Climate Edge

EE3 DTPRJ: Group 9
Final Report

Authors:

Davinia Kulendran	(00933918)
Vanita K	(00933621)
Keyan Sadeghi Namaghi	(00861078)
Afraz Arif Khan	(00963429)
Dhruv Jain	(00945390)
Mikheil Oganesyan	(00853872)
Deloris Owusu	(00942777)

Contents

1 Introduction

Climate Edge is a startup focused on gathering climate data to assist coffee farmers in developing countries in fighting the reduced crop growth brought about by climate change.[1] Currently, they are primarily working in Nicaragua where climate change has brought about severe droughts, thus impacting the quantity and quality of coffee crops produced.[2] Climate Edge installs weather stations in coffee farms that send hourly air temperature, soil temperature and humidity data to servers monitored by technicians. This data is read from the sensors and transmitted via GPRS using a PCB installed in the weather station. These technicians then analyse the data and send feedback and recommendations to farmers via SMS which enables the farmers to improve their crop yield.

1.1 Design Problem

To further expand the scope of Climate Edge's recommendation, our team was tasked with the following 2 projects:

1. Design a precipitation sensor that does not use any moving parts.
2. Create an android application for a smartphone that will replace the PCBs installed on the weather station

1.2 Design Specifications

The primary design specifications to meet the basic client requirements are building a rain sensor without moving parts that can provide useful and accurate rain data and building a phone application that can send the sensor data to Climate Edge's servers via GPRS in a user- readable format. Commonly used rain sensors today usually have moving parts that cause wear and tear and increase the cost of the product. Hence, these sensors are expensive and have reduced operating life, thus warranting the need for a sensor without moving parts. As for the phone application, the PCBs(printed circuit board) are harder to produce and more expensive than a cheap smart phone, therefore requiring an application that could mimic the operation of the PCB. This would significantly reduce Climate Edge's production cost. Further design specifications are described in Figure 1.

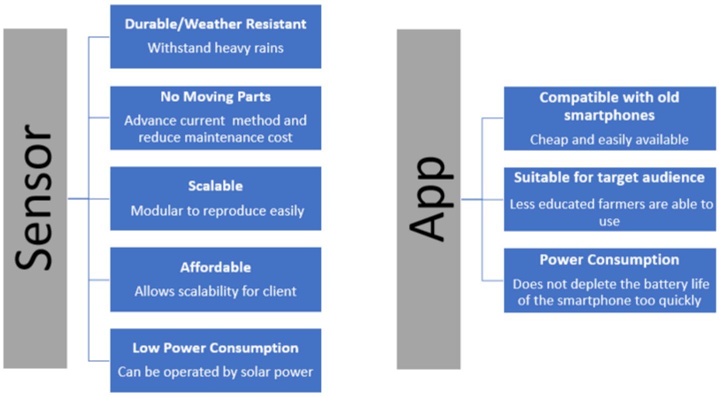


Figure 1: Figure showing design specifications

For both the precipitation sensor and phone application, low power consumption is pertinent. As the weather station is to be powered via solar cells, power consumption needs to be minimal to avoid overloading the supply. Although exact specifications of the solar cells are unknown as they have not been finalised for purchase, the size of the solar panel is estimated to be approximately $600cm^2$. Commercially available solar panels of similar size provide approximately 70Wh per day.[3] Therefore, the sensor and the phone application need to consume less than 70Wh per day.

Other design considerations for the sensor are durability, affordability and scalability. As the sensor will be deployed in areas with appreciable rainfall, a robust sensor is necessary that does not get easily damaged by the rain and is hermetically sealed to prevent the rain from affecting the electrical components. The sensor will also need to be lightweight and easily duplicable for rapid deployment in different farm sites. Furthermore, as the target market includes farmers in developing countries, the product will need to be cost effective to produce on a mass scale. The product also needs to affordable as Climate Edge, being a start-up has limited resources for rolling out a new feature. The budget set by Climate Edge for the project is £45.

The phone application also has further design requirements such as compatibility with old smart phones and suitability for the users. The application must work with old firmwares of Android (such as 2.3 Gingerbread). These old smart phones are readily available at a low cost in the UK and since these are the smart phones that will be deployed at the weather station, the application cannot have features or layouts that did not exist in older versions of Android. In addition, it is important to make the final design of the application as simplistic as possible such that the farmers may also be able to use it.

