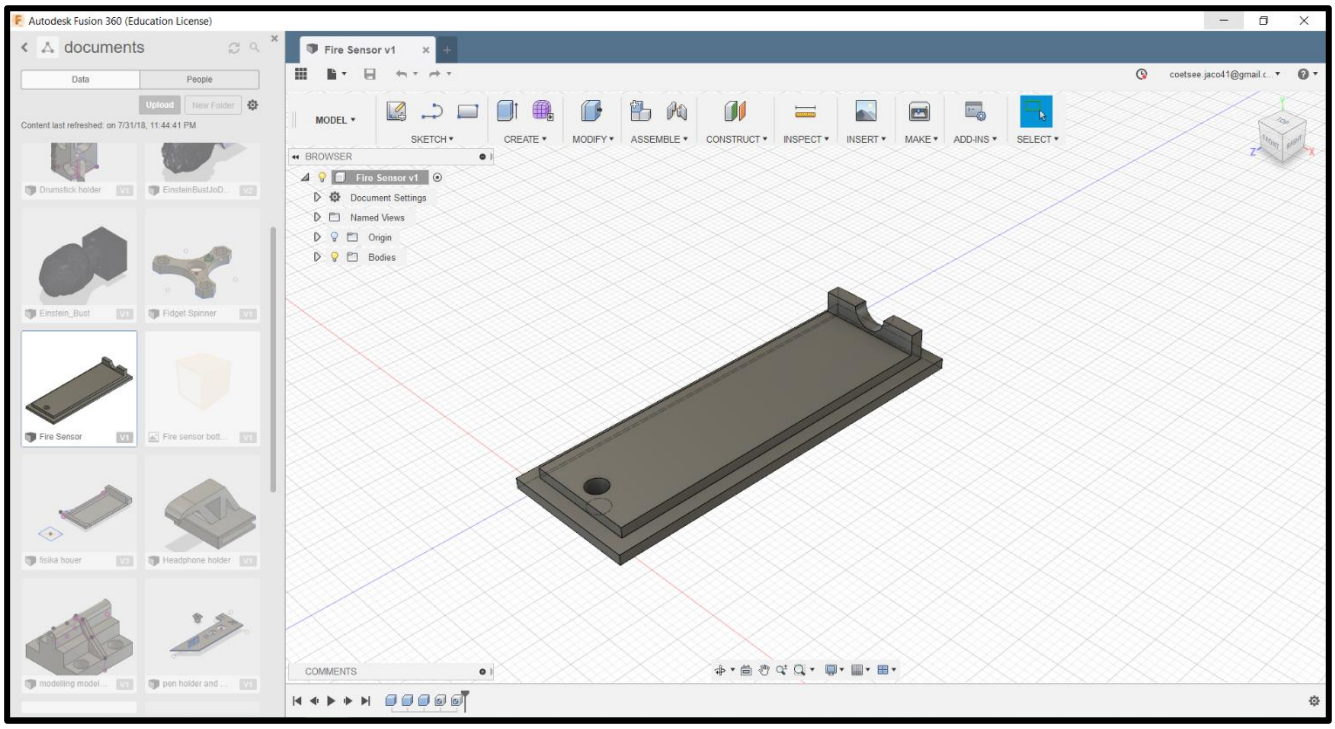
3D printer:



Fusion360:

When making a 3D model to print I first design it in Autodesk's CAD (Computer Aided Drawing) program called Fusion360. It is specially designed for a simple interface that allows for the quick builds of functional 3D models. It also allows me to convert 3D models into STL (Standard Triangle Language) files that enable it to be interpreted by 3D slicing software. All the models included in this project was designed with Fusion360.

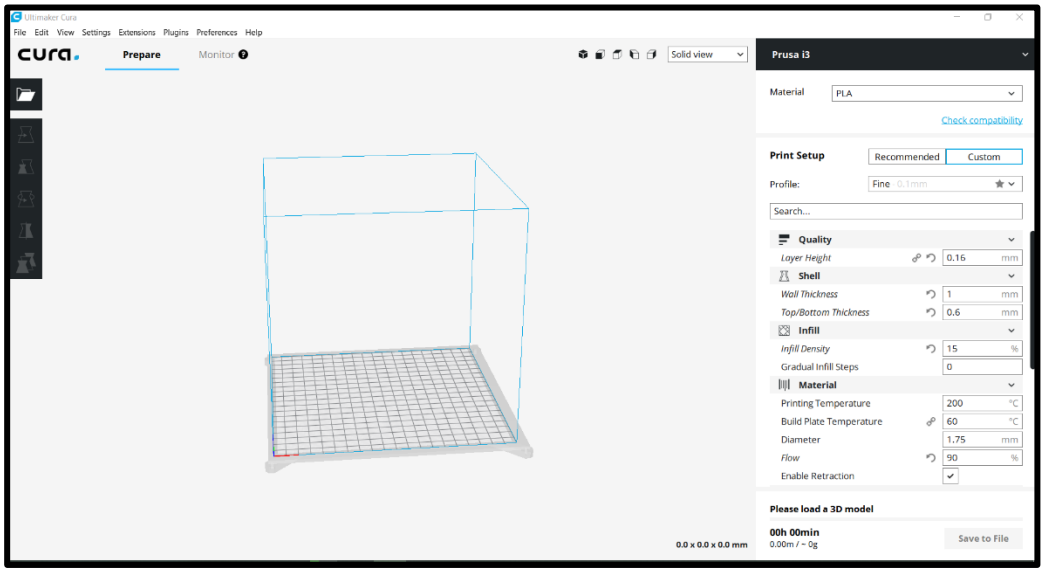
Fusion360 screenshot:



Ultimaker Cura:

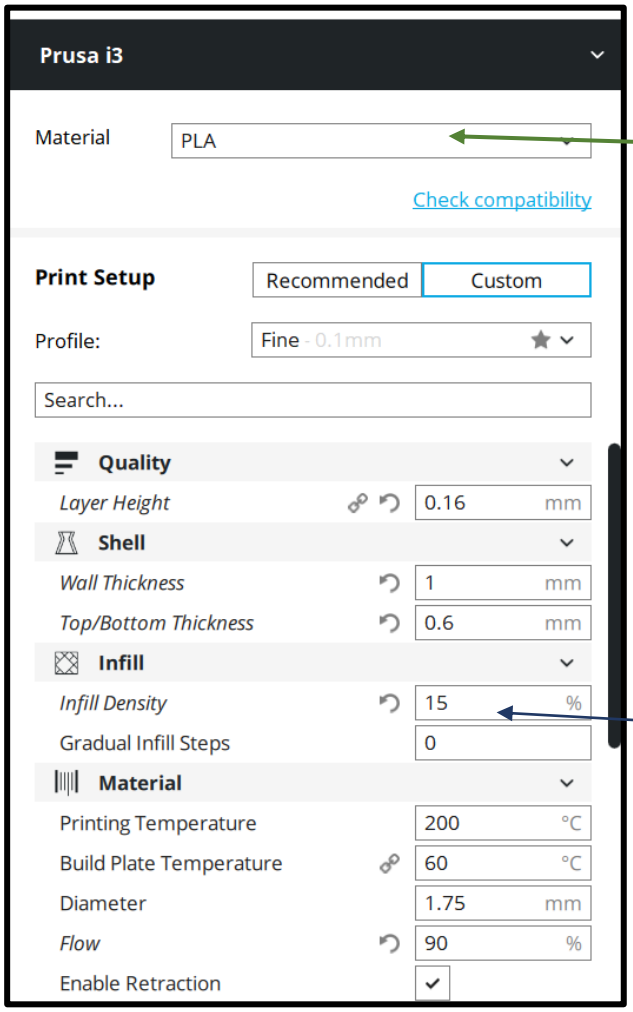
Cura is a 3D slicing software that takes the STL files that Fusion360 produces and interprets them. It generates the 3D model that will be printed by the 3D printer and allows the user to change different properties for various print results. It then produces the Gcode after the settings are finalised. This Gcode contains all the necessary information that the 3D printer needs to successfully and accurately make the desired model. It controls the axis's and the flow of the filament as well as the heat of the nozzle and the bed. Therefore, it is essential to the 3D printing process.

Ultimaker Cura Screenshot:



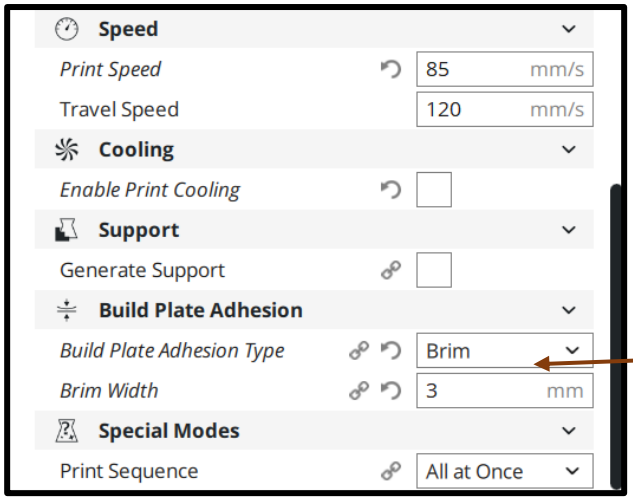
Cura Settings:

To print out decent quality models Cura must be configured very specifically and there are some very tedious settings. Note that the settings discussed below are not precisely the same for each print, but it will mostly be similar. The settings are as follows:



Material of choice: Polylactic Acid (Mostly Transparent Red or Blue)

Determines how much internal support will be generated in open space



Generates a border around the 3D Model to make it stick better to bed surface

Note: Settings that seemed obvious weren't further explained

Research on components to be used:

Arduino Mega 2560:

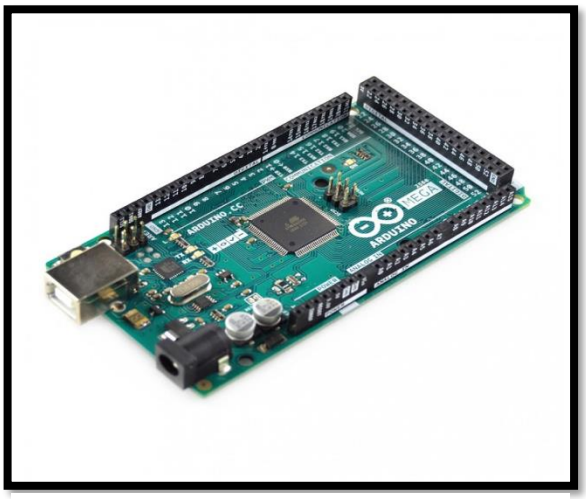


Figure 4: Arduino Mega

The MEGA (Figure 4) is a microcontroller that can control various outputs while simultaneously receiving inputs either digitally or through analogue pins. This specific microcontroller has 54 digital input/output pins and 16 analogue inputs that allow it to control a lot of devices and sensors which is a necessity with the project I am pursuing. It also has a bigger memory capacity and ram amount than the usual Arduino Uno allowing for much more functionality when coding the system. (www.arduino.cc, 2017)

Image sourced from (www.robotshop.com, 2018).

Last year I used the Arduino Uno and was limited by the amount I could do; thus, this is the logical progression.

12V/5V Power supply:

Figure 5: Power Supply

To power the system a beefy power supply is needed that will be able to accommodate all the modules and relays that will be used in the system. This power supply (Figure 5) from Hobbytronics (www.hobbytronics.co.za, 2018) is ideal since it can handle up to 2A which is more than enough for the system in mind. This will be upgraded if necessary, but as stated earlier low cost is prioritised in the conceptualisation of this product.

Image sourced from (www.hobbytronics.co.za, 2018).

Liquid Crystal Display 16 by 2 with I2C support:

Figure 6: LCD display

Image and information courtesy of (www.geeetech.com, 2017).

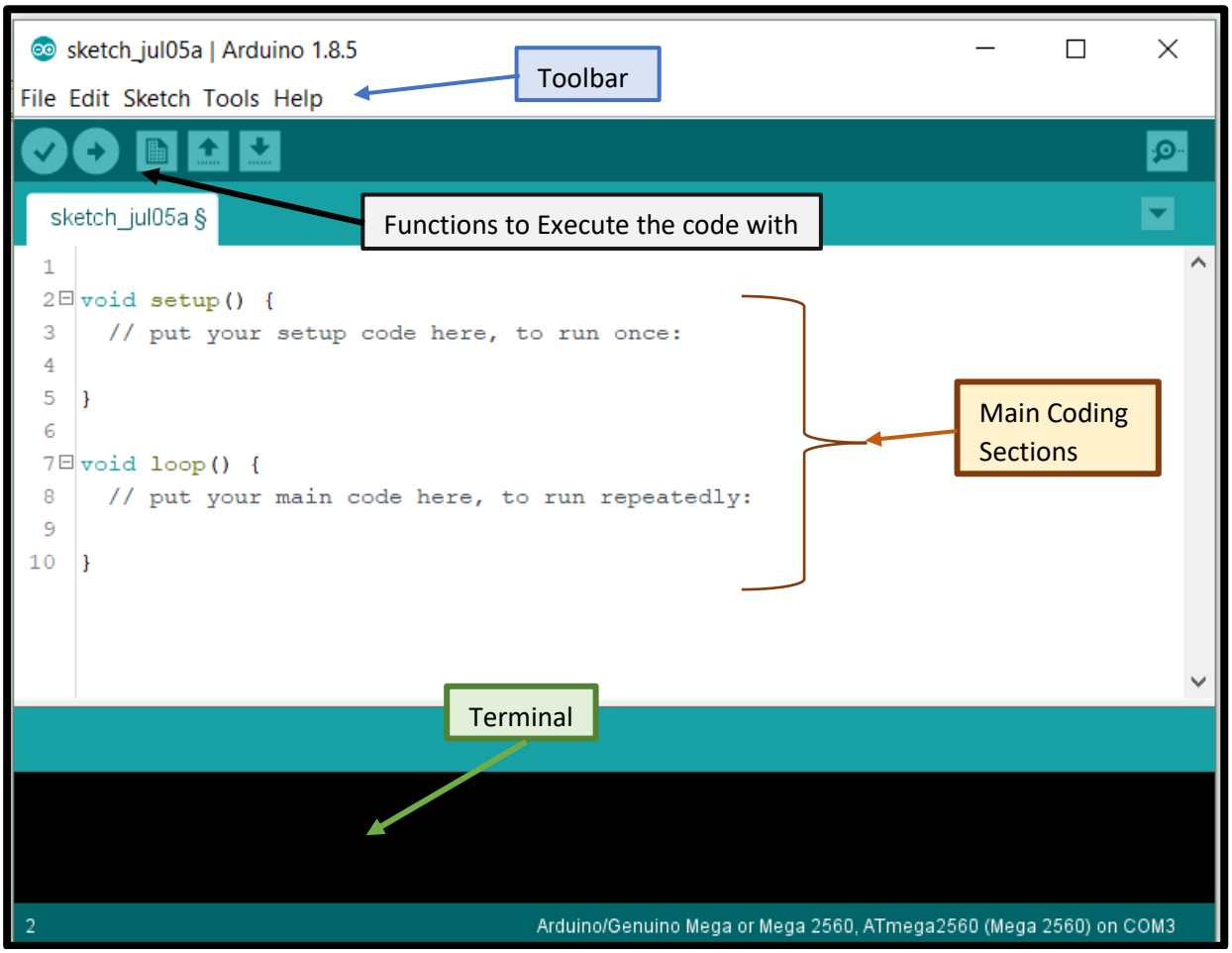
This module (Figure 6) allows the economic use of wires to maximise the number of sensors that can be used. It does via I2C technology where data is transmitted using only two wires. The actual functioning of the module consists of 2 rows of 16 columns each. This allows for a maximum number of 32 characters that will be used to display basic info about the system.

If this module doesn't serve its purpose well enough it will be replaced by a module of larger character capacity. But still prioritising low-cost it will be integrated first and evaluated for effectivity before it is ruled out.

Arduino IDE:

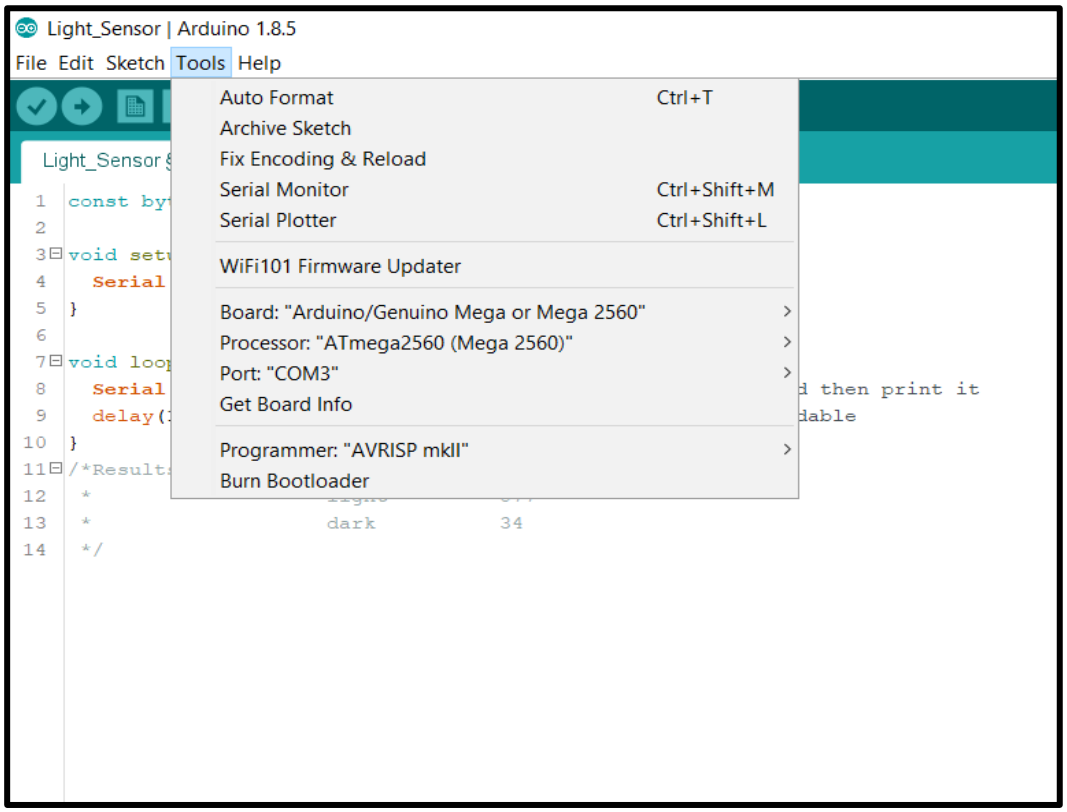
The Arduino microcontrollers are programmed using a form of C# with all the same characteristics as this language. This code is loaded onto the microcontroller via a Serial cable and software found on the Arduino IDE.

The base IDE looks as follows:



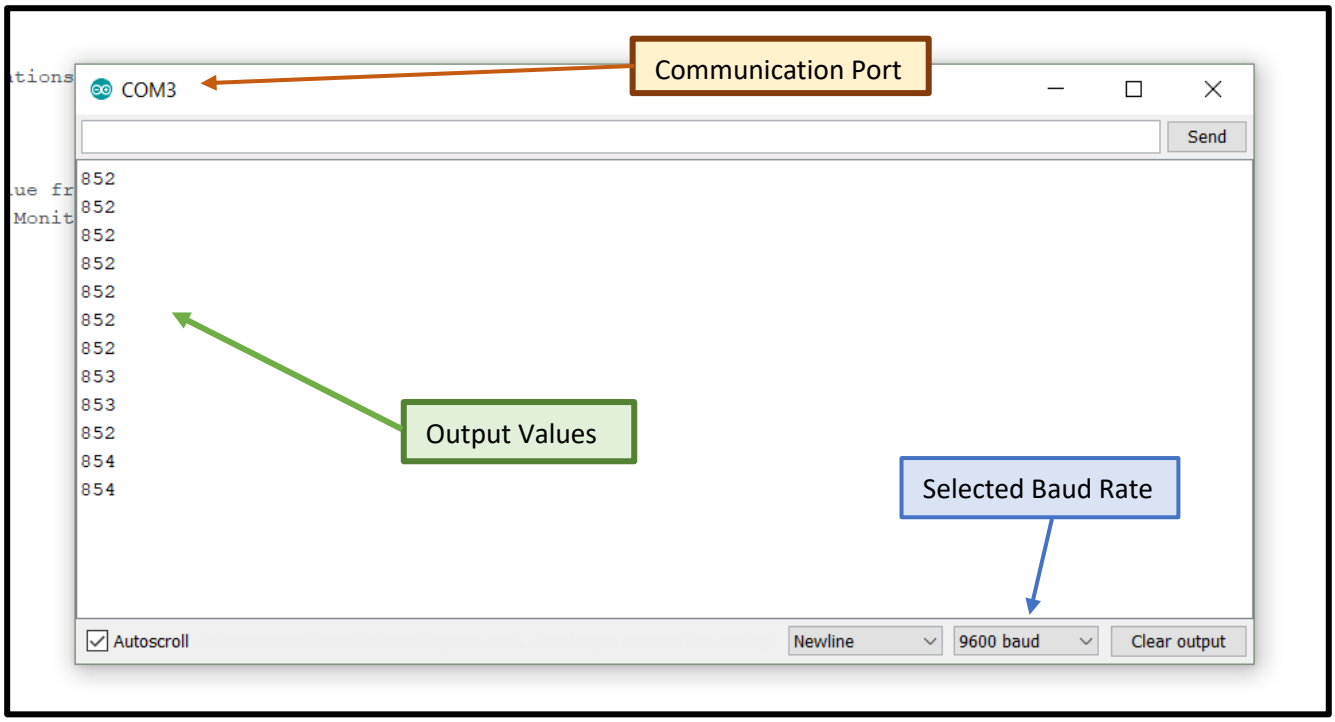
So, it consists of `void setup()` which is contains all the initial code used to setup all the devices and is only meant to run once. The following section is `void loop()` which is the main code which will run repeatedly until power is disconnected. This will be where all the sensor data is retrieved and according actions are made.

The following settings were used to compile the code:



The selected board is the “Arduino/Genuino Mega or Mega 2560”, which means that it will compile the code for the specifications of the Arduino mega. The processor this specific board uses is the “ATmega2560” so this reflects accordingly on the settings. The communication which is selected to communicated with is Port COM3 but can vary if another USB port is used. Lastly the programmer used is “AVRISP mkII”.

The Serial Monitor of the IDE looks as follows:



Engineering Goal:

The goal is to design and construct a smart irrigation system that automates the duties of garden owners who wish to decrease their water expenditure and contribute to preserving our environment.

Description:

The system will be an irrigation system controller of modular design that is automated in most aspects of its functioning and provides feedback on use. It will consist of various sensors that are replaceable and expandable, and the system will adjust accordingly. Thus, the coding allows for modularity as well. It will also have an interface for the user to receive and interpret data accordingly and perceive if water is saved or used sparingly. With the sensors and interface, it will also have relays to control the valves that enable the sprayers to irrigate.

Criteria and Constraints:

Size: The total area that the system spans will be dependent on the size of the garden it is situated on and therefor that factor is variable. Yet the size of the central hub, where all the data is received, processed and outputted, will be compact and very easily transportable. Yet it can still vary since different types and amounts of sensors can be added onto the system. It shouldn't be any larger than a cash register drawer, yet it can be smaller than this.