1.3 Market Research

Firstly, research was performed to identify the rain parameters that need to be measured by the sensor. The rainfall requirements for coffee crops are dependent on the soil properties, humidity and cloud cover. Generally, the ideal annual rain depth is 1200-1800mm.[4] Moreover, the time when rain occurs is crucial. Coffee crops require a short dry spell of 2-4 months for flowering to occur.[4] Excessive rainfall can lead to scattered harvest, low yields and the growth of unwanted fungi on crops.[5] Insufficient rainfall also leads to reduced crop yield. The onset of climate change in the past decade has weighed heavily on the growth of coffee crops due to their sensitive nature. It is estimated that Nicaragua, one of Climate Edge's operating location, could lose most of its coffee crop growing areas by 2050 if no action is taken.[6] Another parameter that is important for these crops is size of the rain drop. Drop sizes have been shown to directly correlate with extent of soil erosion with larger drop sizes having a more substantial impact on soil erodability.[7] Increased soil erosion can lead to reduced nutrient absorption of the crops thus decreasing crop yield. Hence, based on the research, rain depth (in mm), rain drop size and rain intensity were deemed as important parameters that should be measured by the sensor.

Type	Pros	Cons	Commercial Availability	Price
Rain Gauge	Low cost Simplest design	Hard to use	Yes	£12.95
Tipping Bucket	Easy to use Wider range of sensing	Moving parts Expensive Inaccurate	Yes	£60.17
Optic	Wider range of sensing Simpler design	Requires electrical supply More expensive than some others	Yes	£46
Acoustic	Low cost	Increased noise interference Requires electrical supply Narrower range of sensing	No	N/A
Piezoelectric	Low Cost Does not require electric supply	Complexity involved in design	No	N/A

Figure 2: Figure showing different sensor types available

Consequently, research was done on the various methods used currently to measure precipitation. The most common method used is the tipping bucket, which involves a bucket automatically tipping once it has reached maximum capacity and measuring the number of times this occurs. However, this technique is inaccurate especially in the case where there is insufficient rainfall to tip the bucket. The moving parts within the bucket also add cost and introduce wear and tear that reduce longevity of the product.[8] A cheaper alternative, known as the rain gauge, can also be used to measure precipitation. However, this requires users to manually empty the gauge making it more labour intensive than the tipping bucket.[8] Apart from mechanical solutions, research has been done in the past few years into alternative methods such as acoustic, optical and piezoelectric sensors. Acoustic sensors use microphones to detect sound waves. Each rain drop size has a unique sound pattern and the sound signal detected is analysed to retrieve rain information. Despite being affordable, the acoustic sensor requires a supply voltage and current to operate. Moreover, it is easily affected by external sound interferences and has a narrower sensing range compared to the optical and piezoelectric sensors.[9] The optical sensor uses the ability of raindrops to diffract and reduce the intensity of light. Optical sensors generally have a greater sensitivity, but like the acoustic sensor, it requires power supply.[10] The optical sensors are also more expensive due to the need of multiple optical sensors or high precision optical equipment to maintain the accuracy of the sensor. The piezoelectric sensor, while being cheap and not requiring a power supply is more difficult to obtain rain information from. This is mainly due to the signal representing the impact force of the rain drop and cannot be directly correlated to a rain drop size without extensive testing.[11] The summary of the different products available in the market is available in Figure 2.

Finally, research was carried out on Climate Edge's competitors, namely Smart Vineyard and Arable. Both companies use solar energy to power their weather stations. However, unlike Climate Edge, they already have built in precipitation sensors in their weather stations. Arable uses acoustic sensors[12] while Smart Vineyard uses an electromechanical precipitation sensor, similar to the tipping bucket.[13] Moreover, unlike Climate Edge which caters to farmers in developing countries, both companies cater to farms in developed countries. Therefore, although there is an urgent need for Climate Edge to incorporate a precipitation sensor into their weather station, it is essential that the sensor is affordable for farmers in developing countries. Climate Edge's competitors also utilise different methodologies for displaying the sensor data. Smart Vineyard sends the data to their domain hosted website where real time measurements of the sensor are available to the farmer[13] while Arable makes use of a cloud based application to show the data directly on the smart phone of the farmer.[12] While both of these methods have their merits they are not entirely suitable for Climate Edge's clientele as the farmers in Central America and Africa may not have internet access or access to a smart phone. Therefore, the application designed will be hosted on a smart phone purchased by Climate Edge and installed in a weather station at the farm. This is perhaps the only solution which does not require access to modern technology on the farmer's end. Therefore, based on the research into Climate Edge's competitors, it can be seen that due to Climate Edge's client base, extra considerations have to be made to ensure the product is affordable, low power and easily maintainable. This reduces the overall long-term cost of the weather station for the farmer as it will require less power and less skilled labour to maintain.

2 Design Selection

2.1 Precipitation Sensor

Having studied the various different sensors used in commercial applications and research studies, the 3 sensor types that met the basic design specifications were the acoustic, optical and piezoelectric sensor. These 3 types of sensors were then weighed against each other with primary concerns being power consumption, affordability, reliability and innovativeness. The 3 sensors were then ranked in terms of the primary concerns based on the market research. The ranking matrix can be found in Appendix G. The piezoelectric sensor had the overall highest ranking due to its minimal power consumption and cost. The piezoelectric sensor will also be the most innovative implementation as it has yet to be used in any commercial products. However, additional testing is required to ensure the sensor is well calibrated and outputs reliable data.

The sensor also requires a casing to house the electronics. Figure 3 shows the designs considered.

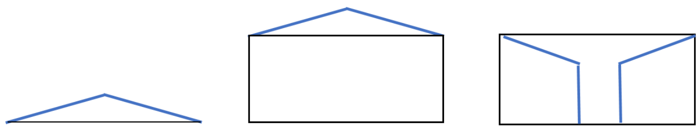


Figure 3: Figure showing design considerations for casing

The design requires the sensors to be beneath an inclined surface to aid water roll off. The first design made of only the hat is not large enough to house the electronics. The 2nd and 3rd design have sufficient space to house the electronics but the 3rd design causes water to roll off into the sensor. This could limit the locations at which the design can be used. For example, the sensor cannot be affixed to the ground as water will accumulate within the sensor. The 2nd design is also simplistic with minimal design components making it easier to print or mould. Therefore, the 2nd design was selected for the final prototype.

Finally, the data processing method needs to be determined. There are 2 possible ways to process the data. One involves an analogue solution that retains the rain information over small periods of time before being sampled. The other is a digital solution that continuously samples the data. The advantage of the analogue solution was the reduced sampling frequency which could potentially lead to lower power consumption. The advantage of the digital solution was the ease of implementation and greater accuracy. Finally, the digital solution was chosen due to its reliability and low complexity. Moreover, to enable low power consumption, low power microcontrollers such as the PIC12F1840[14] can be used . However, during the building process, the microchip was difficult to program without prior knowledge in microchip programming. Resources available were also usually not chip specific. Therefore, the ATMEGA328P was used instead as it enabled low power consumption while allowing for easier programming using the Arduino IDE.

2.2 Phone Application

For the phone application, the main design consideration was the method of communication between the phone and the microcontroller. There were four viable methods which were explored: Bluetooth, BLE (Bluetooth Low Energy), USB OTG and Frequency Modulation via an auxiliary audio cable.

To evaluate each potential method, the criteria on which they would be compared was clearly defined, with the main criteria being cost, power efficiency and compatibility.

Standard Bluetooth (2.1) is the most common method for interfacing between Android and microprocessors out of the shortlisted options. This is due to the hardware being incredibly cheap, available and simple to interface with. As it is a wireless solution it is considered to be robust and easy to waterproof which are important qualities when operating in an agricultural environment, especially if the Android device isn't permanently housed within the weather station. Support for this type of Bluetooth is generally available in most Android phones. The drawbacks focus mainly around the power consumption as it has the highest consumption of any of the options. It is also constantly transmitting data even when idle which increases overall current consumption.