Appearance: The wires and sensors should be hidden from sight and run mostly underground as to not interfere with garden aesthetics of the garden or daily operations of the garden (e.g. lawn mowing or bush trimming). The central hub must also be of an ergonomic design as to not intimidate user. It should also consist of a way to communicate data to the user that is both intuitive and beautiful. The design should also be centred around blending in with the environment and looking like it is a part of the surrounding nature.

Physical Features: The system will be split into two main parts, namely: the central hub and sensors. The central hub will be portable and can be transported from site to site, yet the sensors will have to be detached and reattached when the transportation takes place. This central hub will be encased and protected with a 3D printed body thus, it will be waterproof and drop-proof yet large amount of static energies (e.g. lightning) will cause the system to be destroyed so it must be guarded from that. The sensor part of the system will vary since different amounts of sensors can be used at any given time. It will connect to the central hub through wiring and encased in a unique holder according to its function e.g. the rain sensor will have an encasement that allows it to optimally catch rain water and drain and not retain the water at the same time.

Inputs: The system will prioritise low consumption of energy as not to be counter intuitive with the initiative of saving water (saving one resource at the expense of another). The inputs that connect to this system must be of optimal accuracy and information carried from the sensor to the central hub should not be affected by any factors that could lead to inaccuracy.

Outputs: The system should keep a constant logic level (5V) as to not confuse data athet any stage. The power supply should output this, so the relays can be consistently switched on and off for this purpose. It should also display information on the interface given.

Manufacturing: Prioritising low cost it has been concluded that prices are considerably lower if it is imported from overseas and with that a lot more products are available for selection. Does components of the system will be imported and ordered online. The only drawback is that it might take longer for the parts to be received thus, it should be ordered with careful planning. The encasements will be designed, and 3D printed as to achieve an aesthetic and ergonomic feel to the design. The wiring will be sourced locally and used accordingly. The programming for this will be loaded onto the mega chip beforehand so no programming skills are required of the end-user.

Environmental: It should be able to withstand the average temperature range from -10 degrees Celsius to 35 degrees Celsius which is where the end-user will be most likely located. The central hub should be well protected with materials that prevent corrosion especially for coastal areas. The sensors should be encased in a way that protects them from the elements e.g. sun light or water.

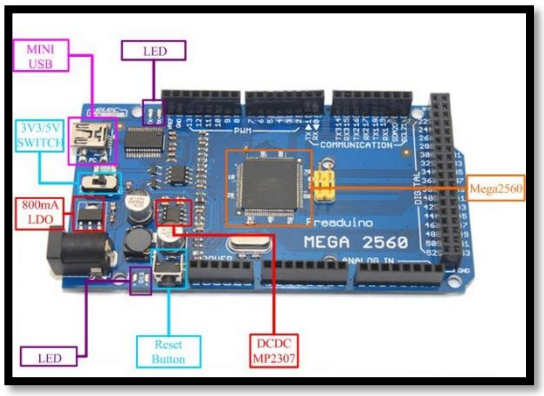
User Requirement: The system will be of extremely ergonomic design. The users will be only required to plug in the needed sensors and activate them via a switch and the system will do the rest of the work. The modularity will also allow broken sensors to be replaced by simply removing them out and plugging in a new one. This will make installation and repairs very simple and, with the help of a brochure and instructions, it will be able to be done by anyone. Though if there are problems with the main system then it should either be completely replaced or looked at by a knowledgeable person on the subject.

Lifespan: Because of the modular design a broken part doesn't mean the whole system needs to be replaced a new sensor can simply be bought and reattached to system. This means that theoretically system failure would only need a replacement of a single part and the system would be fine again. Thus, concluding that this system, with proper maintenance and care, will last a lifetime and would only fail on rare circumstances or in extreme conditions e.g. thunder, extreme heat, extreme cold, floods.

Materials needed:

1. Arduino Mega:
Module for controlling inputs and outputs. Also, to process all the data that comes in.

Photo sourced from (http://kernelreloaded.com/fix-arduino-mega-2560-avr-libc-issue/)



2. Moisture Sensors (YL-69):

Moisture sensor used to get data of ground moisture levels for the system to form an idea of the state of the environment around it. This is a very important part in making the system independent from user input. Because of modular design there isn't a specific number of sensors needed but 4 will be used in prototyping.

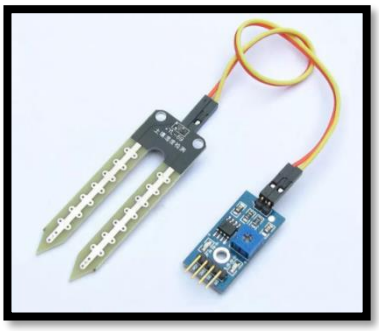


Image sourced from (http://www.microsolution.com.pk/product/soil-moisture-sensor-lahore-pakistan/)

3. Waterproof Temperature Sensor:

Sensor will be used to determine if the heat is to extreme for irrigation. If it is too warm the water will just evaporate and deem watering the plants a waste. This is meant for areas with nights with high temperatures and the light sensor won't e able to prevent wastage in its own. This sensor is also water proof which makes it ideal in a gardening setup. Only one will be used in the prototype but it will be expandable.

Image and information sourced from (www.tweaking4all.com, 2014)

4. LDR (Light Dependant Resistors):

These resistors work by using the amount of light it perceives to determine the amount of resistance it provides. This means the more light that is present the more resistance it will have. This is a cheaper alternative to day night switches and can be made to be more compact. 2 of these resistors will be used in the prototyping of the project but more can be added with modular design.

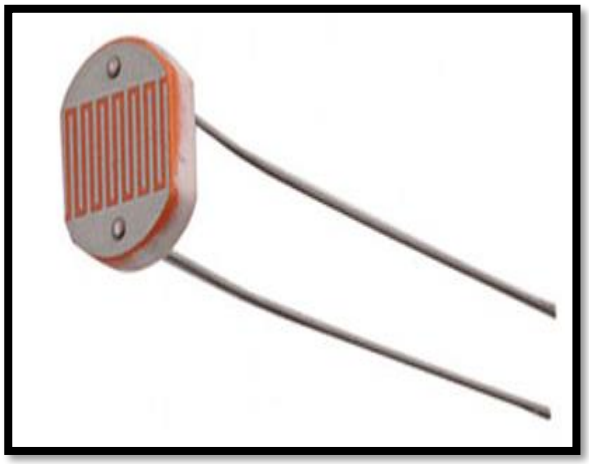


Image sourced from (https://components101.com/ldr-datasheet)

5. LCD Display 16x2 I2C:

Module used to display system information for the user.

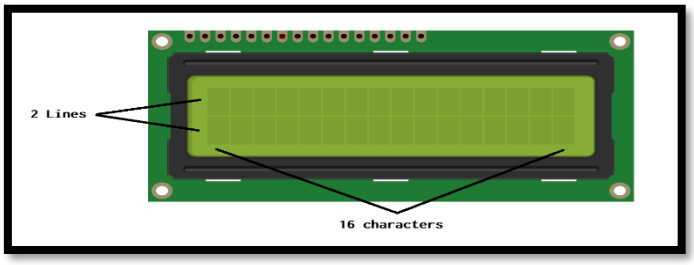
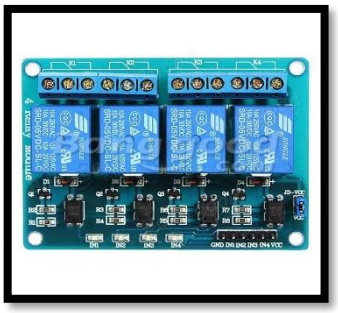


Image sourced from (https://backyardbrains.com/experiments/MuscleSpikerShield_LCD)

6. 4 Relay Module:
This module connects to the electromagnet valves of the irrigation system and turns them on when needed. Image sourced from (https://www.banggood.com/5V-4-Channel-Relay-Module-For-Arduino-PIC-ARM-DSP-AVR-MSP430-Blue-p-87987.html)



7. 12V/5V Power Supply:
This will power the system and provide a reliable logic level from which to operate from. Image provided in background study.

8. Flame Sensor:
This sensor will notify the system of any presence of flames. When conditions are very dry or susceptible to fires this will enable the system to notify any nearby persons as to assist in extinguishing the flames.

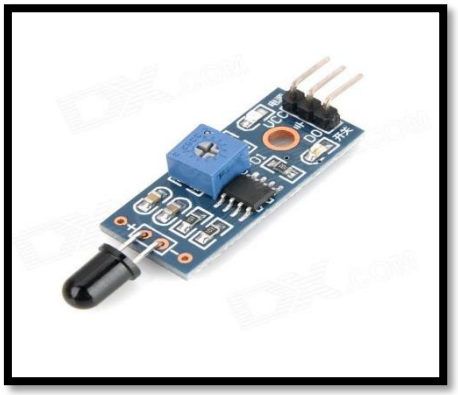


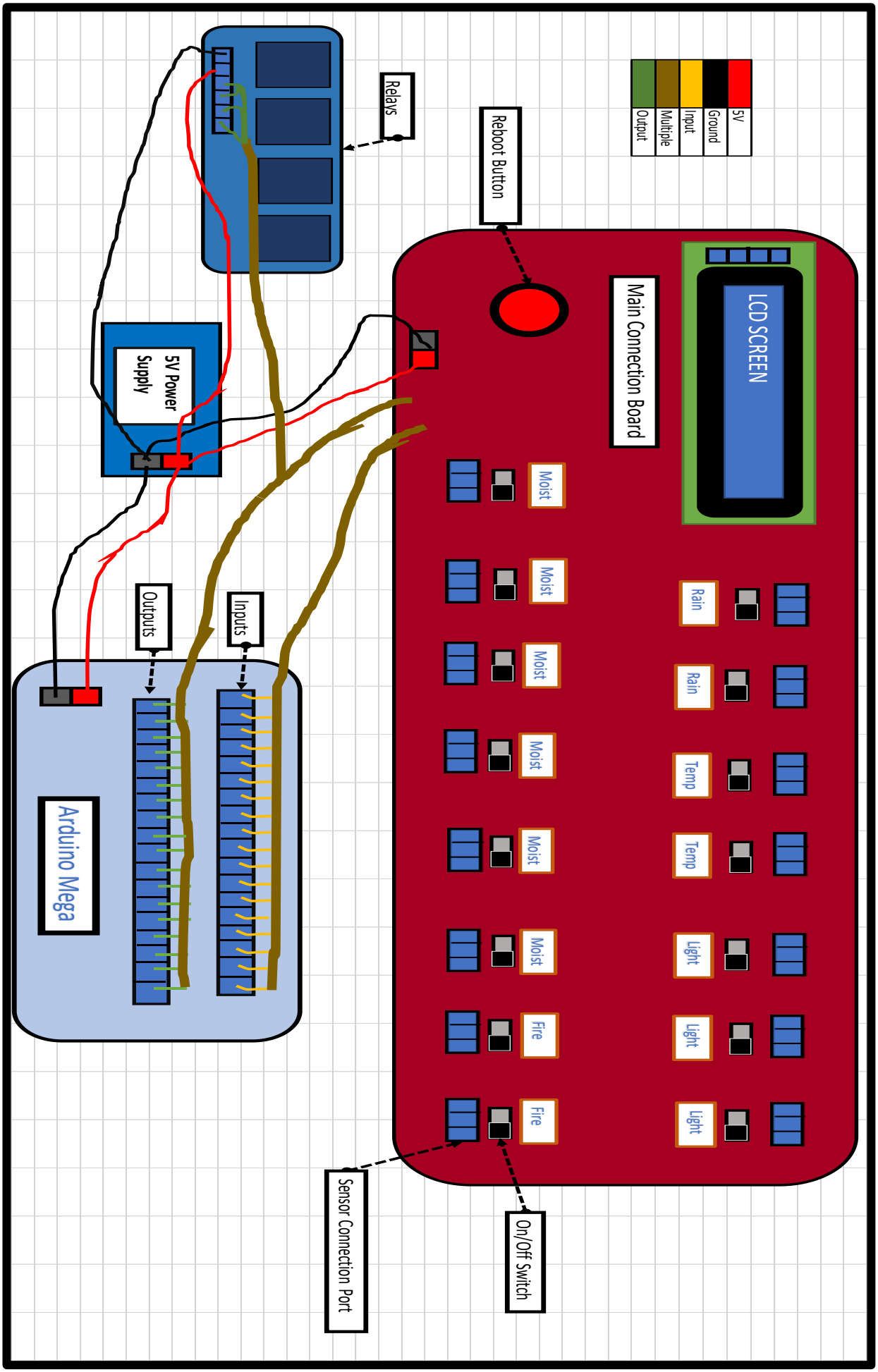
Photo sourced from (http://www.dx.com/p/1-channel-flame-sensor-module-for-arduino-blue-152020#.WwR7MUjRBPY)

Note: Wires, pin headers, resistors etc. will be clarified in the build of the prototype. This is just an overview of the main components.

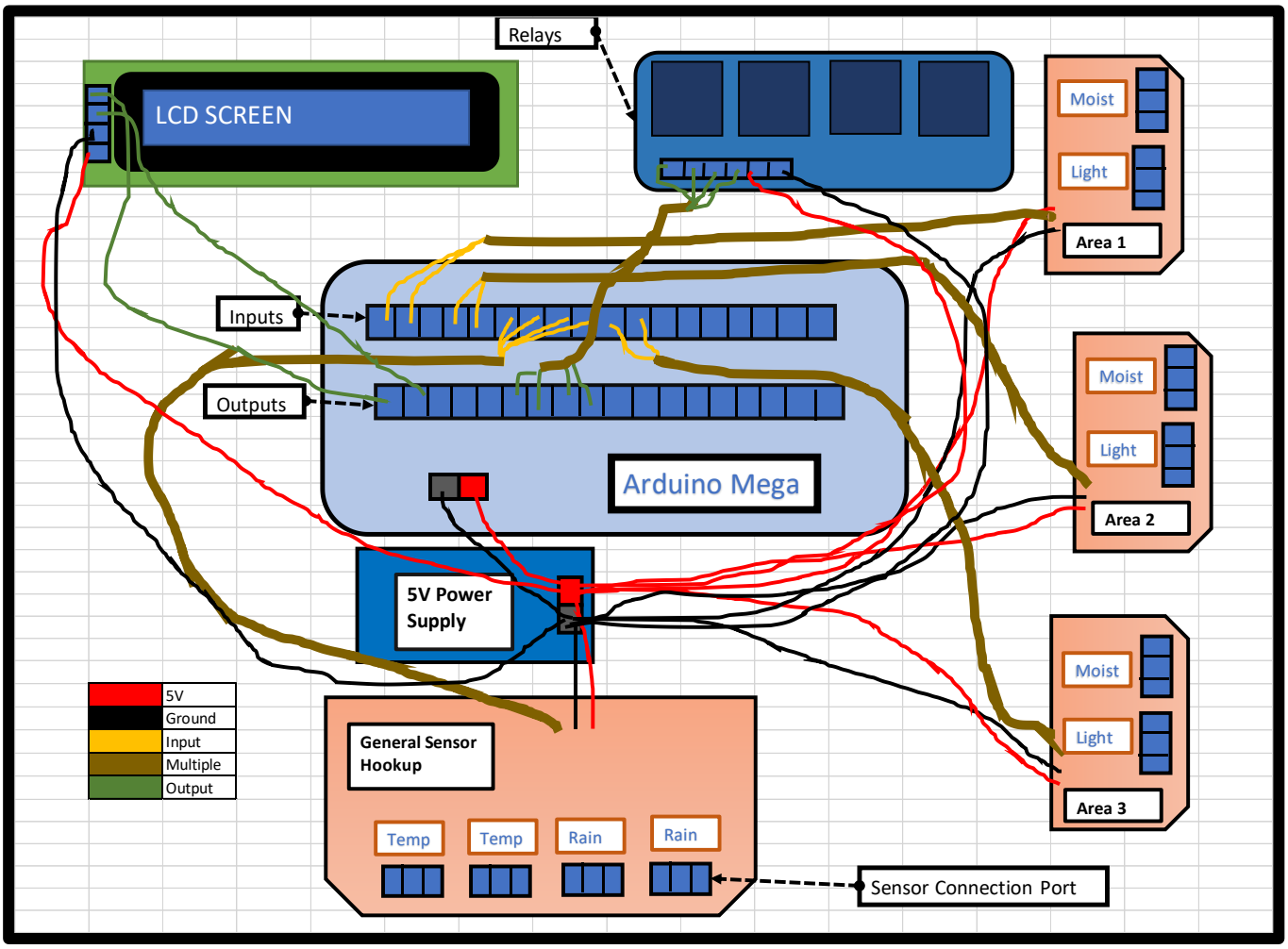
Method/Procedure:

These prototypes are designed and illustrated using Excel 2016 and were done by me, Jaco Coetsee. These are just crude representations of the setup of the system and specific wiring and soldering work will be handled in detail after final prototype design has been chosen.

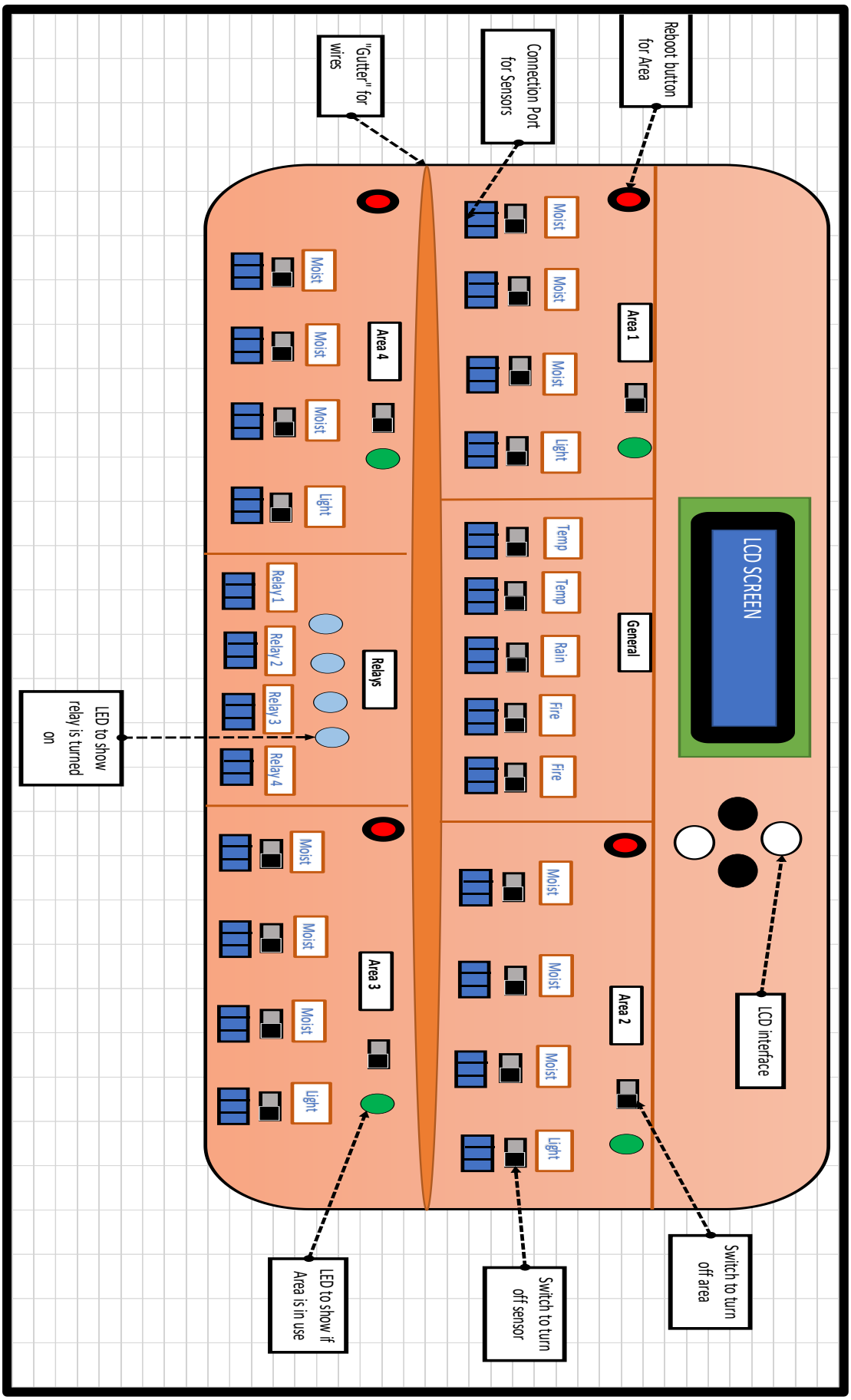
Prototype 1:


5V
Ground
Input
Multiple
Output
Relays
Reboot Button
LCD SCREEN
Main Connection Board
5V Power Supply
Moist
Rain
Temp
Light
Fire
Outputs
Inputs
Arduino Mega
Sensor Connection Port
On/Off Switch

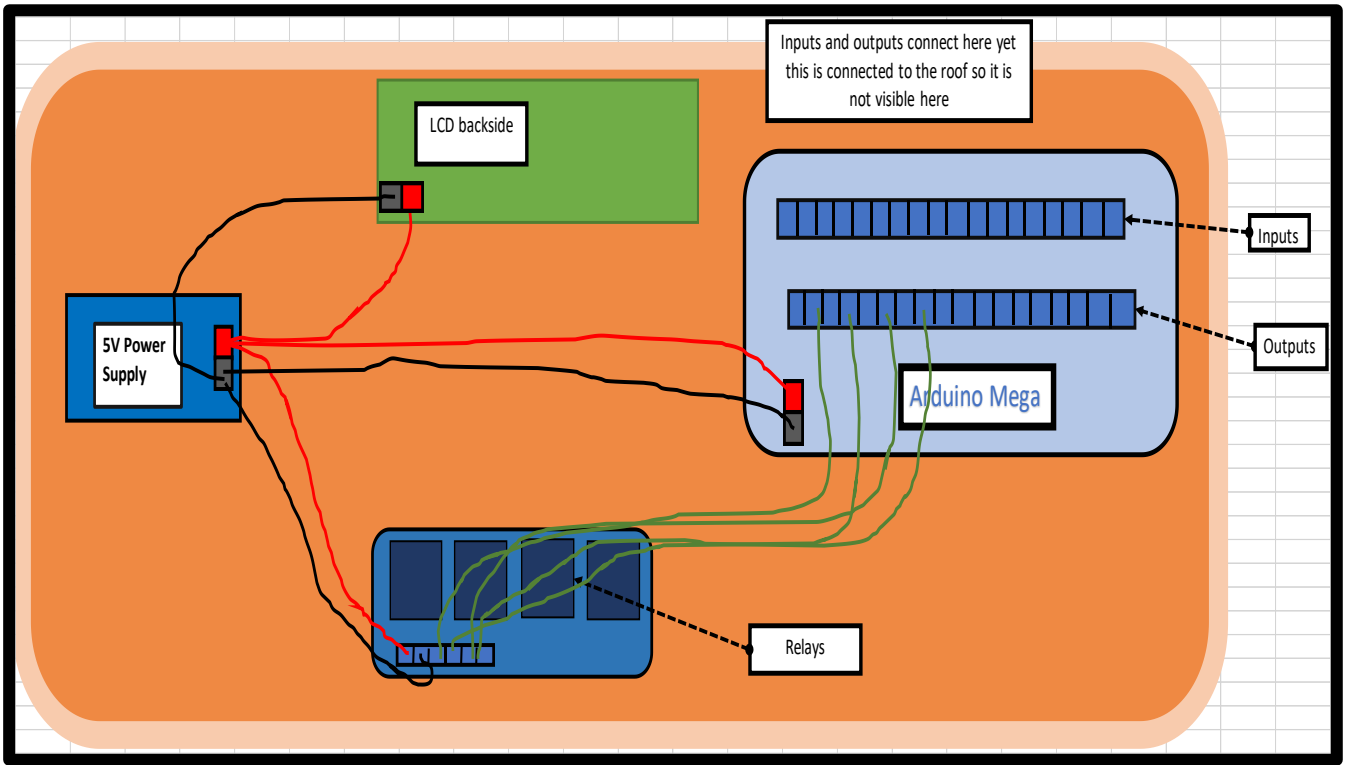
Prototype 2:


Relays
LCD SCREEN
Moist
Light
Area 1
Inputs
Outputs
Arduino Mega
Moist
Light
Area 2
5V Power Supply
Moist
Light
Area 3
5V
Ground
Input
Multiple
Output
General Sensor Hookup
Temp
Temp
Rain
Rain
Sensor Connection Port

Prototype 3(Cover/Front Side):



Internals/Inside:



Evaluation of prototypes:

Prototype 1:

Advantages:

1. Because of its switch-based design the system clearly knows when a sensor is inserted or not, so it wastes no resources trying to retrieve data form empty ports.
2. The system adapts to sensors made available so depending on the size of your garden or the amount of money you are willing to spend you can get the appropriate number of sensors for it.
3. If a new sensor is added to the system, the user can simply press the reboot button and it will recheck all the sensors and only use the ones that are turned on.
4. It is user-friendly in the sense that all the wires are hidden under the enclosure and causes a simple, ergonomic look. Not as intimidating as bare bones electronics would be.
5. Uses an LCD screen to communicate crucial data and system info to the user so that they can be aware of everything.
6. Because the power supply is separate it can power everything without having to work through another device (e.g. power is sent to Arduino and from Arduino the power is sent to sensors) this will be good to keep the logic levels constant and prevent errors in that area.
7. The simple connection port only requires the user to screw in the colour coded wires of the sensor. It isn't complicated and just adds to the modularity of the design. Plug ins would be unreliable if there is tension on the wires.
8. There being 6 slots for moisture sensors allows the design to create an accurate average to what the real environment is experiencing.
9. The light sensors allow the system to determine if it is day or night. This will allow for it to irrigate only at optimal times e.g. when there is no sunlight to evaporate water particles

10. The temperature sensor will ensure that when there is a heat wave or freak temperature rise in the night that water will not be wasted on irrigating then. This is basically a backup measure for when the function light sensors don't apply.
11. Because there are two temperature sensors, one can be placed in a shaded area whilst the other in direct sunlight to get the most realistic average for the complete environment.
12. The rain sensor enables the irrigation system to know when it is currently raining and therefore stay static and not waste water irrigating. The reason for this is the ground takes time to soak up the moisture form the rain. Therefore, to prevent the system to irrigate while the soaking up process takes place the rain sensors assist.
13. The Fire sensor allows users to place it where there are large risks of fire in the garden. Like dense wooded areas or dry patches of grass. The sprayers then enable in attempt to subdue the fire while the owner is notified.

Disadvantages:

1. Because it is not separated into areas the system will turn on all the relays while a one area might still be fine because it is in the shade for example. This will lead the wastage of water and/or over-irrigation of an area in the garden.
2. LCD screen does not have an interface so limited info can be displayed because the user is unable to scroll or switch between menu's.
3. Because Arduino Mega, power supply and relays are outside the enclosure it might be too complicated for the user to connect all the needed wires and even if this was done beforehand it still ruins the "ergonomic" look that is strived for.
4. Because the system isn't separated into areas (for irrigation) this will mean that pressing the reboot button will cause the whole system to restart and is very inefficient versus having all the areas working independently from each other and only one of them being rebooted at a time.
5. Because there are 3 different light sensors it will only take one badly placed sensor (e.g. that is in a heavily shaded area) to throw off the average all the sensors generate together. This might cause the system to never irrigate in the first place.
6. Having two rain sensors is potentially a waste and might corrupt the data the system receives in the sense that it will return a Boolean value. It is either raining or it is not, which means calculating the average of this means that the Boolean is always false if one of the rain sensors is under a roofed area.

Prototype 2:

Advantages:

1. It is separated into areas, so relay will only activate specific sprayers as the area needs it. Because of this the shaded areas will get less water because they don't require as much whilst open areas will receive more water since they require irrigation more often. This will ensure every area gets the correct amount of water according to the environment whilst still expending the least amount of water in total.
2. The open design will make repairs easier since nothing needs to be opened. Though a knowledgeable person will have to do this. General sensors are separated from area sensors which maximise the minimal use of sensors and the fetching of useless data.
3. Areas can just be bought on as needed. When the lad expands the consumer just buys a new area module.
4. Because there are two temperature sensors, one can be placed in a shaded area whilst the other in direct sunlight to get the most realistic average for the complete environment.

5. Because of the open design even key components can be removed, like the LCD, the person (with the right knowledge) can use the software and make his own hardware tweaks. Which means that the system is very customizable.
6. The rain sensor enables the irrigation system to know when it is currently raining and therefore stay static and not waste water irrigating. The reason for this is the ground takes time to soak up the moisture form the rain. Therefore, to prevent the system to irrigate while the soaking up process takes place the rain sensors assist.
7. Because the power supply is separate it can power everything without having to work through another device (e.g. power is sent to Arduino and from Arduino the power is sent to sensors) this will be good to keep the logic levels constant and prevent errors in that area.
8. It uses an LCD screen to output info about the system which could be helpful for the user or technician when troubleshooting needs to be done.

Disadvantages:

1. There is no fire sensor thus if I fire occurs there will be no measure against it. This is particularly dangerous during the dry season.
2. Because of the extreme modular design, it has a lot of loose parts. These parts even when installed correctly will be difficult to mount because of the excessive number of them.
3. They will need a skilled professional to install this because of all the wires and components that must be connected.
4. It looks very cluttered and complicated and it will be daunting for the consumer to try and figure it out.
5. There is only one moisture sensor for each of the designated areas which might lead to inaccurate data since the average moisture across the whole area won't be considered but only a specific spot in this area.
6. When sensors are removed or added the system won't have away of knowing it. It will be terribly inefficient for this to be done from the software side (having to constantly check what sensors are there), there some interface should be allowed to let the system know which sensors are connected.
7. 3 light sensors no matter if it is in different areas are a bit excessive since a maximum amount of only two are needed to determine if it is night or day. So having one for each area is a waste of computing resources and an unneeded expense.
8. Having two rain sensors is potentially a waste and might corrupt the data the system receives in the sense that it will return a Boolean value. It is either raining or it is not, which means calculating the average of this means that the Boolean is always false if one of the rain sensors is under a roofed area.
9. The LCD screen lacks an interface which means that there can be no interaction with it e.g. scrolling through system info are changing between menu's.

Prototype 3:

1. LCD screen can output crucial system info to the user. It also has an interface, so the user can switch between menu's which allow for more types of data to be displayed.
2. Because the design is split into areas the system will run in separate parts that work independently from each other but also simultaneously. This will mean that efficiency is maximised whilst also causing the least expenditure of water because an area will only be irrigated if only it's needs are met.
3. Because of the switch-based design, the user can select which modules he wants active and that has the needed sensor.

4. Because the switches can toggle on and off it allows for the user to simply swap out broken sensors or add additional sensors for more accuracy.
5. The system adapts to sensors made available so depending on the size of your garden or the amount of money you are willing to spend you can get the appropriate number of sensors for it.
6. When changes have been made the system can just be rebooted by pressing the red button without having to let the software try and figure out what changes have been made. This saves a lot of processing power and time.
7. Because the reboot buttons are separate for each area it means that the areas not being rebooted can continue operation even though one is rechecking for sensors. This means there is minimal disruption between areas.
8. LED's indicate if an area is switched on or not by flashing green if it is and being turned off if it isn't. This will give the consumer a clear indication if the area is working properly and adds to the user-friendliness of the design.
9. When a relay is turned on the opposing blue LED gets turned on as well. This can be used for troubleshooting and for the user to check if he as assigned the sensors to the correct area
10. All the "ugly" hardware components are inside the enclosure, so it won't be an eye sore for end-user.
11. The power supply is close to all the components and can connect to each one separately. This means that if one sensor fails it won't affect any others by disrupting the flow of electricity.
12. The relay connection ports are in a central location on the command centre which makes for easy access.
13. Because each area has access to 3 moisture sensors the data that is inputted to the system will be very accurate. Yet it still allows the user to prioritise low cost by only requiring a minimum of one.
14. Each area has access to one light sensor which will mean that depending if it is in the shade or in direct sunlight the system will tailor make an irrigation plan for it.
15. All the general sensors can be connected to the middle and only the minimum of them are used to get the most accurate result.
16. There being only one rain sensor means that data can be corrupted because of opposite Boolean readings. Yet it still is helpful to check for rainfall, just with no drawbacks.
17. Fire sensors can be put in high risk areas for fires which will activate all the sprayers and notify nearby people to subdue the fire.
18. Because there are two temperature sensors, one can be placed in a shaded area whilst the other in direct sunlight to get the most realistic average for the complete environment. Yet it is still optional how many you desire.
19. Everything is clearly branded as to promote an aesthetic look.

Disadvantages:

1. A lot of wires will be used to connect the sensors. But because the connection ports are on top of each other it might get cluttered or hard to fit more wires. Yet the gutter in the enclosure design combats this as best as possible.
2. Because al the wires are packed into one enclosure it might get tight for space inside and cause internal repairs to be increasingly difficult. This will also add to production complexity.
3. The final product might be very large because it must house so many components. Yet this is preferred instead of a lot of loose wires and command centres like in prototype 2.
4. The user might accidentally switch something on leaving the system trying to read data from a non-existent sensor which will severely damage the reliability of the system.

Conclusion on Prototype Choice:

My choice was based on which of the features make it the most user friendly whilst still not sacrificing on the reliability and accuracy of the system. I believe therefore that prototype 3 is the most suitable design for me to build if I would like my system to achieve all the goals I am expecting of it. Prototype 3 is also very modular which means the user can make his/her own variation of it whilst still being accurate etc. It also includes all the above advantages and disadvantages included.

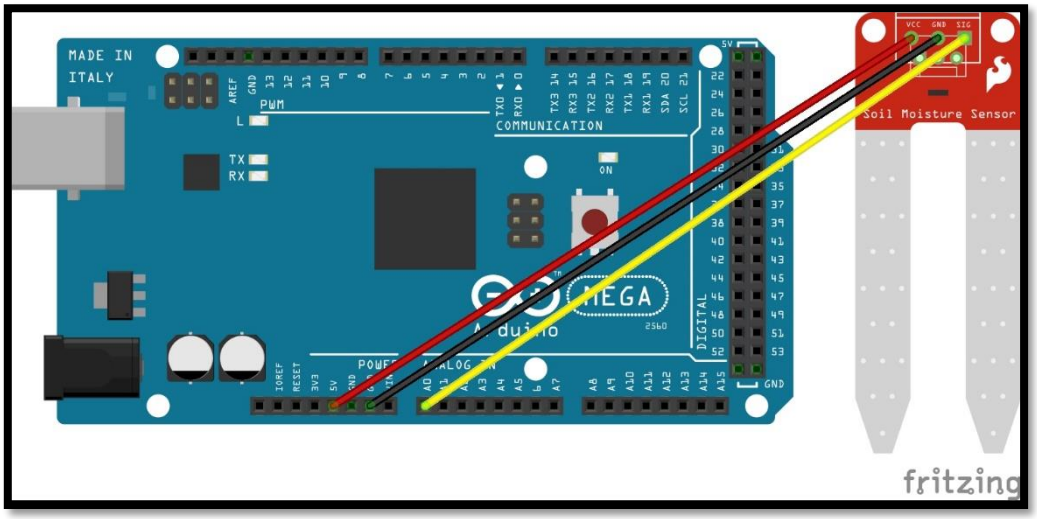
Getting Sensor Parameters:

Now that the final prototype has been chosen and all the needed sensors have been identified, I need to get the parameters for all the sensors that will determine if irrigation is needed or not. This will be done by writing the necessary code, creating the circuit and then the correct values are determined by controlled tests.

Moisture sensor:

First, the circuit must be implemented for the data to be retrieved. The circuit was illustrated using Fritzing which is a circuitry design program that will be used extensively throughout the planning of this system. This circuit was however simple as it only has 3 connections consisting of the following: Mega 5V to VCC, Mega ground > ground, Analog 0 > Signal.

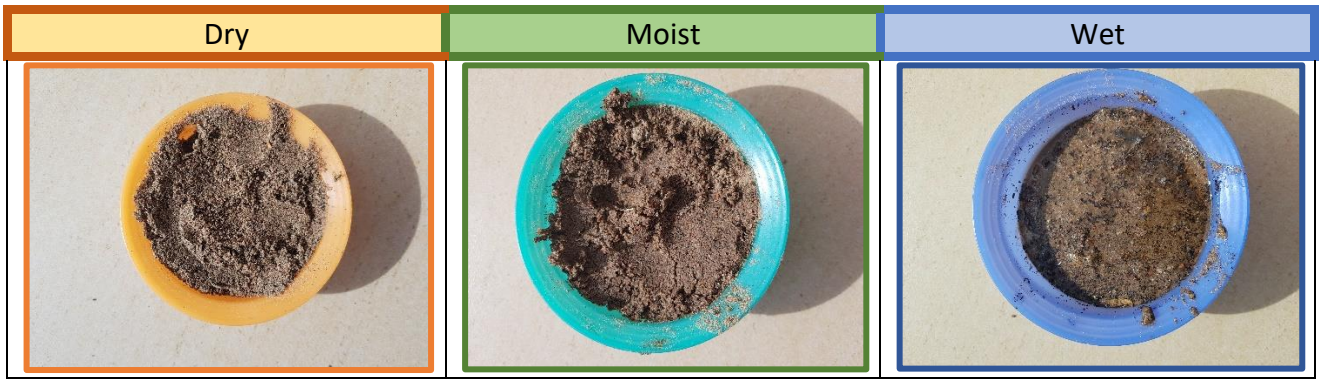
NOTE: A different module moisture sensor is used in the illustration, but the wiring stays similar.



Next the code had to be written to input the data into the system. The program is simple, all it does is it reads the data in from the sensor and then outputs it on the Serial Monitor. The code was written for the Arduino Mega (the one illustrated above) and written in the Arduino IDE. The code is as follows:

```
const byte moistPin = A0;

void setup() {
  Serial.begin(9600);//Begin communication with device
}

void loop() {
  Serial.println(analogRead(moistPin));//read value of moisture
  delay(1000);//delay 1 second
}

/* OUTPUT: Water =              S1: 321        s2: 297
 *         Dry Ground =         s1: 857        s2: 830
 *         Moist Ground =       s1: 262        s2: 260
 *         Extremely wet =      s1: 230        s2: 225
*/
```

The test design for the sensor consisted of three samples of ground: dry, moist, wet. These samples were tested using two moisture sensors to ensure that data was consistent, and that a faulty moisture sensor wasn't used. The ground looked as followed from a qualitative perspective:



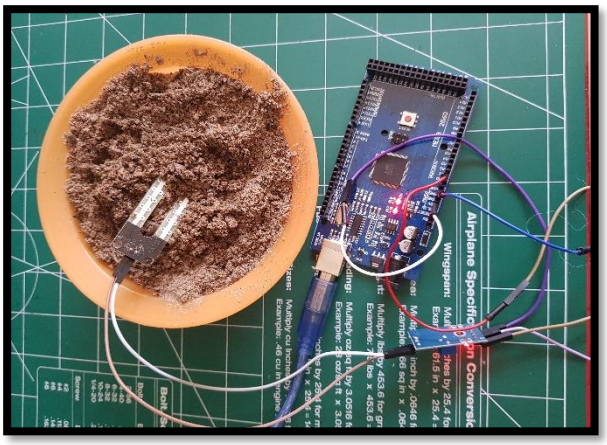
As perceived on top the moist ground will be the ideal level for the ground while the dry will be the threshold that the system uses to determine if irrigation is needed. The results are as follows:

Sensor	Dry	Moist	Wet
Moisture 1	857	262	230
Moisture 2	830	260	225

The test results showed that the moister it got the smaller the difference was that the sensor perceived, but that does not affect the programming since we will be working with an exact threshold where the ground needs irrigation. This was concluded to be the value 450, this value allows the garden to stay healthy whilst also prioritising the minimal waste of water.

The setup for the test looks as follows:

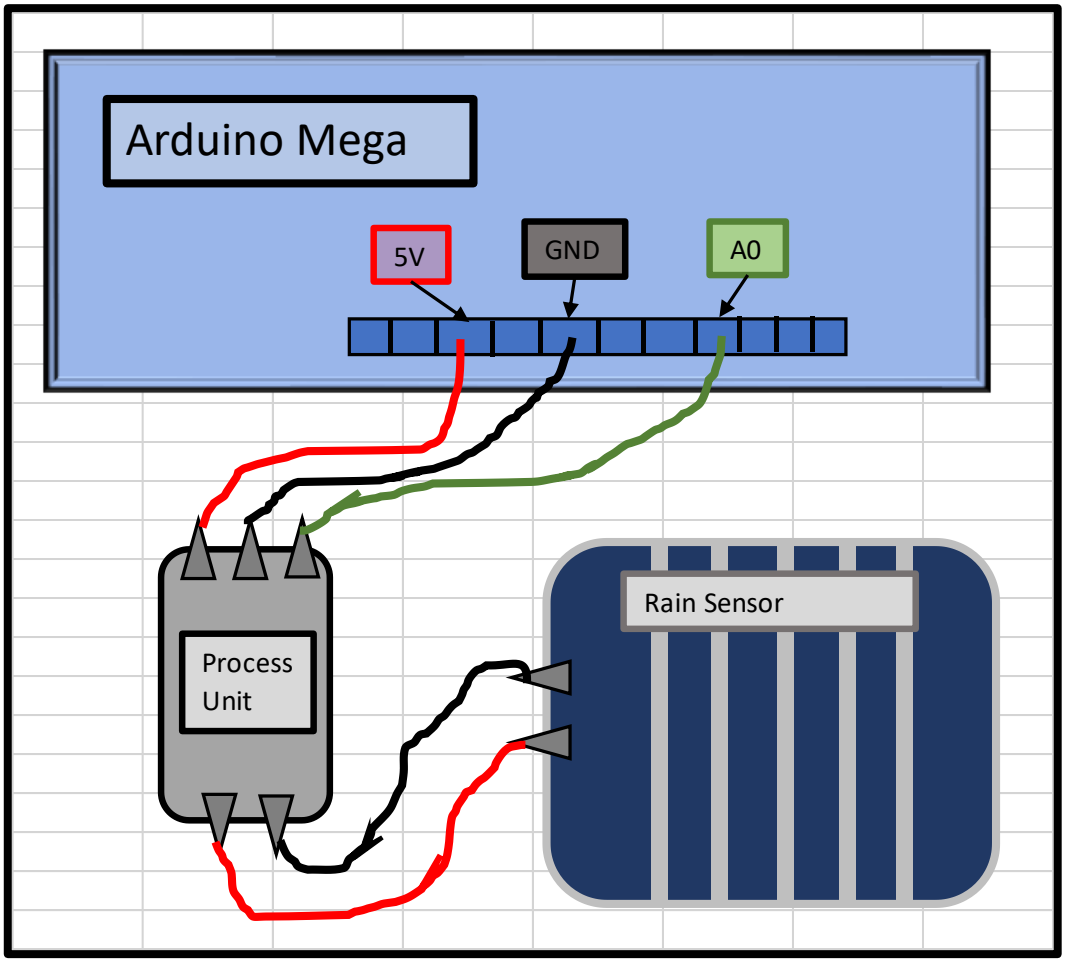
NOTE: The moisture sensor was fully inserted into the ground during the test but for presentation purposes it was slightly lifted.



Rain Sensor:

The circuit for the rain sensor is also straight forward wiring. It consists of two parts: the “processing unit” and the actual rain sensor. Fritzing did not support the rain sensor, so a rough drawing has been made in Excel to illustrate the connection.

Note: Processing unit strengthens the analogue input from the rain sensor for it to be able to be processed by the Arduino. That is why it can’t be connected directly.