BLE is the Bluetooth Special Interest Group's solution to these power issues. It has most of the positive qualities of standard Bluetooth with far lower power consumption as it only sends data when directly instructed. The main issue is that BLE support was only added in Android 4.3[15] thus lots of older devices[16] found in developing areas such as the farms Climate Edge are working with will not work with BLE. It should also be noted that even though power consumption is far lower, it is still non zero and will typically consume between 0.4 and 1.5mA when sleeping and 8.5mA when active.[17]

A wired solution is the optimal solution for reducing power consumption. Using USB OTG would be the logical solution as it only uses 8mA when communicating and can completely disconnect when not. There is no additional hardware other than the physical cable which makes it incredibly cheap to implement. Water and dust proofing are more problematic if the device is not housed within the station. The cost to ensure robustness could add up and become comparable to the cost of the bluetooth modules. USB OTG support has existed in Android since Android 4.1 but that doesn't mean all phones since have supported it with even Google's own Nexus 4 not supporting it despite running Android 5.1.1.[18]

An alternative physical connection is to use the headphone jack and an auxiliary audio cable. The principle is to transmit a frequency modulated sound wave to the phone by using the audio jack's microphone connection. While nice in theory, it adds unnecessary complexity and has the problem that there is not one universal standard for which auxiliary connector is ground and which is the microphone.[19]

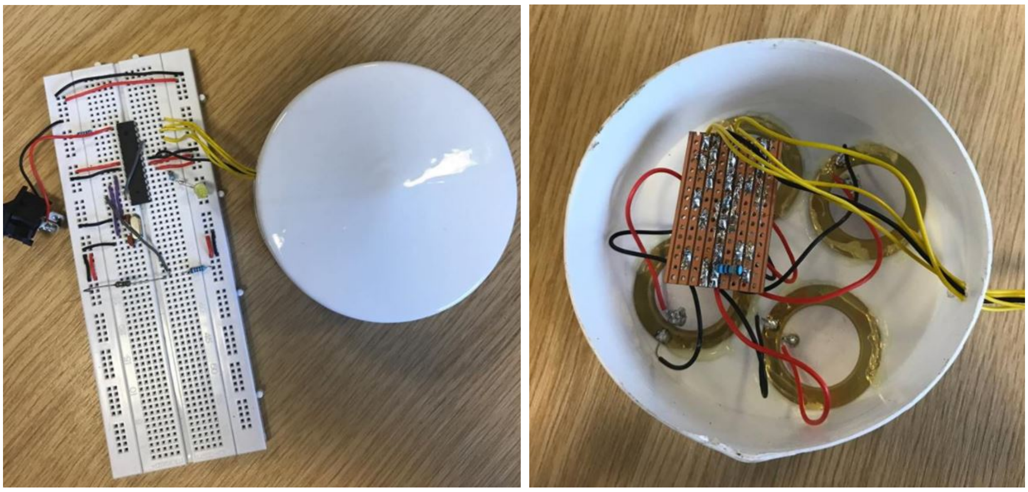
Ultimately, a HC-05 bluetooth 2.1 chip was used because the advantages of a wireless solution gave the flexibility of having the phone housed within the station or being the farmer's own personal phone. Without more data being gathered on the type of phones which would be available, BLE couldn't be justified due to compatibility issues.

3 Precipitation Sensor Design Outline

The design of the precipitation sensor involves 2 major processes. The first design process entails building the exterior casing of the sensor. As described in the design requirements, this exterior needs to be waterproof and rigid to withstand harsh weather conditions. The second design process entails the building of the software program within a microcontroller that translates the sensor data into useful rain information. The program needs to be easily modifiable to enable testing results to be incorporated effortlessly into the algorithm. Moreover, the sensor also needs to be well calibrated to produce reliable data and consume minimal power to meet the design requirements.

3.1 Sensor Circuitry

The final sensor prototype can be seen in Figure 4.



(a) Figure showing sensor prototype module (b) Figure showing inside of sensor module

Figure 4: Figure showing final sensor prototype

3.1.1 Piezoelectric Elements

The precipitation sensor holds 4 piezoelectric elements beneath the sensor surface. The sensors are of 35mm diameter and are arranged in a circular way inside the casing. The piezoelectric elements function by transforming dynamic mechanical strain into electric charge. When mechanical stress is applied on the the element, re-orientation of dipole moments or reconfiguration of the dipole-inducing surrounding occurs, thus causing a change in polarisation.[20] This change in polarisation causes a change in electric field between the faces of the material and the overall process can be described by the following equation[20]

$$S = sT + \delta E$$

where S = mechanical strain, δ = piezoelectric effect coefficient, T = stress, s = compliance and E = electric field.