The code for the Arduino was simply to read from the rain sensor input pin and output that value to the Serial Monitor so it can be evaluated. The code is as follows

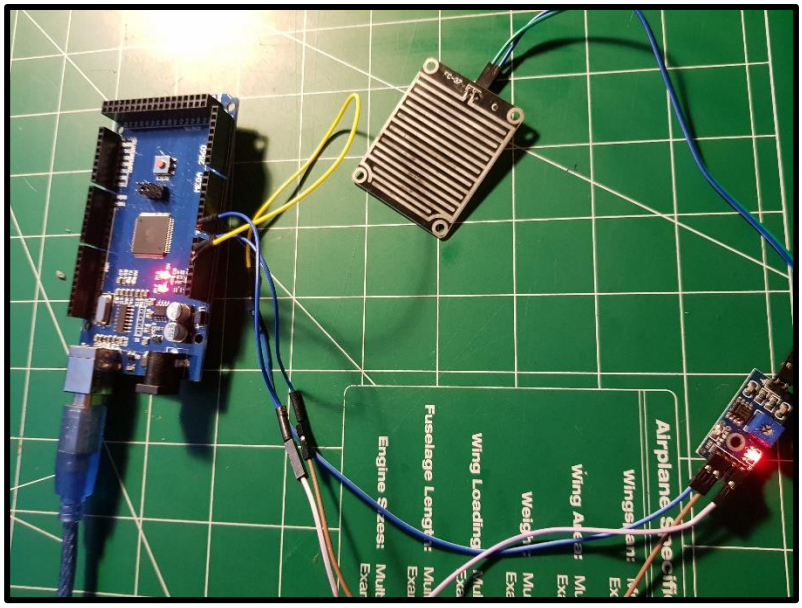
```
const byte rainSens = A0;//initialize rain sensor pin
int rain = 0; //create a variable to store the analogue value
void setup() {
  Serial.begin(9600);//Start Serial Communications
}

void loop() {
  rain = analogRead(rainSens);//assign value read to rain variable
  Serial.println(rain);//output the value
  delay(1000);//Delay one second

}

/*Results:    Sensor          Value          condition
 *            rain            <300           with water on it
 *            rain            >900           no water on it
 *
 */
```

The control test to retrieve the value consisted of reading the value the serial monitor output during 2 states: with water on it and without water on it. This is a control test and it is not done in real rain, for rain is just water and this can be simulated. If different results are encountered later when actual rain is involved the values retrieved here must be revisited and adjusted. The results are as follows:

Sensor Type:	Sensor State	Output Value
Rain	With water	Below 300
Rain	No water	900 and above

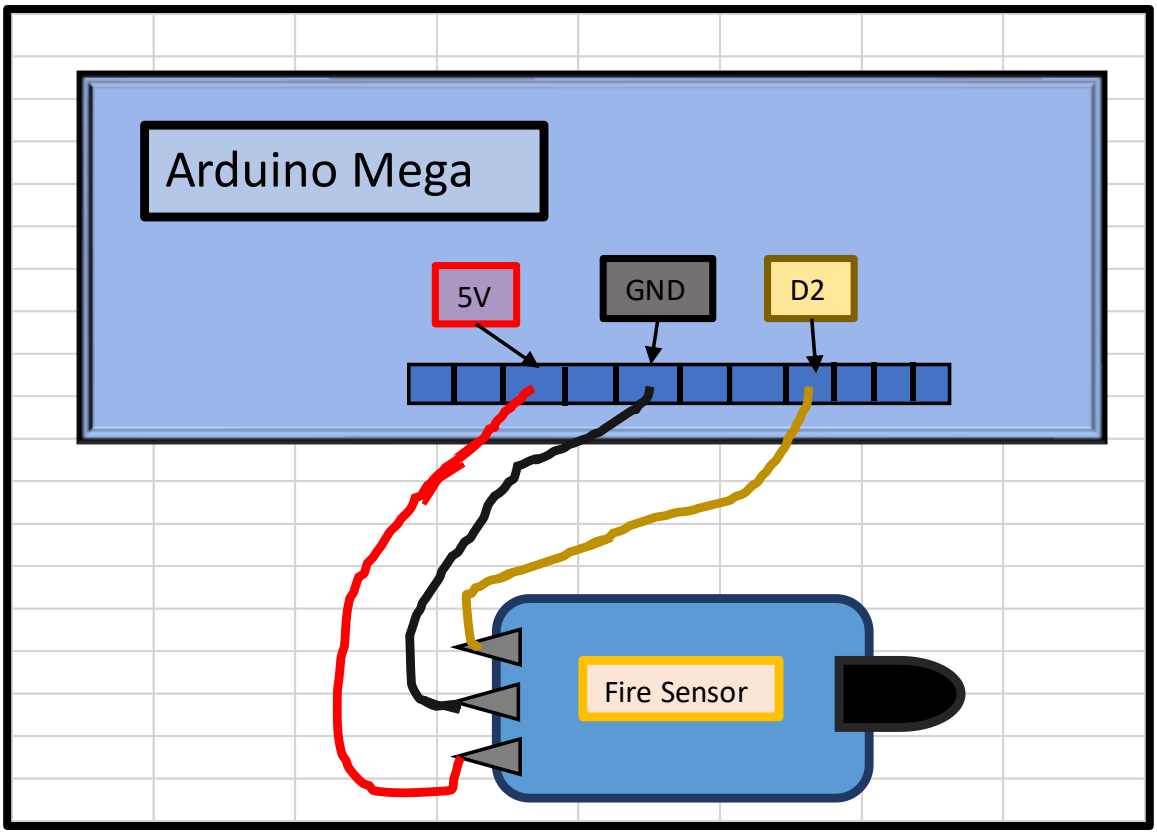
It was clearly visible in the results where the thresholds for the rain sensor should be. Below 300 when rain is present and above 900 when rain isn't present.

The setup for the test environment looked as follows:



Fire Sensor:

The fire sensor consists of the physical fire sensor attached to a processing unit. The sensor gives feedback to the processing unit and the unit determines if there is a fire source by or not. The sensitivity can be adjusted using the potentiometer. First, to begin testing the correct circuit must be implemented. This only consists of powering the sensor module and retrieving the data back via a digital input sensor. This was made by Excel yet again because Fritzing also doesn't support the fire sensor. Thus, it is only a rough sketch, but it illustrates the real-life circuit accurately.



The testing for this is a bit tricky since there are no absolute values. Calibrating the system is based of turning the potentiometer and then the sensor outputs true or false (0 or 1) if it senses a fire given the sensitivity on it. The code is just to give feedback through the Serial Monitor on what the sensor is outputting.

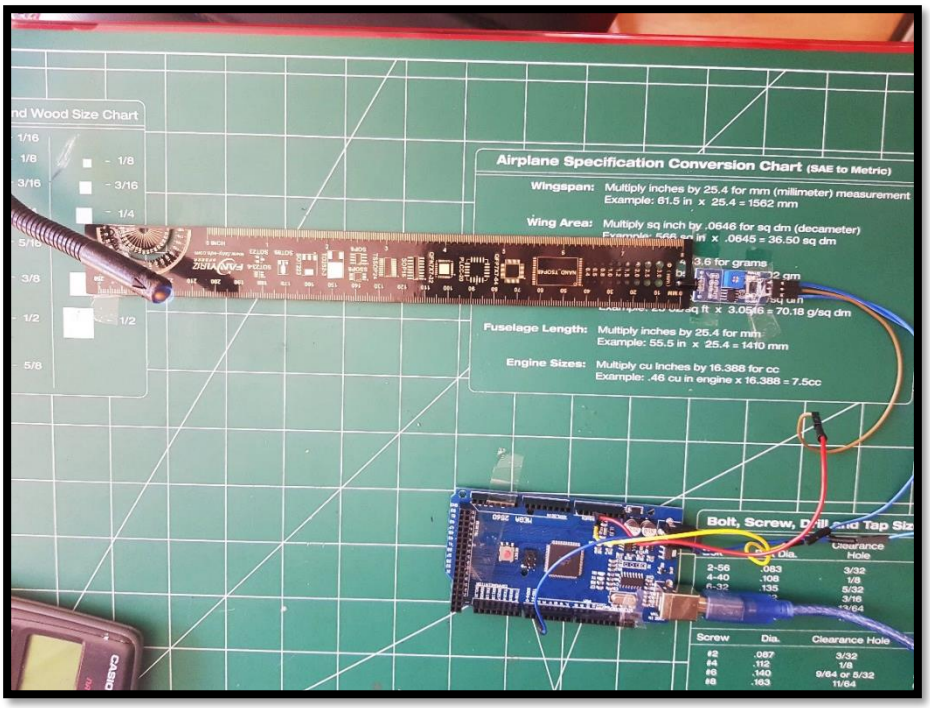
Note: The sensor works inversely: when there is a fire it returns 0 and there is no fire it returns 1.

```
const byte flameSens = 2; //initialize flame sensor pin
void setup() {
  Serial.begin(9600); //Begin serial communications
}

void loop() {
  if(digitalRead(flameSens){ //check if flame sensor returns true or 1
    Serial.println("There is no fire.");//print response
  }else{//if it returns false or 0
    Serial.println("There is a fire!");//print response
  }
}
```

Ideally the sensor should be as sensitive as possible without falsely setting off disrupting all operations of the system. To do this a ruler was taped to the workbench measuring the distance between the sensor and the “fire”. Every sensitivity level was tested and checked for maximum distance. Sensitivity starts from most sensitive ,reducing with a quarter of a turn (on the potentiometer) with each test.

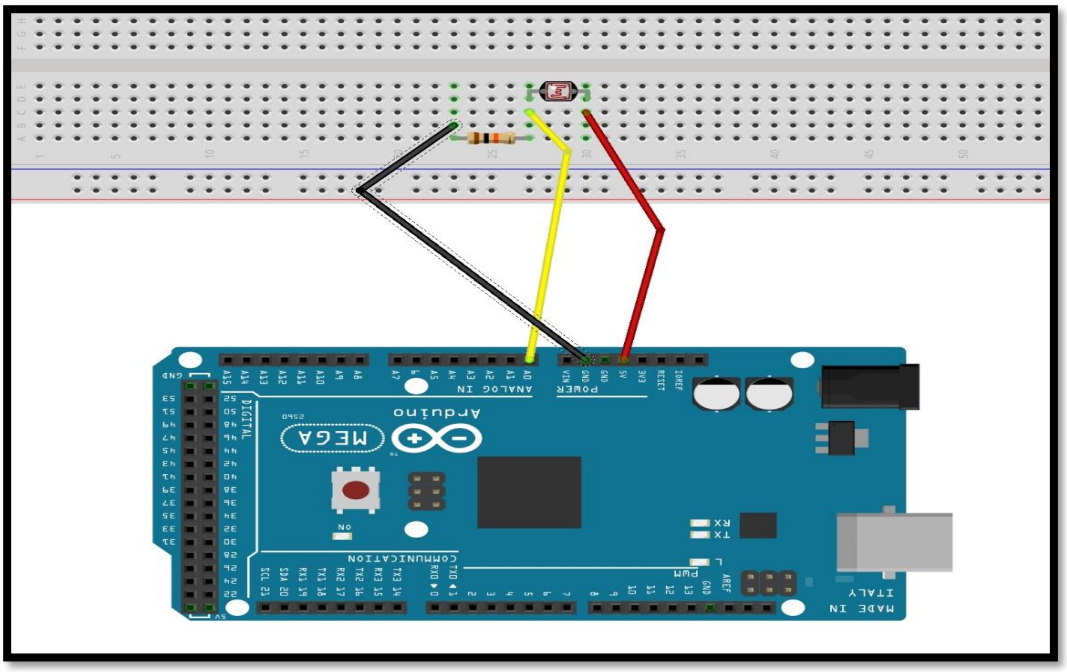
Sensitivity (Number of ¼ turns)	Distance (centimetre)
0	Infinite(always returns true)
1	Infinite(always returns true)
2	42
3	31
4	20
5	5

With these results it is concluded that the best configuration of the sensor is 2 quarter turns that allow for maximum distance whilst also not sacrificing accuracy .The setup for the test environment looked as follows:



Light Sensor (Light Dependant Resistor):

The light dependant resistor uses the amount of light it is exposed to determine the amount of resistance it gives. The more light that is present the less resistance is present therefore there is a bigger potential difference and the sensor outputs a higher value. The light sensor will be used to determine if it is night or day to more efficiently run the system. It does this by using the following circuit: At one end it is powered by 5V and the other the sensor output wire is connected between the actual sensor output and a 10K resistor going to ground. It looks as follows:



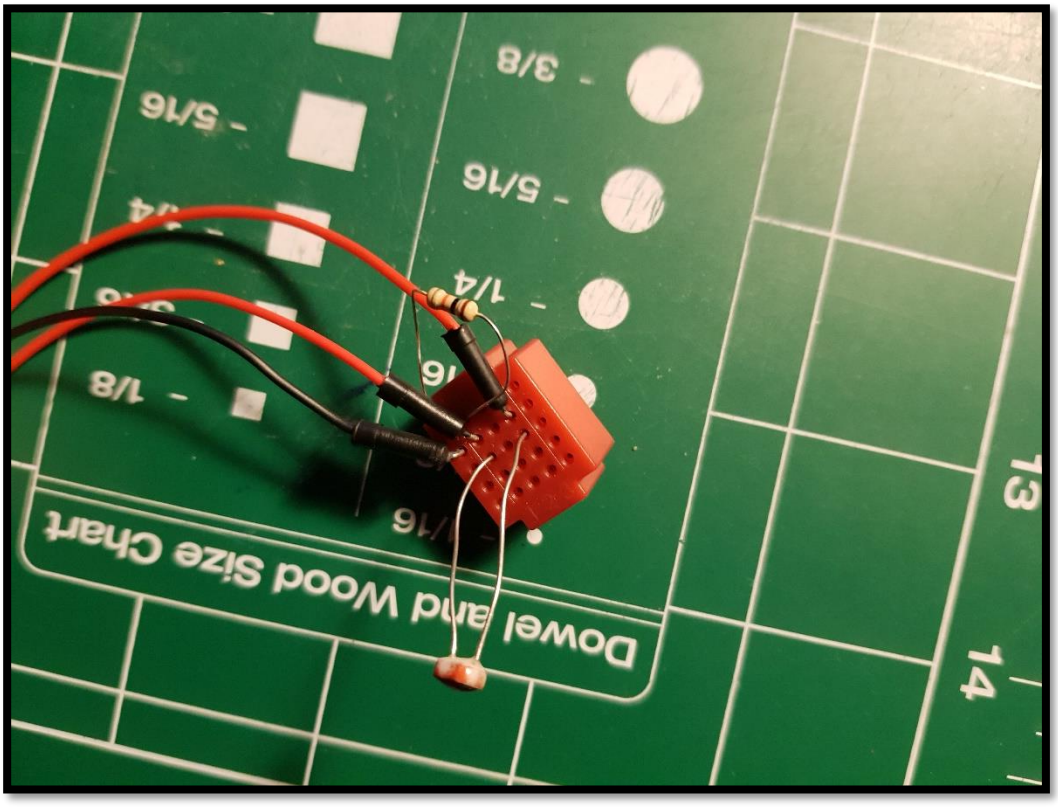
The code for this sensor is repetitive of the other sensors. It just involves retrieving the value from the sensor and then printing it to the Serial Monitor so it can be interpreted by me.

```
const byte light = A0; //declare LDR pin

void setup() {
  Serial.begin(9600);//Start Serial Communications
}

void loop() {
  Serial.println(analogRead(light));//Get value from LDR and then print
it
  delay(1000);//Delay the printing so Serial Monitor is readable
}
/*Results:        Condition                    Value
 *                  light                      650
 *                complete darkness            33
 */
```

The testing was used using artificial light and it will be revised if the system does not work correctly in natural conditions. How the tests were conducted is by darkening the given room (e.g. closing the blinds and the door and switching of the main light) and then using one light source namely the desk lamp. The sensor was placed beneath it to simulate day and when it was turned off it resembled night. The results were as follows:

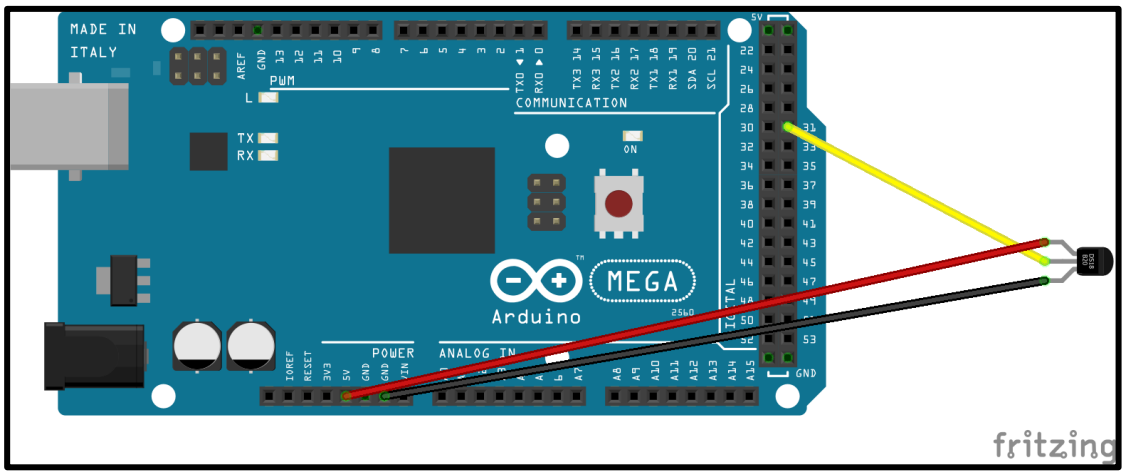
State of Light(Time of Day)	Sensor Output Value
Off(Completely dark)	33
On(Day)	550

The results leave me to conclude that day is equal to any value more than 550 and that night is equal to any value less than 550 .These results will be used accordingly in the program that will run the system and changed if functioning is affected by unrealistic test conditions (e.g. artificial light). The setup for the test environment looked as follows:



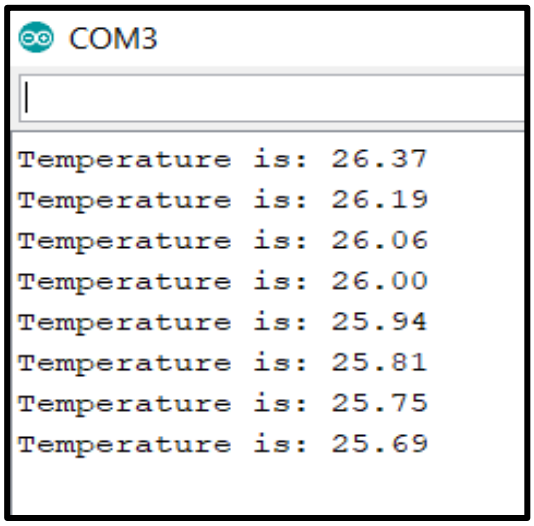
Temperature Sensor (Dallas temperature sensor):

The temperature sensor is a waterproof sensor that can determine the temperature of a given environment. It requires a library though to interface it and therefore the library must be downloaded and included in the test program. The setup is simple since there are only three ports that need to be connected. Left leg to 5V, Middle to Ground and the right leg is connected to sensor pin as follows (made using Fritzing):



The code can be examined using the snippet. First the needed libraries are imported then the pin where the temperature sensor is hooked up to is declared using the variable `ONE_WIRE_BUS`. Then an object of the OneWire library is created using this variable. In the same fashion this object is used to create instance of the Dallas Temperature library. This instance is started as well as the Serial monitor. The protocol to retrieve the temperature data is followed and outputted to the serial monitor to check accuracy.

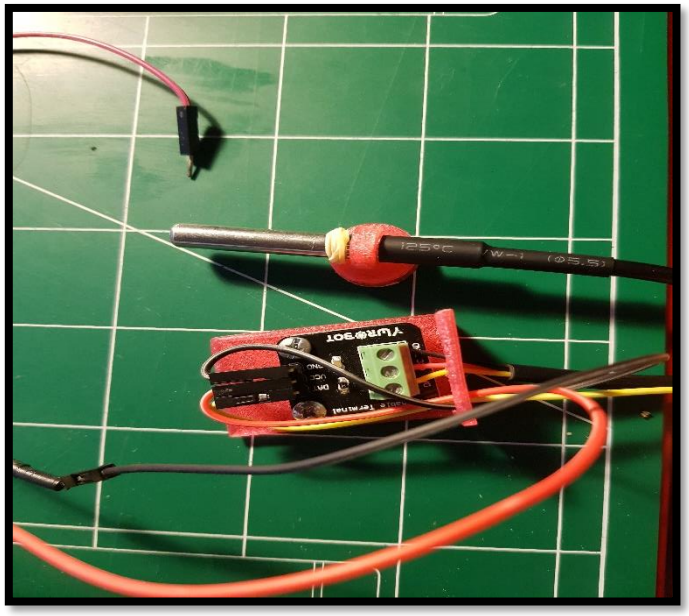
Serial Monitor Output:



Code Snippet:

```
#include <OneWire.h> //include needed libraries
#include <DallasTemperature.h>

#define ONE_WIRE_BUS 31//Declare pin 31 as ONE_WIRE_BUS

OneWire oneWire(ONE_WIRE_BUS);// Setup onewire instance to communicate with the
sensor

DallasTemperature sensor(&oneWire); //Create an instance of the Dallas
temperature sensor

void setup(void)
{
  Serial.begin(9600);//Start Serial communication at a baud rate of 9600
  sensor.begin();//start temperature sensor
}

/*
 * Main function, get and show the temperature
 */
void loop(void)
{
  sensor.requestTemperatures(); // Send the command to get temperatures
  Serial.print("Temperature is: ");// Make it more readable by printing
description first
  Serial.println(sensor.getTempCByIndex(0)); // The function ByIndex to get the
temperature from one sensor only. (0 = 1st device)
  delay(1000);//Delay one second to make it more readable
}
```

After careful consideration it was concluded that the maximum temperature before irrigation is unreasonable should be 25 degrees Celsius. This temperature will be used to control irrigation in the program and will be revised in the future if it is deemed illogical. The setup for the test Environment looks as follows:



Relay:

The relay is an electromagnet that enables the Arduino to interface devices that operate at a higher voltage than 5V. Like the sprayers which are commonly 12V. This will allow the Arduino to turn on and off the sprayers when conditions are met by the program. I originally planned on using the 4 Relay Module that was listed earlier in this project but decided to rather design my ow circuit with the bare bones components.

Advantages of own circuit:

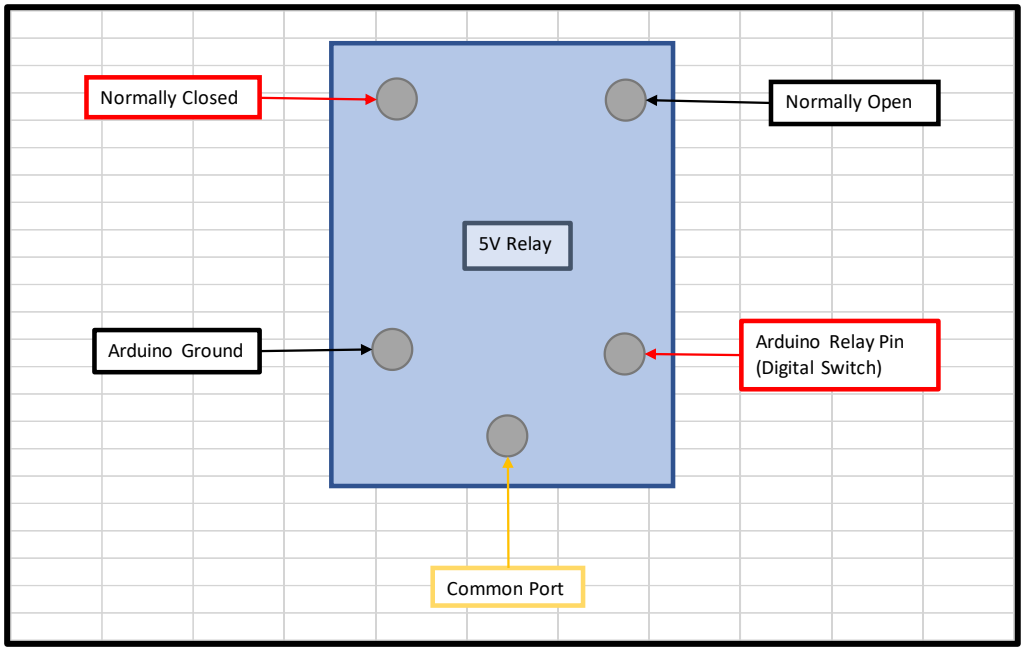
- Will use less space in the box
- Will waste less power than the alternative
- Can be modularised to fit into less conventional spaces
- Allows for more control
- Higher specificity and less noise between relays

Relay Functioning:

Activating the relay requires the electromagnet inside of the relay to be switched on and when deactivated the electromagnet needs to be switched off. Switching the relay on is accomplished by creating a potential difference (apply voltage) over the two input pins (displayed below). When you remove the potential difference (remove voltage) then the relay is switched of.

Of course, there must be stated what is meant by “Switched Off” and “Switched On”. Well the relay has 3 pins specific to the electromagnet mechanism: Normally Open, Normally Closed and Common. The normally open pin is where the voltage that the connected device will receive when the relay is switched off is connected e.g. 0V which in that case will mean it is connected to ground. Normally Closed is where the voltage that the connected device will receive when the relay is switched on is connected e.g. 12V which is the operating voltage of the sprayer switches of the average home. Then finally the common is the output pin. This pin is connected to normally open and normally closed as the relay switches between on and off.

Relay Pin Layout:



3D-Printed Housings Design:

Rain Sensor:

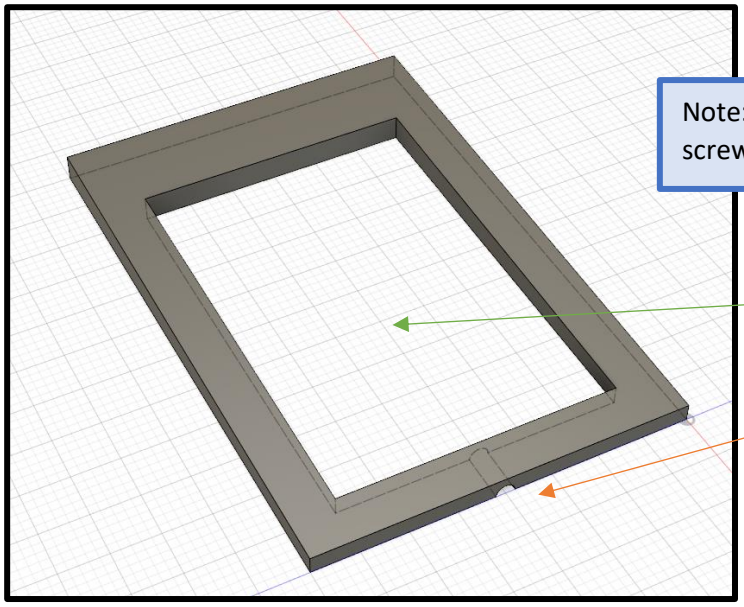
Function: The aim of this housing is to expose as much of the rain sensor's surface as possible whilst still protecting the electronics within. The idea is also that the water must evaporate to be able to leave the surface of the rain sensor, this way the system will be aware of rain for longer and compensate for its lasting effects. Yet for large quantities of water that might cause damage there must be a drain type structure set in place to let the majority leak out.

Description: This housing consists of two parts: the top and bottom. The top part is placed on with screws and is basically a large rectangle with the surface of the rain sensor cut out onto it. It also has a drain like hole at the bottom to let large amounts of water pass through. The bottom will contain all the rain sensor's electronics and an exit point for the wires to be lead to the Arduino (main hub).

Criteria:

- Space for wires to enter/exit housing
- Should be easily made waterproof
- Big opening for surface of rain sensor
- Flat in the sense that water can't run off it
- A drain to remove excess water

Top:

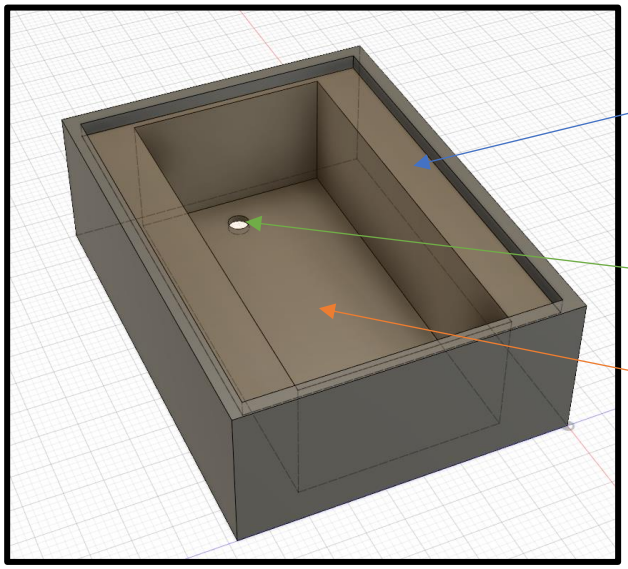


Note: The holes for the screws will be drilled in.

Cut out to expose the surface of the rain sensor.

"Drain"

Bottom:

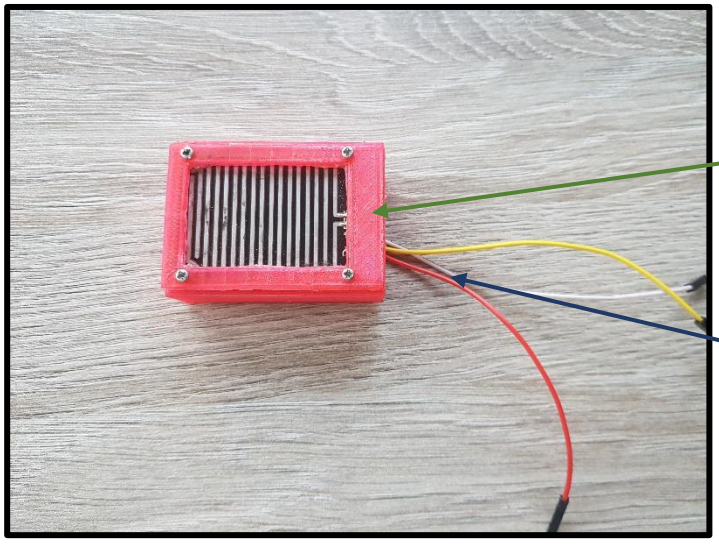


Space for holes to be drilled into.

Hole for wires

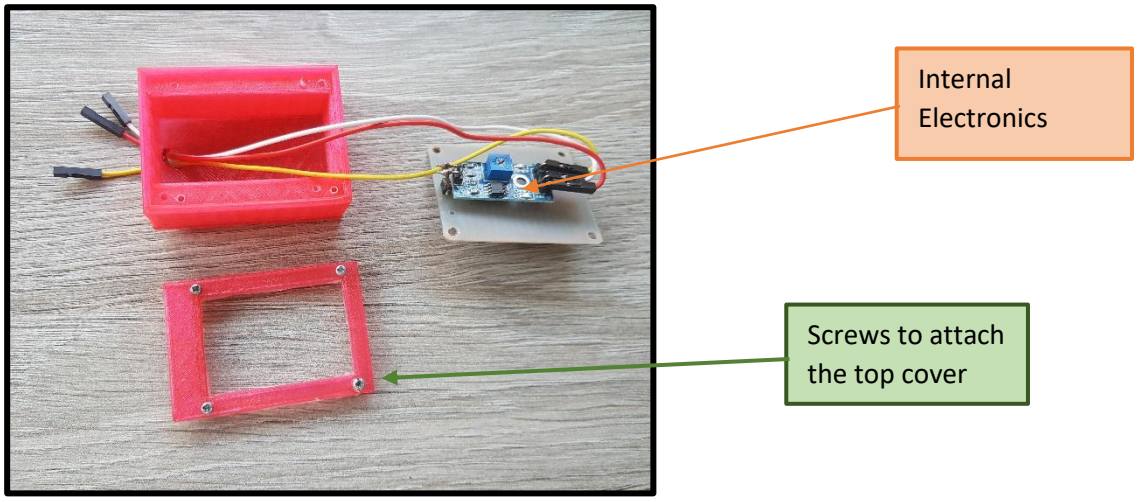
Space for electronics

Final Print:



Rain Sensor Cover

Wires for sensor



Fire Sensor:

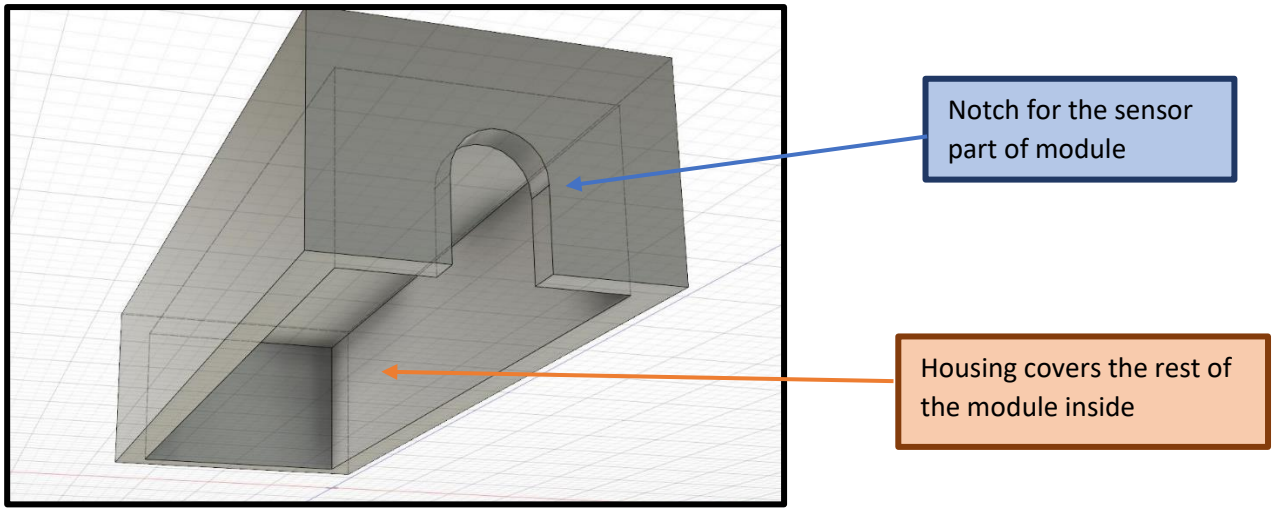
Function: The aim is to create a structure that exposes the black sensor part of the module that picks up the infra-red rays, whilst also creating a housing that protects the electronics from the elements.

Description: The design consists of two parts: The cover that has a spacing in it where the sensor part of the module sticks out and the electronics get covered. And the base which is where all the components will be glued onto. The base also has a hole where the bulk of the wires will go through and it has a resting structure where the sensor part of the module will be held up on. All of these structures are attached through snapping onto each other.

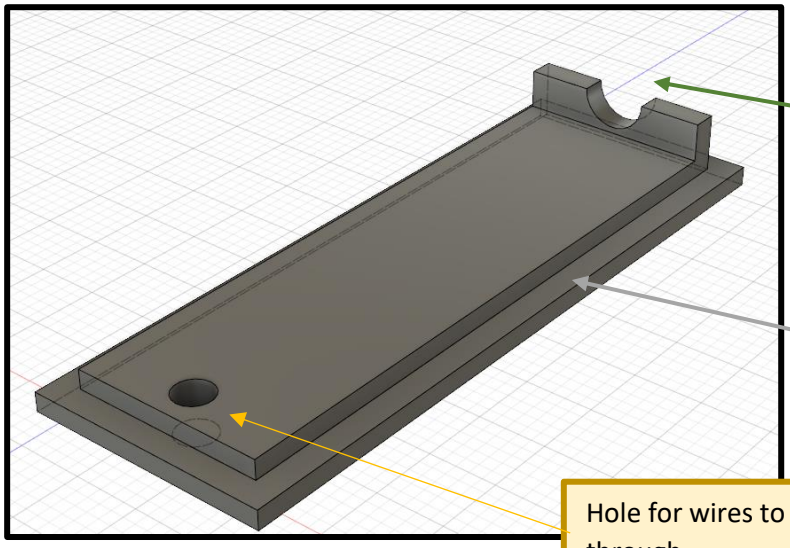
Criteria:

- Hole for the wires to be transported through
- Should be easily made waterproof
- Opening at front exposing the sensor part of the module
- A structure to keep the sensor part of the module in place
- The cover should snap onto the base

Cover:



Base:

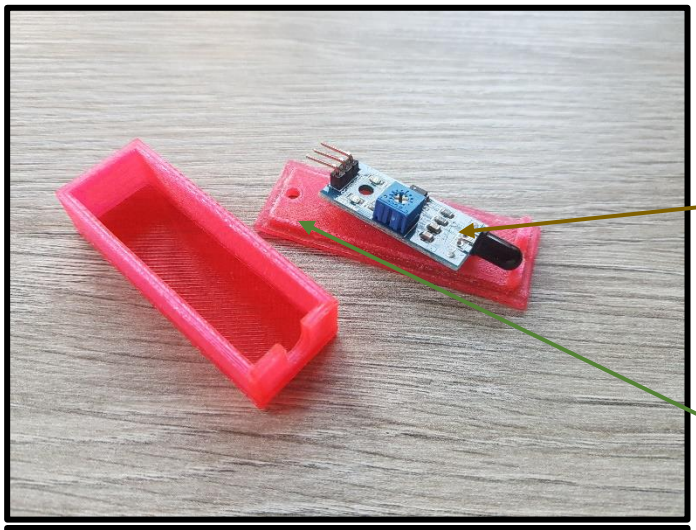


Structure to hold sensor part of module in place

Pathway where the cover snaps onto the base

Hole for wires to travel through

Final Print:



Internal Electronics

Hole for Wires

Notch for Sensor

Temperature sensor:

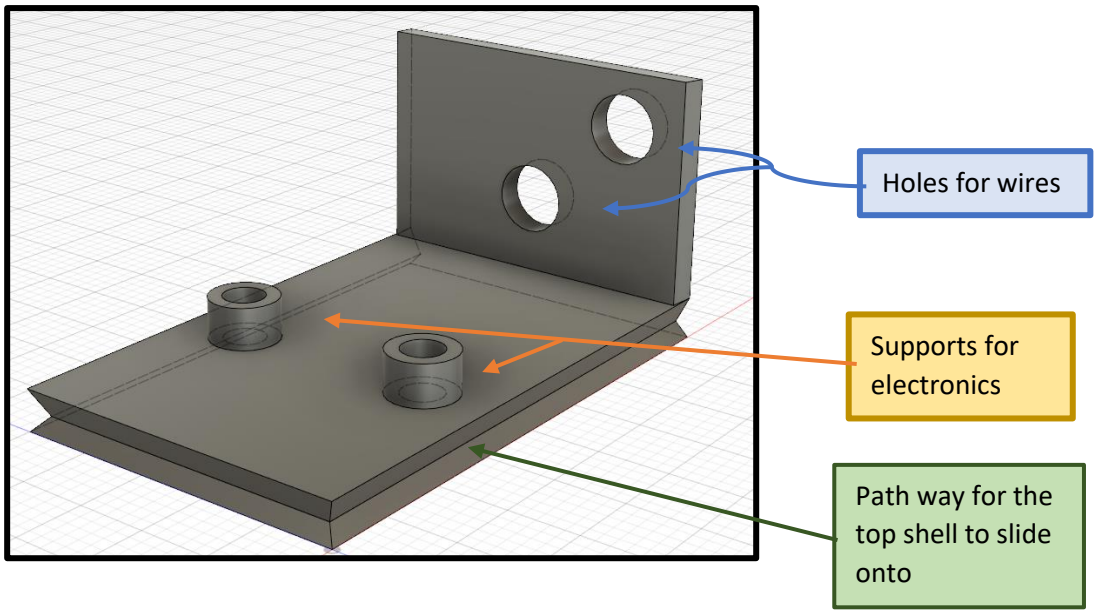
Function: This specific housing should be able to provide a structure to protect the processing unit. It should also have a clip to be able to hook up the probe to any metal surface e.g. on the roof or a gutter.

Description: Since the temperature sensor probe is water proof it doesn't need protection and that is why only a clip will be used to attach it to different surfaces. The clip will contain a magnet and a ring that the probe goes through. The actual s processing unit of the temperature sensor will be attached will be a cubic box that has a hole for the needed wires to go through. It will also have a magnet attached to its outside, so the probe can be attached to the box when there is no other available surface. This box consists of two parts: the bottom that has supports in place for the electronics and then the top which slides onto the bottom with the needed pathway

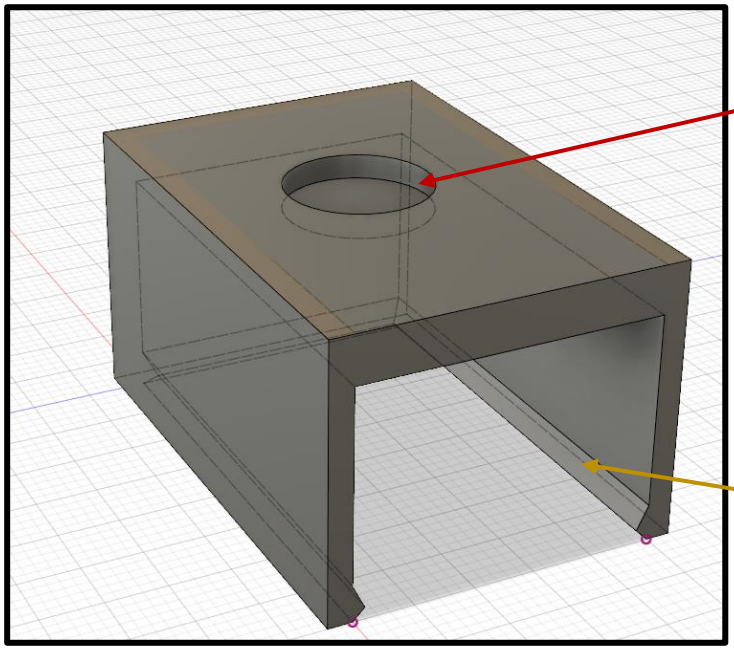
Criteria:

- Hole for the wires to move through
- Shell for the temperature sensor processing unit be easily made waterproof
- A place to glue a magnet on the shell so the probe can be attached to it
- A ring to attach to the temperature probe
- A magnet on the ring to be able to stick to various surfaces
- The top shell should be able to slide onto the bottom shell

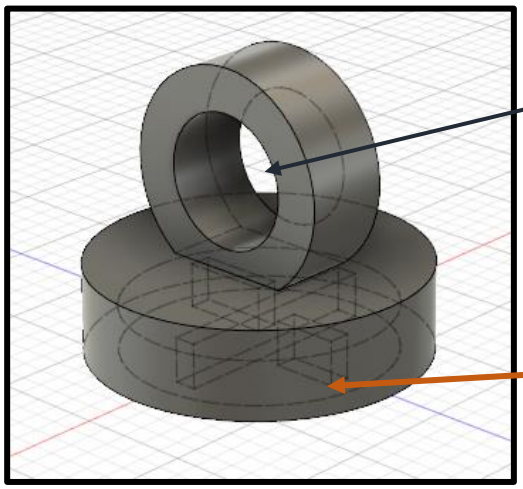
Shell Bottom:



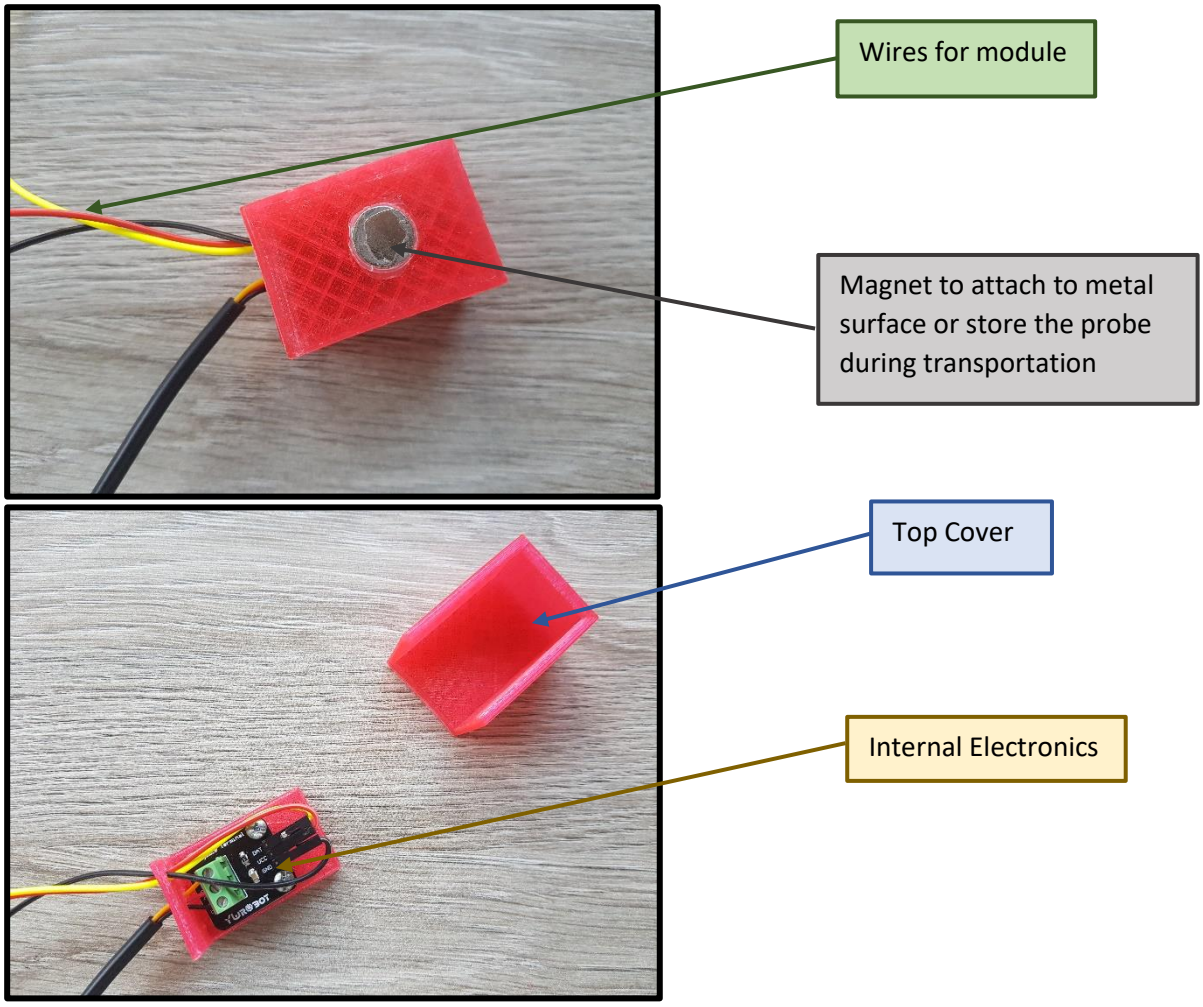
Shell Top:



Probe Ring:



Final Print:



Moisture Sensor:

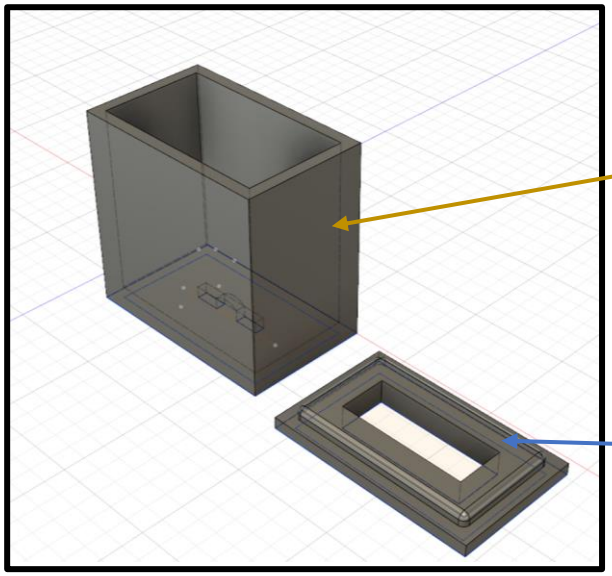
Function: The aim of the housing is to provide exit points for the legs of the moisture sensor to stick out and be able to reach the ground. It should also protect the electronics inside from rain or other elements. Furthermore, it should be of large or distinct shape, so it can be avoided when trimming/gardening or lawn mowing is taking place. The housing should also provide space where the wires can be soldered on to the processing module.