The size of the each piezoelectric element is chosen based on the ease of availability of the element. The piezoelectric element of 35cm is the most commonly found, with other sizes requiring custom production. Larger sized elements also increase the statistical variation of more than one drop hitting the element at the same time.[11] This produces an erroneous reading of the combined energy of both drops instead of just one drop. Using more than 1 element allows for increasing the sensor area which enables the acquiring of more reliable data. However, using too many elements increases the computational complexity and reduces the range of microcontrollers that can be used for the program. Moreover, too many elements would increase the size of the sensor, thus making it harder to transport to Climate Edge's locations. Therefore, 4 elements was found to be a good compromise.

3.1.2 Microcontroller

The microcontroller provides the 4 ADC channels for analog to digital conversion, computational power to execute the software program and an analog comparator that is used for interrupts within the program. ATMEGA328P is chosen for this project as it can be easily programmed in C using the Arduino IDE. This also simplifies testing as real-time results can be seen using Arduino's serial monitor. In the final prototype, the ATMEGA328P is built on a standalone breadboard. The main purpose of this is to reduce power consumption by bypassing high power components such as the linear regulator and USB bridge. The schematic can be seen below and will be explained in greater detail in Section 1.3.

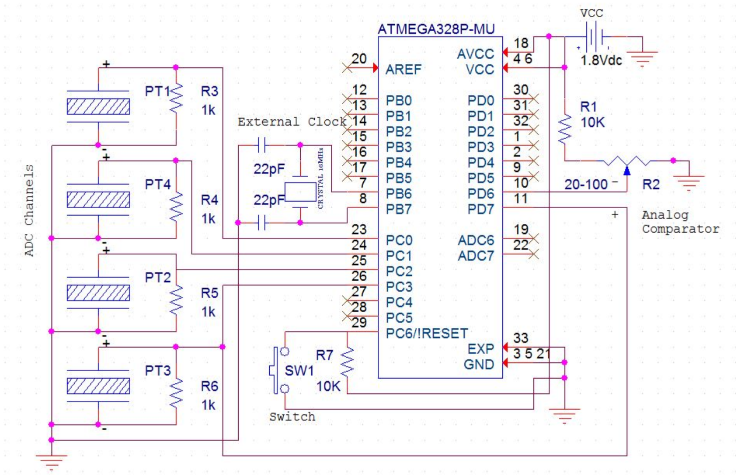


Figure 5: Figure showing schematic for microcontroller

3.2 Exterior Casing Design

The main requirement for the casing is to protect the electronics from the weather conditions. However, the casing itself can introduce reading errors if not properly designed. A casing that does not have a smooth top will cause water to spread and collect on the surface. The puddles of water formed on the surface will dampen the force of any new drops thus introducing inaccuracies in readings. However, surfaces that are ultra hydrophobic such that they allow water to roll off easily cause collisions of water droplets with the surface to be more elastic, therefore causing the water droplet to rebound on the surface with a greater force[21] This induces additional peaks in the signal and causes one drop to be registered as multiple drops.

In addition, the casing needs to be tilted at an angle to allow the water to roll off. A steeper angle allows the water to roll off more easily but amplifies the radius effect.[11] The radius effect is the difference in signal when a drop hits different parts of the piezoelectric element. When the drop hits closer to the centre, a greater electric field is produced as opposed to hitting the element further away from the centre. A compromise is reached when the angle of slope is approximately 10°.[22]