Description: This housing consists of two parts: the container and the lid. The container will have two holes at the bottom where the legs of the moisture sensors are guided through. It will have a large interior mostly since it must be large to be noticeable. It will also have a large opening at the top to provide space for wires to be soldered on to the processing unit.

Criteria:

- Space at the top to provide solder area for wires
- Should protect electronics from elements
- 2 spaces at the bottom where moisture sensor legs can be guided through
- Sizeable enough to warn gardeners of its presence

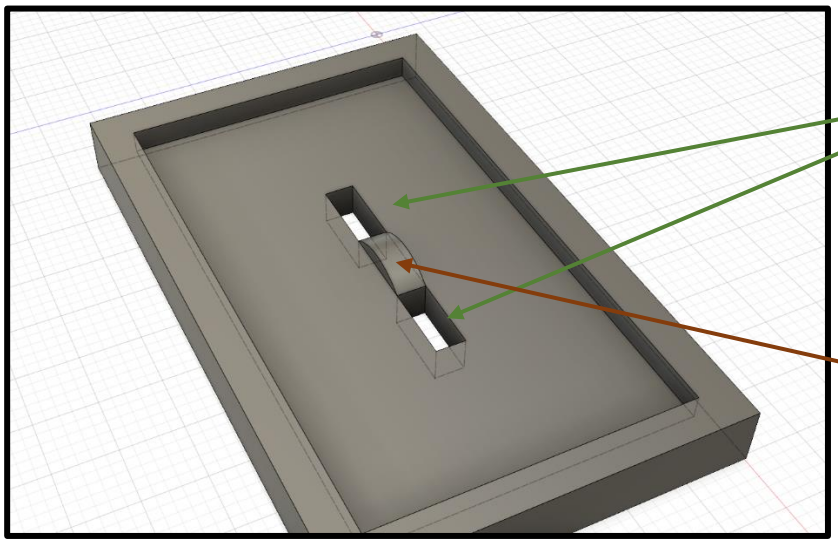
Container and Lid:



Container which houses electronics

Lid: with space on top for soldering wires

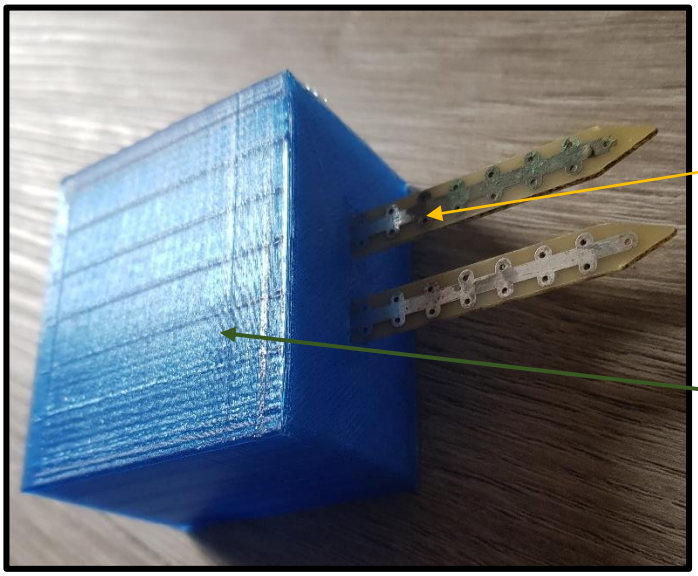
Internal View of Container:



Holes for the legs of the moisture Sensor

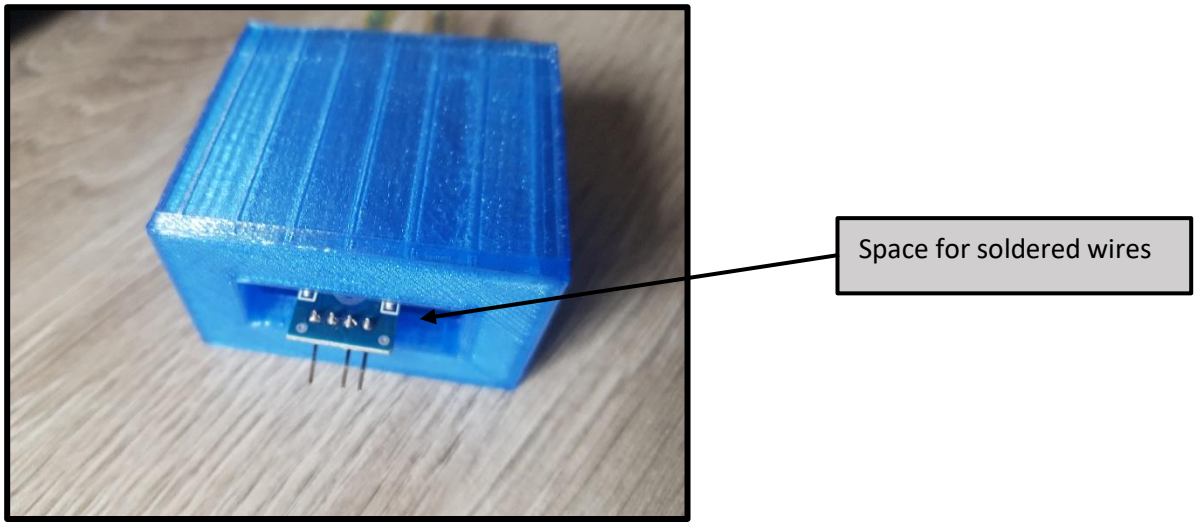
Rest for the center of the moisture sensor

Final Print:



Moisture Sensor probes that stick out at the bottom

Big and Bulky box to make it noticeable



Light Sensor:

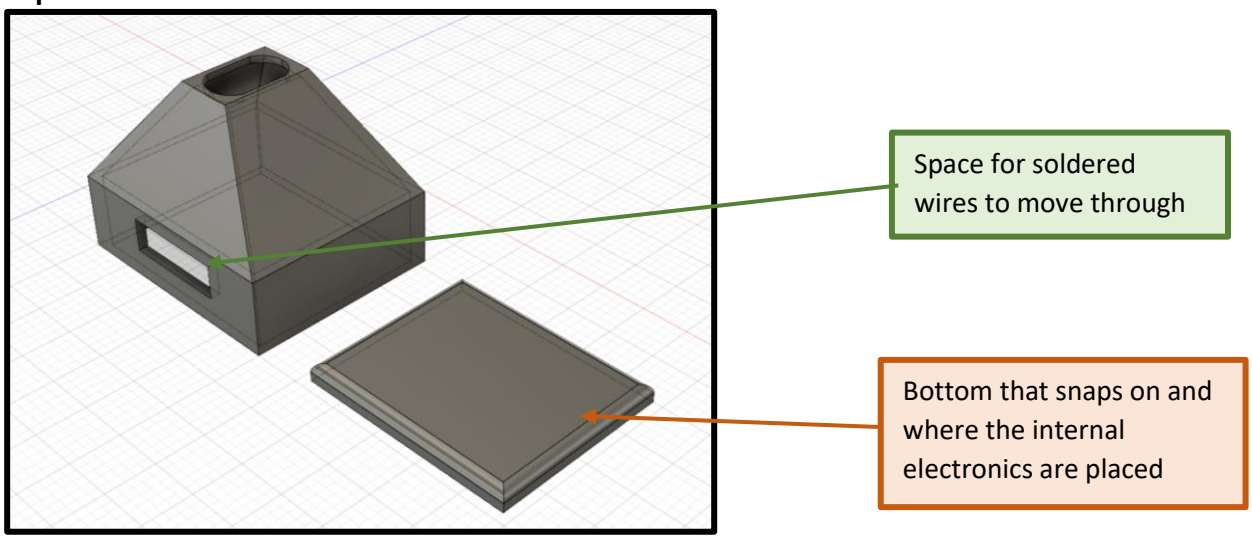
Function: The aim of the enclosure should be to allow the light sensor to stick out at the top to receive light if there is any present. It should allow any soldered wires to run through it and to the main hub via an opening or hole. It should also protect the internal electronics against the elements.

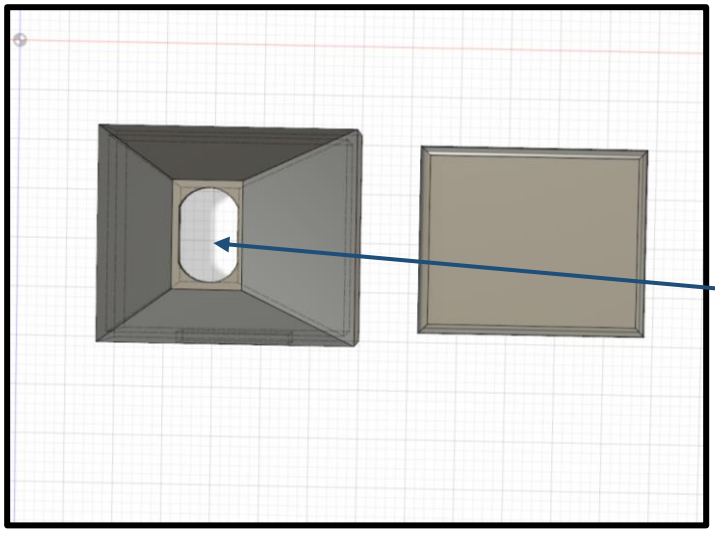
Description: The enclosure is separated into two parts: the bottom which snaps onto the top and which only has one purpose of closing the enclosure and protecting the internal electronics. And then the top which has a hole in the side for soldered wires and also decreasing in perimeter as it nears the top. The top is where the sensor sticks out to receive any present light.

Criteria:

- Space on the side for soldered wires to move through
- Protection for the internal electronics
- A hole at the top for the sensor to receive light from
- Big enough for the internal electronics to be able to fit into it.

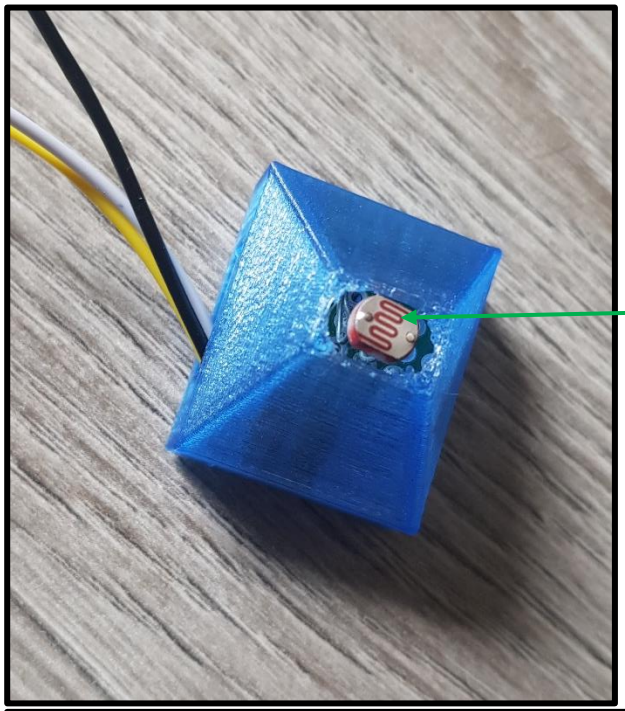
Top and Bottom:



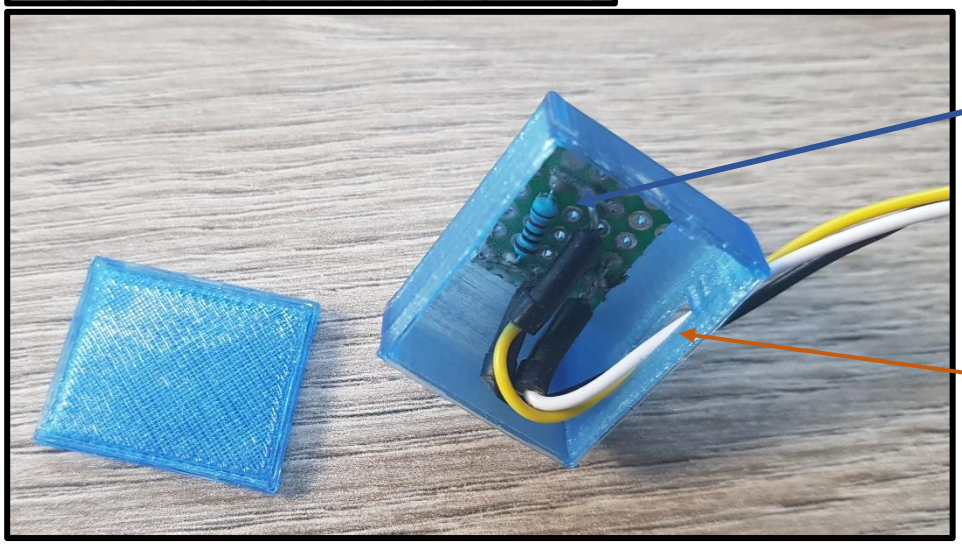


Hole on top for light to be perceived by the sensor

Final Print:



Provides hole for the light sensor to perceive light from.



Internal Electronics

Space for wires to exit

Main Hub Enclosure:

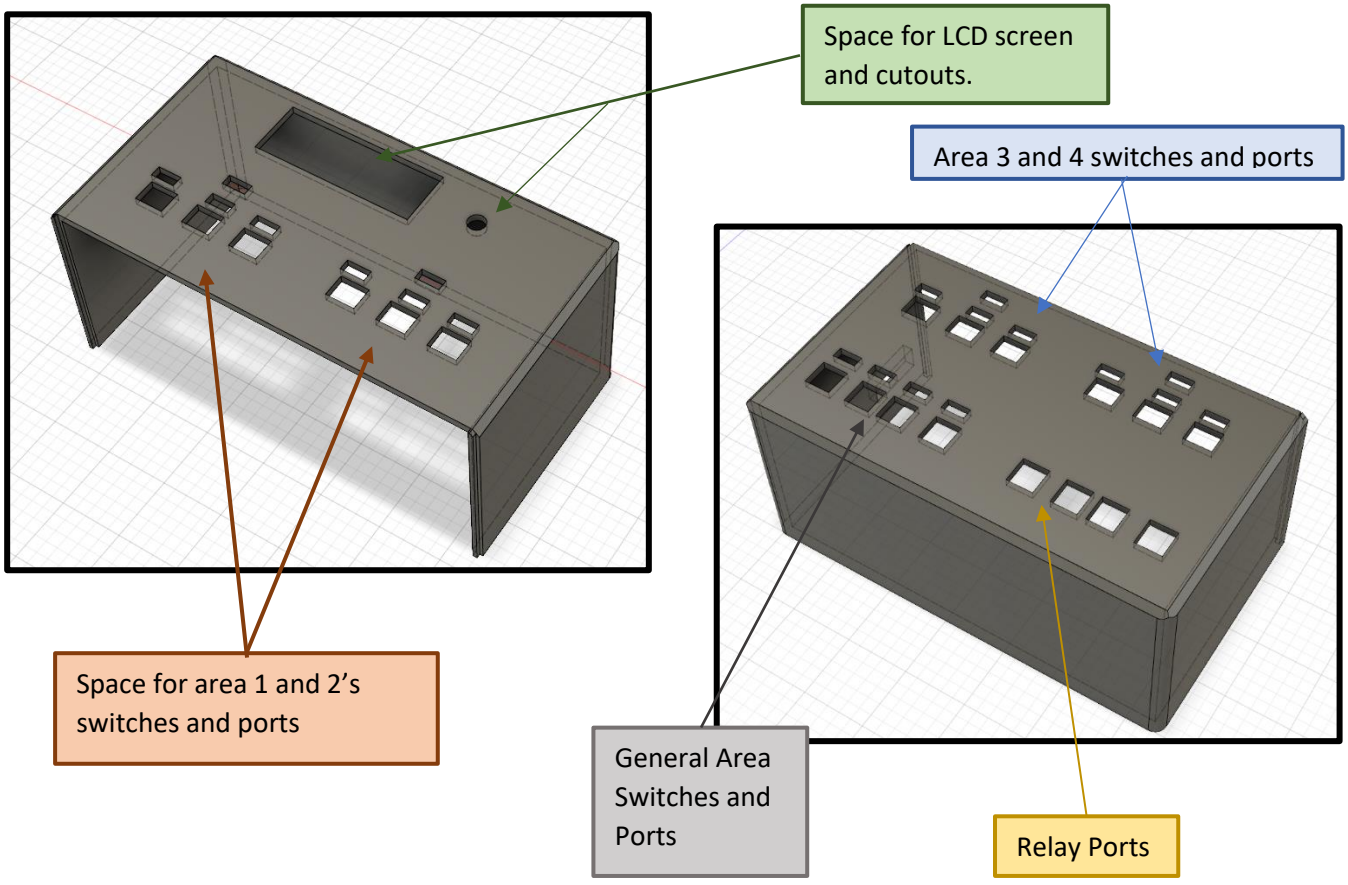
Function: The aim of this enclosure is to house all the switches and ports and make a main hub that is easy to understand and accessible by the user. It should also hold a display model, so users can retrieve information form it and therefore troubleshoot. It should allow for space to allow the necessary electronics to fit in.

Description: It is a box that has various cut-outs in its front for switches and display units. A cut out in the side for the power supply switch and then space inside for all the necessary electronics. It consists of four parts. The front side that was split in half due to restrictions with 3D printer size. And then the back cover yet again split for restrictions regarding 3D printing.

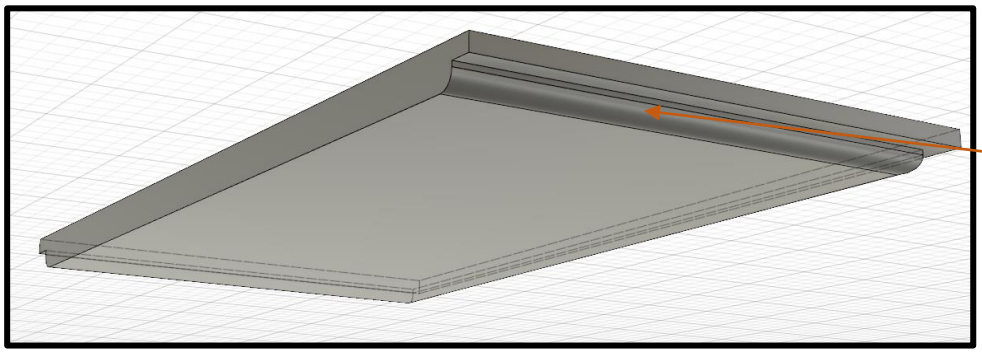
Criteria:

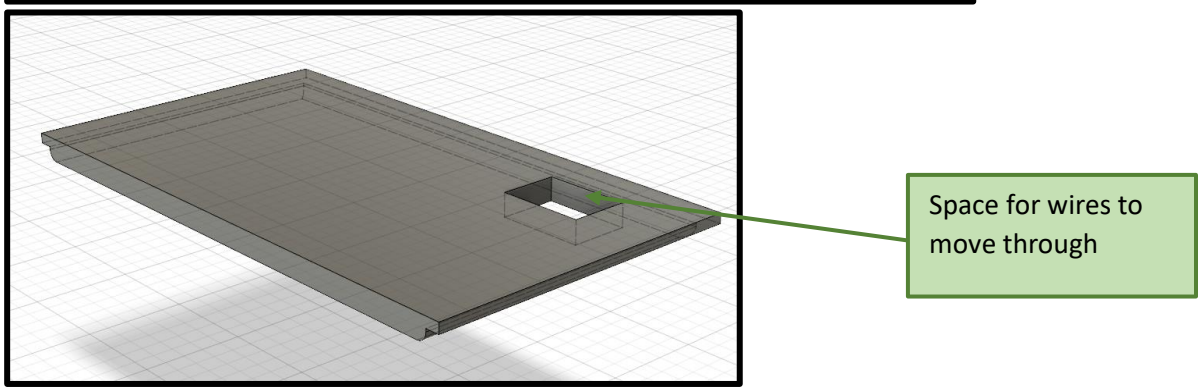
- Space for all the necessary switches
- Easy to understand and logical order
- Space for all the wire ports
- Should be large enough to house all the internal electronics of the system
- Should have space at the side for the on and off switch
- Should be able to accommodate for wires going out the back

Front (Split for printing purposes):



Back Plate (split into halves):





Program Design:

The program will be developed using the Arduino IDE which was documented in the Background Research.

Design Criteria:

Function: It will provide an interface for the user to see the state of his garden and for technical personnel to troubleshoot problems with the system. It will also monitor the state of the surrounding environment separated in Areas and control the corresponding relays to start irrigating if it is needed.

Capacity: The system will be modular, so the program needs to adapt to a varying number of sensors and devices at any given time but the maximum number of devices it will handle is the following:

- 8 moisture sensors
- 2 fire sensors
- 1 rain sensor
- 2 temperature sensors
- 17 module switches
- 4 area switches
- 4 relays
- 4 light dependant resistors
- Rotary Encoder
- I2C LCD screen

Efficiency: Speed isn't crucial in the design of this program because 2 or 3 seconds reaction time when irrigation is needed won't upset the whole environment. It also isn't needed with the LCD that

interfaces with the user because it won't be used constantly used or in high-stake situations that require millisecond reactions.

Features:

- "Fire Alarm" which warns users of a fire via the LCD display and turns on all the sprayers with the hopes that this will extinguish the fire.
- Menu that can be scrolled through using the rotary encoder and navigated with the button on the rotary encoder.
- Timers that enable screens to update every second with new information and jump out of while loops after 20 seconds to ensure that the normal functioning of the system is not affected.
- Alternating general info screens that displays different information automatically while the system is in "idle" mode.
- Able to turn on sprayers separately according to area as to provide for the least expenditure of water.

User-Interface: The UI for this system should be purely informative. The whole purpose is the project is to avoid water wastage due to user neglect therefore users shouldn't be able to change thresholds or set timers. Therefore, it will only consist of a display and rotary encoder to scroll through the available info.

Customizability: There are some restrictions, the user won't be able to change the threshold variables that the program uses to determine if there should be irrigated. This is to prevent water wastage through human error.

Modularity: The program will be divided in various components. The first being a custom library that creates instances of all the areas and then there will be dozens of custom functions to interface the data retrieved from the library and finally this data will be used in the main program to determine if irrigation is needed or not. Therefore, the program will be very modular and adaptable in the future.

Compatibility: This program is written in a variation of C++ specifically designed for Arduino and will therefore only be interface with Arduino-based devices.

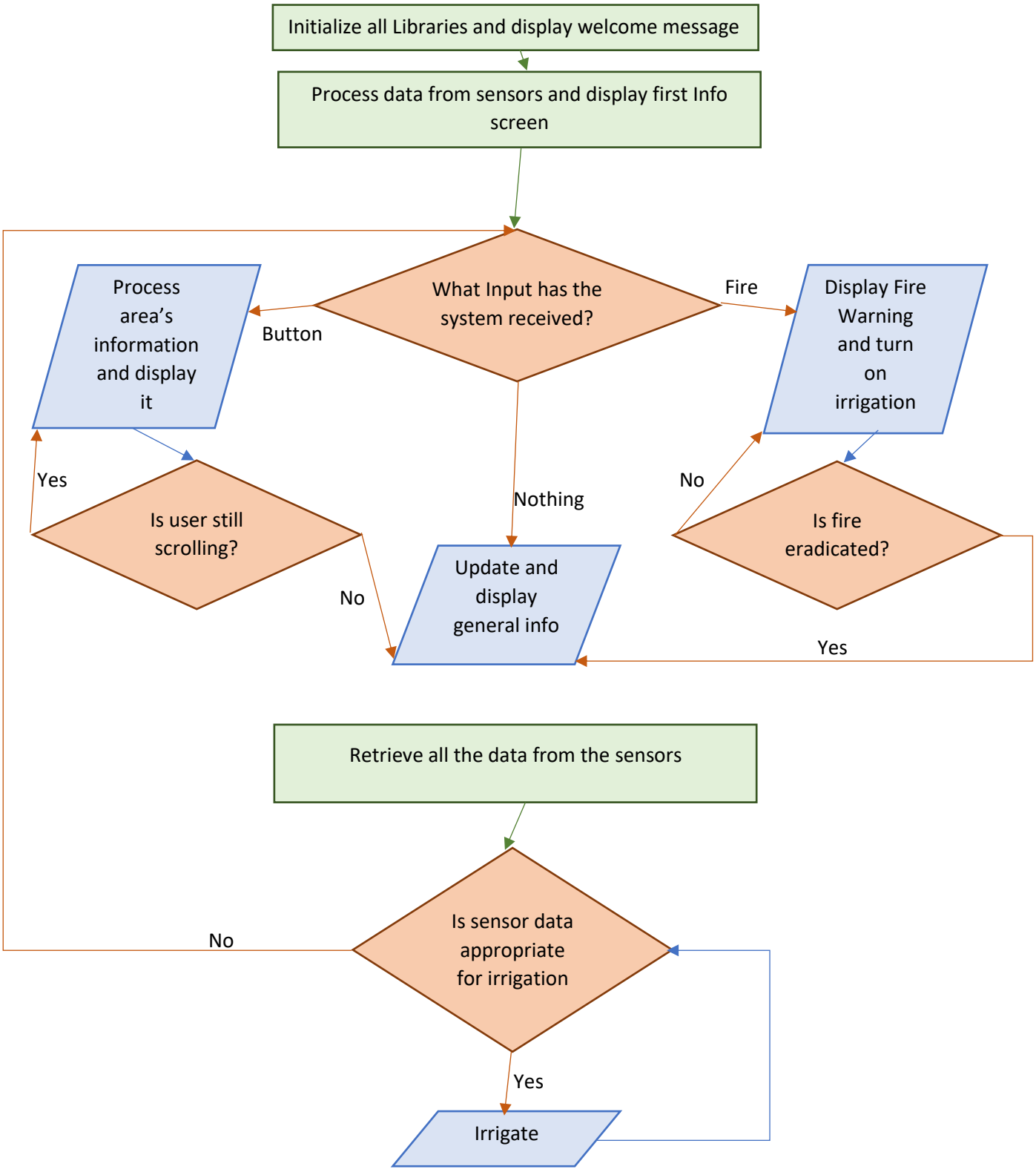
Robustness: The program should be written considering all types of input especially NaN (Not a Number) when working with float variables. Validity checks should be put in place to avoid errors or bugs when dealing with data processing.

Scalability: This program will have to be ultra-adaptive. When a user adds or removes a sensor the program must be able to process data accordingly and be able to function without out. Essentially it should be able to work and irrigate accordingly if there is only one sensor or even the maximum amount. The user is therefore able to choose whatever and however many sensors he/she believes is needed.

Flow Chart:

The following flow chart shows the basic operation of the programming from start-up.

Florivate: Flowchart



System Code:

The System Code is displayed below. It is written with the necessary comments to reveal each line's function and there is a flow chart given above to see how all the parts work together. The program consists of three parts: The Main Program, Header File of custom and library and lastly the CPP File of custom Library.

Main Program Source Code:

```
/* ---include needed lbraries--- */
#include <LiquidCrystal_I2C.h> //Library used for LCD display
#include <Wire.h> //library used to enable I2C for the lcd
#include <OneWire.h> //Library used to interface with the temperature sensor
#include <DallasTemperature.h> //Library used to retrieve and interpret data from
library
#include "sensorInterpretation.h" //custom area sensor library header file

/* ---General Sensors marked with a suffix of "_g"--- */
const byte firePin_g[2] = {2,3}; //Fire sensors using interrupt pins 2,3 so it can
halt the system
const byte rainPin_g = A0; //rain sensor uses first analog pin
#define ONE_WIRE_BUS 31 //Define Temperature sensor pin
OneWire oneWire(ONE_WIRE_BUS); //Setup one wire instance to communicate with the
temperature sensor
DallasTemperature sensors(&oneWire); //Pass one wire reference to Dallas
temperature library
const byte modSwitch_g[5] = {50,49,48,47,46}; //Temp 1 =  50 , Temp 2 = 49 , Rain
= 48 ,Fire 1 = 47 ,Fire 2 = 46

/* ---Area 1 Sensors marked with suffix of "_a1"--- */
const byte moistPin_a1[2] = {A3,A4}; //Moisture pins for area 1
const byte lightPin_a1 = A11; //Light Dependant Resistor for area 1
const byte areaSwitch_a1 = 4; //Overall Area switch for area 1
const byte modSwitch_a1[3] = {53,52,51}; //Sensor enabling switches
const byte relay_a1 = 26;//Relay pin for Area 1

/* ---Area 2 Sensors marked with suffix of "_a2"--- */
const byte moistPin_a2[2] = {A5,A6};
const byte lightPin_a2 = A12;
const byte areaSwitch_a2 = 5;
const byte modSwitch_a2[3] = {45,44,43};
const byte relay_a2 = 27;

/* ---Area 3 Sensors marked with suffix of "_a3"--- */
const byte moistPin_a3[2] = {A7,A8};
const byte lightPin_a3 = A13;
const byte areaSwitch_a3 = 6;
const byte modSwitch_a3[3] = {42,41,40};
const byte relay_a3 = 28;

/* ---Area 4 Sensors marked with suffix of "_a4"--- */
const byte moistPin_a4[2] = {A9,A10};
const byte lightPin_a4 = A14;
const byte areaSwitch_a4 = 7;
const byte modSwitch_a4[3] = {39,38,37};
const byte relay_a4 = 29;

const byte relays[4] = {26 ,27 ,28 ,29}; //an array to simply interface with al
the relays
```

```
const byte rotary[3] = {19,18,17}; //rotary encoder pins: 19 = button , 18 =
Clock, 17 = Data
byte screen = 1; //screen selector for the area updater
byte genScreen = 1; //Screen selector for the general info display
unsigned long timeS = 0; //This is used to determine when to update the menu
screen
unsigned long timeOutVar = 0; //Lock out variable to ensure that menu isn't left
on disabling the rest of the system
unsigned long genTimeUp = 0; //Time till the general gets changed to alternate
information
unsigned long genInfoTime = 0; //Time till the current info on the general screen
gets updated
boolean clockS; //state of the clock pin e.g HIGH or LOW
boolean dataS; //state of the data pin e.g HIGH or LOW

/* ---Threshold Variables--- */
float tempThresh = 25; //Temperature should be less than this amount
int rainThresh = 300; //should be considered raining if it is less than this
threshold

/* ---variables to keep track of system and check where it is irrigating--- */
boolean isWatering[4] = {false, false ,false ,false};

/* ---Initialize all the areas in garden as objects of the custom library for
sensor interpretation--- */
//Per area the following info is given: The 1st moisture pin, its corresponding
switch ,2nd moist ,corresponding switch ,light sensor, corresponding switch
sensorInterpretation area_1(moistPin_a1[0],modSwitch_a1[0] ,moistPin_a1[1]
,modSwitch_a1[1] , lightPin_a1 , modSwitch_a1[2]);
sensorInterpretation area_2(moistPin_a2[0],modSwitch_a2[0] ,moistPin_a2[1]
,modSwitch_a2[1] , lightPin_a2 , modSwitch_a2[2]);
sensorInterpretation area_3(moistPin_a3[0],modSwitch_a3[0] ,moistPin_a3[1]
,modSwitch_a3[1] , lightPin_a3 , modSwitch_a3[2]);
sensorInterpretation area_4(moistPin_a4[0],modSwitch_a4[0] ,moistPin_a4[1]
,modSwitch_a4[1] , lightPin_a4 , modSwitch_a4[2]);

LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE); //initalize
instance of the lcd library
void setup(){
  Serial.begin(9600); //Start Serial Communication at a baud rate of 9600
  startupLCD(); //Start custom function that configures LCD
  sensors.begin();
  for(int i = 0; i < 4; i ++){
    if(i < 3){ //if for loop is num 0-2
      digitalWrite(rotary[i] , HIGH); //Activate pins on rotary encoder
    }
    pinMode(relays[i], OUTPUT); //Set relay pins to Output
  }
  attachInterrupt(digitalPinToInterrupt(rotary[1]), clockIn, RISING); //attach in
interrupt to enable rotary encoder to function properly
  delay(2000); //Delay program so welcome screen is readable
  generalUpdate(genScreen); //Update lcd screen to first general update screen
  genTimeUp = millis(); //Reset screen change time to current program run time
  genInfoTime = millis(); //Reset screen update time to current program run time
}

void loop() {
  if((millis()-genInfoTime)/1000>1){    //updates screen and checks irrigation
every second
    irrigate(); //start custom function to check what areas need to be irrigated
    FIRE(); //start custom function to check if there is a fire and deal with it
accordingly
    if((millis()-genTimeUp)/1000>20){ //changes displayed screen every 20 seconds
      if(genScreen>1){
```

```
        genScreen = 1; //if the screen is more than one (2) change to 1
      }else{
        genScreen = 2; //otherwise if screen 1 than change ot screen 2
      }
      genTimeUp = millis(); //rest the variable
    }
    generalUpdate(genScreen); //update the screen after potential adjustments have
been made
    genInfoTime = millis();  //reset the variable
  }
  menu(); //start custom function to check if user wants to scroll throough the
area menu

}

/* ---Function to configure LCD upon startup--- */
void startupLCD(){
  lcd.begin(16,2); //begin LCD 16 characters in width and 2 rows in height
  lcd.backlight(); //turn on the LCD backlight
  lcd.clear(); //clear screen if it was previously populated
  lcd.setCursor(3,0); //set cursor to the third character on the first row
  lcd.print("Welcome to"); //display string on LCD
  lcd.setCursor(4,1);
  lcd.print("iGarden");
}
/* ---Function to calculate the average moisture across all sensors--- */
int moistTotal(){
  int total = 0; //variable to store the total moisture
  byte count = 0; //variable to count the amount of sensors checked
  if(digitalRead(areaSwitch_a1)){ //only execute if the area switch is turned on
    total+= area_1.moistAverage(); //read the amount of moisture and add it to
total
    count ++; //increment count variable
  }
  if(digitalRead(areaSwitch_a2)){
    total+= area_2.moistAverage();
    count ++;
  }
  if(digitalRead(areaSwitch_a3)){
    total+= area_3.moistAverage();
    count ++;
  }
  if(digitalRead(areaSwitch_a4)){
    total+= area_4.moistAverage();
    count ++;
  }
  return total/count; //get the average by dividing the total moisture with the
amount of active sensors
}
/* ---Function that updates the LCD screen with info of the Area specified by the
passed num variable--- */
void areaUpdate(byte num){
  int moistValue = 0; //variable to store the moisture percentage of the area
  switch(num){ //check which area's info is desired
    case 1: //info about area 1 is desired
      if(digitalRead(areaSwitch_a1)){ //check if Area 1's switch is on
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Area 1 :");
        lcd.setCursor(0,1);
        lcd.print("M:");
        lcd.setCursor(3, 1);
        moistValue = area_1.moistAverage(); //populate the variable with custom
moisture function
```

```
        lcd.print(moistValue); //print the moisture percentage for the area
        if(moistValue <10){
          lcd.setCursor(4,1); //if moist percentage is only 1 digit display %
symbol at character 4
        }else if(moistValue< 100 && moistValue > 9){
          lcd.setCursor(5,1); //if moist average is 2 digits display % symbol at
character 5
        }else{
          lcd.setCursor(6,1); //if moist average is 3 digits display % symbol at
character 6
        }
        lcd.print("%");
        lcd.setCursor(8 ,1);
        lcd.print("L:");
        lcd.setCursor(10,1);
        lcd.print(area_1.timeOfDay()); //print what the time of day is for
specific area with custom function
      }else{ //if area switch is OFF print that the area is offline
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Area 1 :");
        lcd.setCursor(3,1);
        lcd.print("Offline");
      }
    break;
    case 2: //info for area 2 is desired
      if(digitalRead(areaSwitch_a2)){
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Area 2 :");
        lcd.setCursor(0,1);
        lcd.print("M:");
        lcd.setCursor(3, 1);
        moistValue = area_2.moistAverage();
        lcd.print(moistValue);
        if(moistValue <10){
          lcd.setCursor(4,1);
        }else if(moistValue< 100 && moistValue > 9){
          lcd.setCursor(5,1);
        }else{
          lcd.setCursor(6,1);
        }
        lcd.print("%");
        lcd.setCursor(8 ,1);
        lcd.print("L:");
        lcd.setCursor(10,1);
        lcd.print(area_2.timeOfDay());
      }else{
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Area 2 :");
        lcd.setCursor(3,1);
        lcd.print("Offline");
      }
    break;
    case 3: //info for area 3 is desired
      if(digitalRead(areaSwitch_a3)){
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Area 3 :");
        lcd.setCursor(0,1);
        lcd.print("M:");
        lcd.setCursor(3, 1);
        moistValue = area_3.moistAverage();
```

```
        lcd.print(moistValue);
        if(moistValue <10){
          lcd.setCursor(4,1);
        }else if(moistValue< 100 && moistValue > 9){
          lcd.setCursor(5,1);
        }else{
          lcd.setCursor(6,1);
        }
        lcd.print("%");
        lcd.setCursor(8 ,1);
        lcd.print("L:");
        lcd.setCursor(10,1);
        lcd.print(area_3.timeOfDay());
      }else{
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Area 3 :");
        lcd.setCursor(3,1);
        lcd.print("Offline");
      }
    break;
    case 4: //info for area 4 is desired
      if(digitalRead(areaSwitch_a4)){
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Area 4 :");
        lcd.setCursor(0,1);
        lcd.print("M:");
        lcd.setCursor(3, 1);
        moistValue = area_4.moistAverage();
        lcd.print(moistValue);
        if(moistValue <10){
          lcd.setCursor(4,1);
        }else if(moistValue< 100 && moistValue > 9){
          lcd.setCursor(5,1);
        }else{
          lcd.setCursor(6,1);
        }
        lcd.print("%");
        lcd.setCursor(8 ,1);
        lcd.print("L:");
        lcd.setCursor(10,1);
        lcd.print(area_4.timeOfDay());
      }else{
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Area 4 :");
        lcd.setCursor(3,1);
        lcd.print("Offline");
      }
    break;
  }
}
/* ---Function that updates the LCD screen with general info according to the num
area--- */
void generalUpdate(byte num){
  int moistValue = 0; //initialize a variable to store moist percentage
  String temperature = getTemp(); //check temperature and populate temperature
variable already so there isn't any lag
  switch(num){ //checks what general screen is desired
    case 2: //screen 2 is desired and displayed on LCD
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print("General:");
```

```
      lcd.setCursor(0,1);
      lcd.print("M:");
      lcd.setCursor(3, 1);
      if(moistTotal() == 0){ //If there is 0% moisture populate moistValue
variable with "OFF"
        moistValue = "OFF";
      }else{ //otherwise make the variable equal to the moist percentage
        moistValue = moistTotal();
      }
      lcd.print(moistValue); //display the contents of the moist value variable
      if(moistValue <10){
        lcd.setCursor(4,1);
      }else if(moistValue< 100 && moistValue > 9){
        lcd.setCursor(5,1);
      }else{
        lcd.setCursor(6,1);
      }
      lcd.print("%");
      lcd.setCursor(8 ,1);
      lcd.print("L:");
      lcd.setCursor(10,1);
      lcd.print(dayORnight()); //display if it is night or day overall according
to all the LDR's
    break;
    case 1: //general screen 1 is desired and displayed on LCD
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print("General:");
      lcd.setCursor(0,1);
      lcd.print("R:");
      lcd.setCursor(3, 1);
      if(isRaining()){ //check if it is raining with this custom function
        lcd.print("!"); //if it is then print ! symbol
      }else{
        lcd.print("X"); //if it isn't then print X symbol
      }
      lcd.setCursor(6 ,1);
      lcd.print("T:");
      lcd.setCursor(9,1);
      lcd.print(temperature); //display the contents of temperature variable
    break;
  }
}
/* ---Function to check if it is day or night across all variable--- */
String dayORnight(){
  int days = 0; //Counter for the amount of sensors that output that is day time
  int count = 0; //Counter for the amount of active sensors
  if(digitalRead(areaSwitch_a1)){ //check if the specific area is active
    count++; //if it is active increment count variable
    if(area_1.timeOfDay().equals("Day")){ //if the time of day for specific area
is day than increment day counter
      days ++;
    }
  }
  if(digitalRead(areaSwitch_a2)){
    count++;
    if(area_2.timeOfDay().equals("Day")){
      days ++;
    }
  }
  if(digitalRead(areaSwitch_a3)){
    count++;
    if(area_3.timeOfDay().equals("Day")){
      days ++;
```

```
    }
  }
  if(digitalRead(areaSwitch_a4)){
    count++;
    if(area_4.timeOfDay().equals("Day")){
      days ++;
    }
  }
  if(days > 0 || count == 0){ //if there are no sensors or at least one outputs
day
    return "Day"; //then return that it is day
  }else{
    return "Night"; //otherwise return that is night time
  }
}
/* ---Function that controls the navigation of the Area Info Menu--- */
void menu(){
  boolean butt = digitalRead(rotary[0]); //check the state of rotary button
(Pressed or Released)
  if(butt == false){ //If it is pressed enter the menu and initially inform user
to release the button
    lcd.clear();
    lcd.setCursor(0 ,0);
    lcd.print("Getting Info,");
    lcd.setCursor(0 ,1);
    lcd.print("Release Button");
    screen = 1; //reset screen to the 1st area
    int temp = 5; //Temporary variable that will be used to see if there is change
in screen
    delay(2000); //Delay gives users the chance to release button
    butt = true; //Turn the button state OFF
    timeS = millis(); //reset area updating timer to current program runtime
    timeOutVar = millis(); //reset the kickout variable to the current program
runtime
    while(butt == true && (millis()-timeOutVar)/1000<60){ // checks if button is
still unpressed and that 60 seconds has not passed
      if(screen != temp){
        timeOutVar = millis(); //when screen changes variable resets so that users
are not locked out while still navigating
      }
      if(screen!=temp || (millis()- timeS)/1000> 1){ //check if the selected
screen is not already displayed and if at least one second has passed.
        areaUpdate(screen); //then update screen
        timeS = millis(); //and reset to area updating timer to current program
runtime
      }
      temp = screen; //change temporary variable to current screen
      butt = digitalRead(rotary[0]); //check if the button is being pressed
    }
    //display message stating that menu has been exited and that button should be
released
    lcd.clear();
    lcd.setCursor(0 ,0);
    lcd.print("Going back,");
    lcd.setCursor(0 ,1);
    lcd.print("Release Button");
    delay(1000); //delay gives user chance to release button
  }
}
/* ---Interrupt function that increments the screen variable according to the
rotary encoder--- */
void clockIn(){
  noInterrupts();// stop any more signals from interfering
  dataS = digitalRead(rotary[2]); //check the state of the data pin
```

```
  clockS = digitalRead(rotary[1]); //check the state of the clock pin
  if(dataS == clockS){ //if data is equal to clock than decrement screen variable
    if(screen <= 1){ //if the screen is already at 1st screen than change to 4th
screen
      screen=4;
    }else{
      screen -= 1; //otherwise decrement as normal
    }
  }else{ //if data is not equal to clock than increment screen variable
    if(screen >= 4){ //if the screen is already at 4 go to 1st screen
      screen = 1;
    }else{
      screen += 1; //otherwise increment as normal
    }
  }
  delay(100);//smoothen the code out
  interrupts();// enable interrupts again
}
/* ---Function that gets the average temperature and converts it to a string--- */
String getTemp(){
  float tempTotal = 0; //stores the total addition of temperatures
  byte count = 0; //counts the total amount of active sensors
  if(digitalRead(modSwitch_g[0])){
    sensors.requestTemperatures(); //request the temperatures
    tempTotal= sensors.getTempCByIndex(0); //add temperature outputted to the
tempTotal variable
    count++; //increment the count variable
  }
  if(count == 0){
    return "N/A"; //if there aren't any sensors active return that information is
"N/A"
  }
  return String(tempTotal); //otherwise return the average temperature in string
format

}
/* ---Function that checks if temperature is low enough for irrigation to take
place--- */
boolean tempReady(){
  String temp = getTemp(); //get the average temperature
  if(temp.equals("N/A")){
    return true; //if there aren't any sensors present return that irrigation may
take place
  }else{
    if(temp.toFloat() < tempThresh){
      return true; //if the temperature is lower than the maximum return that
irrigation may take place
    }else{
      return false; //if the temperature is too high return that irrigation may
not take place
    }
  }
}
/* ---Function that checks if it is raning--- */
boolean isRaining(){
  if(digitalRead(modSwitch_g[2])){ //check if the rain sensor is active
    if(analogRead(rainPin_g) < rainThresh){
      return true; //if the rain pin returns a lower value than the threshold
return that it is raining
    }else{
      return false; //it is higher return that it isn't raining
    }
  }else{
    return false; //if the sensor isn't active than return that it isn't raining
```

```
  }
}
/* ---Function that checks all the general sensors for if irrigation is possible--
- */
boolean generalReady(){
  if(tempReady() && !(isRaining())){
    return true; //if the temperature is low enough and it isn't raining return
that irrigation may take place
  }else{
    return false; //otherwise return that irrigation shouldn't take place
  }
}
/* ---Function that controls all the relays seperate according to their
corresponding environment/area--- */
void irrigate(){
  if(generalReady()){ //first check if all the general sensors allow irrigation to
take place
    //Check Area 1
    if(digitalRead(areaSwitch_a1)){ //check if area is active
      if(area_1.waterReady()){ //check if the area needs to be irrigated
        if(isWatering[0] == false){
          digitalWrite(relay_a1, HIGH); //if it isn't currently being irrigated
then turn the relay on
          isWatering[0] = true; //set corresponding variable to it is busy
irrigating
        }
      }else{
        if(isWatering[0]){ //if the area doesn't need to be irrigated than first
check if it is currently irrigating
          digitalWrite(relay_a1, LOW); //then turn off the corresponding relay
          isWatering[0] = false; //and set the isWatering variable to it isn't
busy irrigating
        }
      }
     }else{
      if(isWatering[0]){ //if the area is set to inactive check if it was busy
irrigating
        digitalWrite(relay_a1, LOW); //then turn the corresponding relay off
        isWatering[0] = false; //and set isWatering variable to not irrigating
      }
    }
    //Check Area 2
    if(digitalRead(areaSwitch_a2)){
      if(area_2.waterReady()){
        if(isWatering[1] == false){
          digitalWrite(relay_a2, HIGH);
          isWatering[1] = true;
        }
      }else{
        if(isWatering[1]){
          digitalWrite(relay_a2, LOW);
          isWatering[1] = false;
        }
      }
     }else{
      if(isWatering[1]){
        digitalWrite(relay_a2, LOW);
        isWatering[1] = false;
      }
    }
    //Check Area 3
    if(digitalRead(areaSwitch_a3)){
      if(area_3.waterReady()){
        if(isWatering[2] == false){
```

```
          digitalWrite(relay_a3, HIGH);
          isWatering[2] = true;
        }
      }else{
        if(isWatering[2]){
          digitalWrite(relay_a3, LOW);
          isWatering[2] = false;
        }
      }
    }else{
      if(isWatering[2]){
        digitalWrite(relay_a3, LOW);
        isWatering[2] = false;
      }
    }
    //Check Area 4
    if(digitalRead(areaSwitch_a4)){
      if(area_4.waterReady()){
        if(isWatering[3] == false){
          digitalWrite(relay_a4, HIGH);
          isWatering[3] = true;
        }
      }else{
        if(isWatering[3]){
          digitalWrite(relay_a4, LOW);
          isWatering[3] = false;
        }
      }
    }else{
      if(isWatering[3]){
        digitalWrite(relay_a4, LOW);
        isWatering[3] = false;
      }
    }
  }else{ //if the general sensors don't allow the garden to irrigate
    for(int i = 0 ; i < 4; i ++){ //use for loop to individually check if an area
was busy irrigating and then turn off corresponding relay
      if(isWatering[i]){
        digitalWrite(relays[i], LOW);
        isWatering[i] = false; //then set corresponding variable to not irrigating
      }

    }
  }
}
/* ---Function that checks if there is an fire and then acts accordingly--- */
void FIRE(void){
  boolean switch1 =digitalRead(modSwitch_g[3]); //capture the state of the first
fire switch
  boolean switch2 = digitalRead(modSwitch_g[4]); //capture the state of the second
  if(switch1 || switch2){ //if either of the fire switches are active execute
following code
    /*The while loop requires the following conditions:
     * 1*--- that the first fire sensor shows that there is a fire and for the
fire sensor's corresponding switch to be active
     * 2*--- or that the same apllies to the scond fire sensor
     */
    while((switch1 && digitalRead(firePin_g[0])== false) || (switch2 &&
digitalRead(firePin_g[1])== false)){ //this will repeat till the conditions arent
met(when there isn't a fire)
      //print fire warning
      lcd.clear();
      lcd.setCursor(3 ,0);
      lcd.print("FIRE!!!!!");
```

```
      lcd.setCursor(3 ,1);
      lcd.print("Get Help!");
      for(int i = 0 ; i < 4; i ++){
        if(isWatering[i] == false){//checks if the sprayer was already irrigating
          digitalWrite(relays[i], HIGH); //if it wasn't then the relay is turned
on
          isWatering[i] = true; //and the corresponding isWatering variable shows
it is irrigating
        }

      }
      delay(700);//make warning text readable
    }
  }
}
```

Sensor Interpretation Custom Library Source Code (Header File):

```
/* ---Start header file--- */
#ifndef sensorInterpretation_h
#define sensorInterpretation_h

#include "Arduino.h"//inlude basic functions and syntax from the arduino library

class sensorInterpretation{
  //declare all the public functions
  public:
    sensorInterpretation(byte ,byte ,byte ,byte ,byte ,byte);
    sensorTester();
    int moistAverage();
    String timeOfDay();
    boolean waterReady();
    //declare all the private variables and functions only used within the library
  private:
    byte _moist1;
    byte _moist2;
    byte _switch1;
    byte _switch2;
    byte _switch3;
    byte _light;
    boolean moistReady(byte);
    boolean lightReady();
};


#endif //end header file
```

Sensor Interpretation Custom Library Source Code (CPP File):

```
#include "Arduino.h" //insert library with all the main arduino functions
#include "sensorInterpretation.h" //insert header file of the library

//main function of the library where all the library variables get passed values
sensorInterpretation::sensorInterpretation(byte moist1, byte switch1 , byte moist2
, byte switch2 , byte light , byte switch3){
  /* ---The library assigns varaibles the values passed to it from the main
program--- */
   _moist1 = moist1;
   _switch1 = switch1;
   _moist2 = moist2;
   _switch2 = switch2;
   _light = light;
   _switch3 = switch3;
}
```

```
/* ---Function that helps to troubleshoot system through Serial Monitor when
something goes wrong--- */
sensorInterpretation::sensorTester(){
  if(digitalRead(_switch1)){ //checks if the sensor is active
    Serial.print("Moist 1 : "); //prints the name of the sensor to serial monitor
    Serial.println(analogRead(_moist1)); //prints the value of the sensor
  }else{
    Serial.println("Moist 1 is deactivated"); //if the sensor is deactivated it
prints this to the serial monitor
  }
  //the previous code is repeated for all the sensors in the class
  if(digitalRead(_switch2)){
    Serial.print("Moist 2 : ");
    Serial.println(analogRead(_moist2));
  }else{
    Serial.println("Moist 2 is deactivated");
  }
  if(digitalRead(_switch3)){
    Serial.print("Light : ");
    Serial.println(analogRead(_light));
  }else{
    Serial.println("Light is deactivated");
  }
}
/* ---Function that returns the average of the moisture sensors--- */
int sensorInterpretation::moistAverage(){
  double totalMoist = 0.0; //Variable to store the total amount of moisture
  byte count = 0; //variable that increments with the amount of active sensors
  if(digitalRead(_switch1)){ //checks if first moisture sensor is active
    totalMoist += float(map(analogRead(_moist1),1023 ,0,0 ,1023)); //it reverses
the value and adds it to the total moisture
    count ++; //increments the amount of active sensors
   }
   if(digitalRead(_switch2)){
     totalMoist +=float(map(analogRead(_moist2),1023 ,0,0 ,1023));
     count++;
    }
  return (totalMoist/(1023*count))*100; //returns the moisture percentage (total
moisture divided by the number of active sensors
}
/* ---Function that checks what the time of day is--- */
String sensorInterpretation::timeOfDay(){
  if(digitalRead(_switch3)){ //checks if the light sensor is active
    int sensVal = analogRead(_light); //the amount the sensor outputs is stored in
a variable
    if(sensVal > 550){ //if the value is more than 550 than it is day
      return "Day"; //return the string day
    }else{
      return "Night"; //if it is less than return the night
    }
  }else{ //if the sensor is inactive return Not Available
    return "N/A";
  }
}
/* ---Function that checks if the area needs to be irrigated or not--- */
boolean sensorInterpretation::waterReady(){
  if(moistReady(1) && moistReady(2) && lightReady()){ //if all the sensors meet
the conditions:
    return true; //return true that this area should be irrigated
  }else{
    return false; //return false ,this area shouldn't be irrigated
  }
}
```

```
/* ---Function that checks if the moisture sensors determined that the area should
be irrigated--- */
//function takes the argument of num which determines which sensor is tested
boolean sensorInterpretation::moistReady(byte num){
  switch(num){
    //sensor 1 is tested
    case 1:
      if(digitalRead(_switch1)){ //checks if sensor is active
        if(analogRead(_moist1) > 450){
          return true; //if sensor outputs more than 450 return true
        }else{
          return false; //if sensor outputs less than 450 than return false
        }
      }else{
        return true; //if sensor is inactive than return false
      }
    break;
    //test sensor 2
    case 2:
      if(digitalRead(_switch2)){
        if(analogRead(_moist2) > 450){
          return true;
        }else{
          return false;
        }
      }else{
        return true;
      }
    break;
  }
}
/* ---Function that checks if it is the appropriate time of day to irrigate--- */
boolean sensorInterpretation::lightReady(){
  if(digitalRead(_switch3)){ //checks if light sensor is active
    if(analogRead(_light) < 550){
      return true;  //if sensor outputs less than 550 then return true
    }else{
      return false; //if sensor outputs more than 550 then return false
    }
  }else{
    return true; //if sensor is inactive then return true
  }
}
```

LCD Display Sequence:

Firstly, the user is greeted with a welcome message upon start-up this is displayed while the system sets everything up.

Then the general screen is displayed showcasing all the data on the system, this updates every second with real time information. Though when 20 seconds has passed without any human input or fires then the screen switches to an alternate general screen with other information on it.

General Screen 1:
Rain and Temperature

General Screen 2:
Moisture Level and Time of Day

When the rotary button is pressed on the main hub the system will first alert user that it was pressed and then follow with the Area screens with all the individual information displayed and the user can also then scroll through all the areas and just press the button when they are done. If no scrolling is present for at least 60 seconds, then the screen will revert back to general area information.

And finally, when there is a fire detected it will display this message while also turning on the sprayers.

Test Plan:

Electronics:

1. Check that the power supply is outputting correct voltage by measuring it with a multimeter.
2. Ensure that all the wires are soldered correctly by doing continuity test with a multimeter on all the connections
3. Ensure that the relays are functioning properly by connecting LEDs to it and turning them on and off manually.
4. Ensure that all the modules are functioning by referring to their onboard LED lights
5. Make sure that the LCD is connected properly by checking that the backlight is lit when it is fed power.
6. Check that switch is functioning properly by making sure the complete turns off and on with the movement of the switch.

3D-Printed Housings:

1. Make sure that water sensitive electronics are covered by a 3D printed housing that disallows water to enter.
2. Ensure that all the switches and ports are open to the surface of the housing and usable by the user.
3. Ensure that any visual surface (Like the LCD screen) is not covered by 3D-printed material.

Programming:

1. Check that all the switches are outputting the correct data by running a program that checks their states and then correspond that with the real-world state.
2. Check that the LCD is printing the correct symbols and that all the strings fit in the limited screen space by scrolling through and activating all the possible screen outputs.
3. Check that the interrupt is being activated by the press of the rotary button by using the Serial Monitor to troubleshoot when the Interrupt was activated.
4. Check that the general screen switches timely enough between the two available screens by timing the time between switching. It should be 20 seconds.
5. Ensure that the rotary button is functioning properly by checking that the area screens get scrolled through without any bugs.
6. Use common sense to determine if the LCD screen outputs information similar to real-world conditions. E.g. when it is a clear day it should state that it isn't raining and that it is daytime.
7. Make sure that the Fire mechanism of the coding works by simulating a fire (via a lighter or matches in a controlled and safe manner) in front of the fire sensor. The LCD should state that there is a fire and all the relays should turn on regardless if any sprayers are connected to it.
8. Ensure that the general area is working by switching off all the areas and then checking if there is being irrigated when the conditions are met (simulate the correct conditions).
9. Check if each area is working individually by plugging in all the needed sensors and then disabling the other areas including the general area. Simulate the correct and see if the corresponding relay turns on. Repeat this for all the areas.

10. Ensure that all the formulas and calculations are correct by manually retrieving the data via the Serial Monitor and doing them with a calculator and then checking if the processed data is correct.
11.
12. Check that each sensor is working individually by troubleshooting the screen:
    a. Light Sensor: When you manipulate it by covering it up to simulate night check that the screen reflects this data by stating it is “night” and vice versa.
    b. Moisture Sensor: Manipulate this data by putting it in different types of soil and then check if the data on the screen changes accordingly.
    c. Rain Sensor: Manipulate the rain sensor by pouring water on it and check that screen displays that is indeed raining. When you dry the water check that the screen display outputs that it isn’t raining anymore.
    d. Temperature Sensor: Retrieve the output from the temperature sensor via the screen and compare this to real world conditions using a weather app and check if it is closely reflected.

Evaluation of System:

Electronics:

1. The Power supply is outputting constant 5V according to the multimeter as it is meant to be.
2. According to the continuity tests conducted with all the wires there are no overlapping circuits or incorrectly soldered wires.
3. The relays switch on and off correctly as evident by the connected LEDs turning off and on with it.
4. After checking all the modules, it was concluded tat everything is running and active. This was deduced from the fact that all the onboard LEDs were shining when the modules were connected to the relevant ports.
5. When the LCD was connected the backlight lit showcasing that it is functioning properly.
6. The switch’s on and off state corresponds correctly with the on and off state of the complete system.

3D-Printed Housings:

1. Housings covered most water sensitive electronics all though it was not completely water proof. This should be revisited in the future.
2. All the switches and ports are accessible through the housings of the system without having to physically remove the housing.
3. All surfaces that need to be able to be seen by the user are visible through or on top of the housing.

Programming:

1. A program returning the states of all the switches on the system was run and this corresponded perfectly with the real-world states of the switches.
2. The LCD was checked for string truncation or garbled information because of unknown symbols and everything seemed to be displaying correctly.

3. The Interrupt was modified to print a message whenever it was activated by the rotary button. After numerous presses it proved that rotary button was working and in fact activating the interrupt.
4. The time it takes for the LCD to switch between the general screens were more or less 20 seconds each time. This is correct and shows that the programming is functioning properly.
5. The area screens were scrolled through using the rotary encoder and it worked fine with minor glitches here and there.
6. A fire was simulated (In a safe and regulated manner) and the system responded correctly by displaying a warning message and turning on all the relays.
7. The general area functioned correctly after the necessary steps were taking to test it individually separate from the areas.
8. Each Area was tested individually and functioned properly by giving the correct data and activating its corresponding relay when conditions were opportune.
9. The code was temporarily adjusted to give the raw values on the Serial Monitor before they were processed by the programming formulas. The raw values were than manually processed and all the answers were similar to those outputted by the system. This proving the formulas are functioning aptly.
10. The screen was used to troubleshoot the following sensors:
    a. The screen displayed night when the light sensor was covered and vice versa.
    b. Different soil types, like those documented in the parameter testing part of this Expo, were used to test the moisture sensor and the screen displayed logical results.
    c. After water was poured on the rain sensor the screen displayed that there was presence of rain and after the water was dried the screen displayed that it wasn't raining.
    d. The temperature was recorded on several occasions and compared with a weather app and the results were mostly similar. Yet the temperature sensor being a more specific and less predictive it was allowed the benefit of the doubt if values seemed a bit far apart.

Changes made:

- The number of general sensors were changed. It seemed over excessive to have two temperature sensors (as illustrated in prototype 3) as one logically placed sensor would serve the purpose of two. Therefore, the final sensor tally for the general area is: 1 temperature sensor, 1 rain sensor and 2 fire sensors. Code was adjusted accordingly
- The number of sensors per Area were changed. A single light sensor in an Area would be more than enough to provide accurate data (unlike the two per Area illustrated in prototype 3). Therefore, the final sensor tally per area is 2 moisture sensors and 1 light sensor. Code was adjusted accordingly.
- To ensure that power is not wasted in the operation of this system there was a switched installed to turn on and off the 240V power from the grid to the onboard power supply.
- The LCD button interface (As seen in prototype 3) was changed to a single Rotary pin to make it more ergonomic for the user. Therefore, this makes the system easier to use and also uses less pins on the Arduino Mega.
- A beefier power supply was used rather than the one previously mentioned. The update price is included in the costing table.

Costing:

Calculating the price of Florivate is a bit more complex than one single table. First, the prices of all the separate modules have to be calculated along with their housings. Then a table has to made with the cost of the base model and then a final table which will show the cost of a maximum capacity Florivate. Note that all the prices were sourced from the same site, namely (Banggood, n.d.). Except for the 3D-printer filament which was sourced from (www.uglymonkey.co.za, n.d.) at R360 for 1KG.

Cost of Moisture Sensor Module:

ITEM	Price
Base Moisture Sensor	R21.31
Wires (x3)	R3.45 (R34.50 for 10)
Filament (20g)	R7.20 (R360 for 1KG)
Total:	R31.96

Cost of Light Sensor:

ITEM	Price
Light Dependent Resistor	R1.93 (R19.36 for 10)
Wires (x3)	R3.45 (R34.50 for 10)
Filament (2g)	R0.72 (R360 for 1KG)
Prototyping Board (1/4)	R1.81 (R36.21 for 5)
10K Ohm Resistor	R0.09 (R51.39 for 560)
Total:	R8.00

Cost of Fire Sensor:

ITEM	Price
Fire Sensor	R25.34
Wires (x3)	R3.45 (R34.50 for 10)
Filament (8g)	R2.88 (R360 for 1KG)
Total:	R31.67

Cost of Rain Sensor:

ITEM	Price
Rain Sensor	R29.94
Wires (x3)	R3.45 (R34.50 for 10)
Filament (13g)	R4.68 (R360 for 1KG)
Screws (x4)	R3.47 (R139.13 for 160)
Total:	R41.54

Cost of Temperature Sensor:

ITEM	Price
DS18B20 Temperature Sensor	R61.97
Filament (12g)	R4.32(R360 for 1KG)

Total:	R66.29

Now that all the module prices have been calculated the price of the base model has to be calculated. This is without any sensor prices included.

Base Model:

ITEM	Price
LCD Screen	R47.35
Rotary Encoder (x1)	R14.48 (R72.42 for 5)
Switches(x20)	R7.32 (R36.63 for 100)
Relays(x4)	R58.60 (R14.90 for 1)
3 Port Screw-In Pins(x20)	R39.60 (R1.98 for 1)
5V Power Supply	R194.84
Arduino Mega	R134.81
Filament(330g)	R118.8(R360 for 1KG)
Panel Wire (Estimate)	R30
10K Ohm Resistors(x20)	R1.80 (R51.39 for 560)
Power Supply Switch	R16.43
Prototyping Board (x7)	R50.69 (R36.21 for 5)
Total:	R714.72

Now that the base price is calculated the total price can be calculated if a person was to buy to full capacity of sensors.

ITEM	Price
Base Model	R714.72
Moisture Sensors(x8)	R255.68 (R31.96 for 1)
Light Sensors(x4)	R32.00 (R8.00 for 1)
Temperature Sensor	R66.29
Fire Sensor	R31.67
Rain Sensor	R41.54
Total:	R1141.9

Therefore, the maximum price with maximum capacity of sensor would be R1141.90 which is still a 1/3 of the price of competitors (As discussed in background study). Which makes it very affordable in that sense. Especially considering its functionality and considering the saving of water expenditure and therefore water expenses.

Basic Description:

The final product is a complete automated irrigation system. It has a 3D-printed housing for the main hub and also the separate sensors to make it more appealing for the user and also for more functional reasons like protection against the elements. It is fully customizable in terms of sensor quantity, yet it retains user-friendliness by making the addition and removal sensor easy with just the flick of the switch and loosening of ports. The system also has an LCD display that shows all the needed information about the immediate surrounding which can be used for troubleshooting or for

general information. It turns on the sprayers using relays which run the operating voltage of the sprayer's electromagnet. It is also available at a lower cost than similar products.

Conclusion:

I conclude that I have made a system that fulfils all the goals expected of it at the beginning of this project. It can comfortably take over all irrigation processes while ensuring that the least amount of water is expended. It also thinks of user-usability by having adjustable sensor quantities and easy to setup areas and sections while also boasting well designed 3D enclosures that protect it from the elements and generating appeal. As far as pricing the base model of this irrigation system is considerably less than other system that offer similar services as described in my background study.

Possible Improvements:

- Instead of using wires to connect the sensors to the main hub, Wi-Fi modules can be used to transmit the signal rather than underground wires that are effort to install and can also be easily damaged.
- A more modular approach can be taken. Instead of a base model every area can be purchased separately in a containerised fashion and so it can be built up from a smaller size to a much larger size and capacity.
- More user input can be added. Like the threshold of variables can be allowed to be slightly modified according to the season but not to extremes that would cause the over-expenditure of water or that the garden ends up dying.

Things I learned:

While creating this expo I improved existing skills and gained new ones. In terms of programming I better grasped Object-Oriented-Programming in C++ (Language used for Arduino) and can now implement it confidently in other projects. I learned about the state of our country concerning the water crisis and other countries in need of help. Furthermore, I got to work with things I've never previously encountered like acrylic paint (which is obvious in the way it looks) and bare bones relay which I figured out with a lot of trial and error. In conclusion this Expo has taught me to remain steadfast and embrace all challenges not with fear but with excitement.

Acknowledgements:

Anne-Marie Theron: For assistance with documenting the Expo Project and the Admin regarding signing up for the Eskom Expo.

Johan Coetsee: For replacing the head of my 3D-printer after it got clogged up and also the funding of most of the components.

Bibliography:

/www.news24.com/, 2017. *SouthAfrica/News/whats-causing-cape-towns-water-crisis-20170517.* [Online]
Available at: https://www.news24.com/SouthAfrica/News/whats-causing-cape-towns-water-crisis-20170517
[Accessed 30 April 2018].

Banggood, n.d. [Online]
Available at: https://www.banggood.com/
[Accessed 12 August 2018].

businesstech.co.za, 2016. */news/general/104441/who-is-using-all-the-water-in-south-africa/.* [Online]
Available at: https://businesstech.co.za/news/general/104441/who-is-using-all-the-water-in-south-africa/
[Accessed 30 April 2018].

https://albertonrecord.co.za/, 2017. */rainfall-monitor-compares-cape-town-rainfall-40-years/.* [Online]
Available at: https://albertonrecord.co.za/148712/rainfall-monitor-compares-cape-town-rainfall-40-years/
[Accessed 30 April 2018].

https://qz.com/, 2017. */cape-town-drought-and-water-shortage-in-south-africa-is-now-a-disaste/.* [Online]
Available at: https://qz.com/1110143/cape-town-drought-and-water-shortage-in-south-africa-is-now-a-disaste/
[Accessed 29 April 2018].

www.arduino.cc, 2017. */en/Guide/ArduinoMega2560.* [Online]
Available at: https://www.arduino.cc/en/Guide/ArduinoMega2560
[Accessed 6 May 2018].

www.electronicshub.org, 2015. */auto-irrigation-system-using-soil-moisture-sensor-and-pic-microcontroller/.* [Online]
Available at: http://www.electronicshub.org/auto-irrigation-system-using-soil-moisture-sensor-and-pic-microcontroller/
[Accessed 3 May 2018].

www.ft.com, 2016. */content/8de6e8d0-6e5e-11db-b5c4-0000779e2340.* [Online]
Available at: https://www.ft.com/content/8de6e8d0-6e5e-11db-b5c4-0000779e2340
[Accessed 30 April 2018].

www.geeetech.com, 2017. */wiki/index.php/Serial_I2C_1602_16%C3%972_Character_LCD_Module.* [Online]
Available at:
https://www.geeetech.com/wiki/index.php/Serial_I2C_1602_16%C3%972_Character_LCD_Module
[Accessed 20 May 2018].

www.hobbytronics.co.za, 2018. *p/547/power-supply-12v5v-2a.* [Online]
Available at: https://www.hobbytronics.co.za/p/547/power-supply-12v5v-2a
[Accessed 6 May 2018].

www.rainmachine.com, 2018. */products/rainmachine-touch-hd-12.* [Online]
Available at: http://www.rainmachine.com/products/rainmachine-touch-hd-12.html
[Accessed 6 May 2018].

www.robotshop.com, 2018. */en/arduino-mega-2560-microcontroller-rev3.html.* [Online]
Available at: https://www.robotshop.com/en/arduino-mega-2560-microcontroller-rev3.html
[Accessed 6 May 2018].

www.tweaking4all.com, 2014. */hardware/arduino/arduino-ds18b20-temperature-sensor/.* [Online]
Available at: https://www.tweaking4all.com
[Accessed 20 May 2018].

www.uglymonkey.co.za, n.d. *15-pla.* [Online]
Available at: https://www.uglymonkey.co.za/15-pla
[Accessed 12 August 2018].