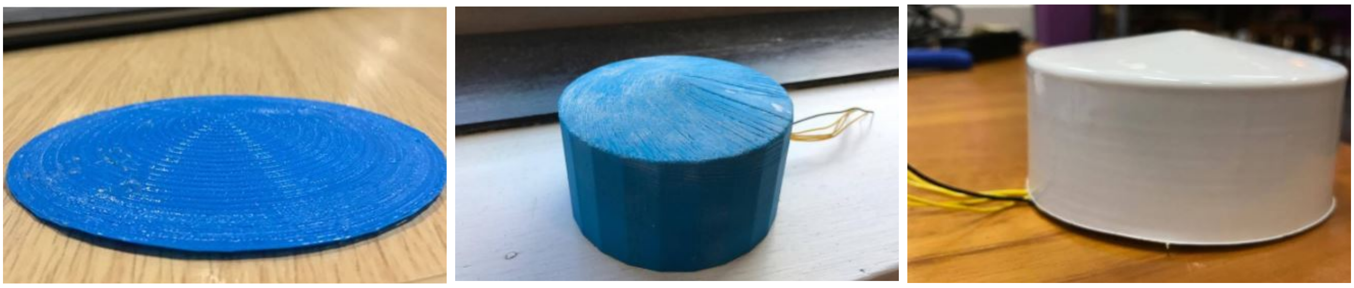


Figure 6: Figure showing evolution of prototype

Figure 6 shows the evolution of the casing from the 1st 3D printed prototype on the far left to the final vacuum injected prototype to the far right. All 3 prototypes have a 10° inclination on the top. Initially, PLA(Polylactic Acid) was used as the casing material and the prototype was 3D printed due to ease of availability and low cost. The material is also waterproof and met the basic requirement for the casing. However, as can be seen from Figure 6, the 3D printed casings did not have a smooth finish on the top surface which as explained earlier, was undesirable. This was because the prototype had to be printed on its side to ensure that the inside of the casing would be hollow. The final prototype is vacuum molded and made of high impact polystyrene. This material not only allows for a smooth finish, but is also rigid and has high impact strength. Moreover, it is waterproof and resistant to chemical corrosion, making it a suitable casing material.[23]

The final considerations for the casing would be size and thickness. The diameter of the casing is such that it can enclose all 4 sensing elements. The thickness of the top surface is crucial for the sensitivity of the sensor. A surface that is too thick will reduce the impact force detected by the element, thus making it difficult to register smaller and lower impact drops. However, a surface too thin will be difficult and costly to reproduce and can be easily damaged. The current prototype has a thickness of approximately 0.5cm. At this given thickness, the smallest raindrop size registered by the sensor is 0.5mm, which is acceptable given that raindrop sizes range from 0.1mm to 5mm.[24]

The casing also requires a cylindrical bottom half to contain the electronics. The piezoelectric elements are glued beneath the surface of the sensor as can be seen in Figure 4b using hot melt adhesive, commonly known as hot glue. The hot melt adhesive is easily accessible, but not resistant to extreme temperature changes.[25] An alternative design would be to create slots for the piezoelectric elements during moulding. However, this increases the complexity of the design, making it harder to reproduce. Having acknowledged that temperature changes are minimal in tropical countries where Climate Edge operates, the final prototype was created by affixing the elements to the surface using hot melt adhesive.

3.3 Software Program

The software programmed into the ATMEGA328P primarily performs 3 functions. It processes the signal from the piezoelectric elements, reduces power consumption of the sensor and processes the data. The full program code can be viewed in Appendix A.

3.3.1 Signal Processing

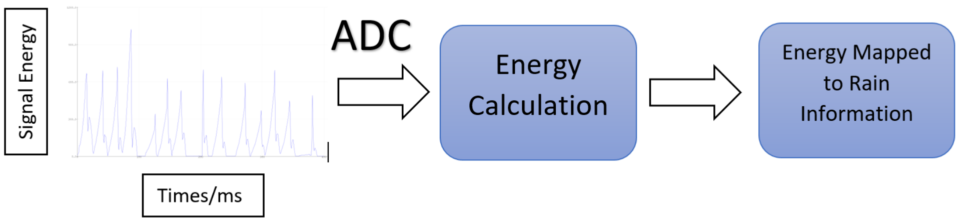


Figure 7: Figure showing how signal is processed

Figure 7 shows how the data is processed from the piezoelectric sensor. The first stage involves converting the analogue signal to digital bits for more convenient processing. The ADC sampling frequency can be set in the program, however, the micro-controller requires the ADC clock frequency to be between 50kHz and 200kHz to preserve 10 bit precision due to its internal architecture.[26] The ADC sampling frequency is then derived from the ADC clock frequency and is approximately 1/13th of the clock frequency as it takes 13 clock cycles to perform the conversion.[26] The ADC sampling frequency of the signal also needs to be at least twice that of the signal frequency to prevent aliasing. To obtain the signal frequency, 20 different piezoelectric signals were timed using the `millis()` function. The timings for the start, peak and end of signal were recorded. The full data can be viewed in Appendix E. Based on the data, the average period of the signal is 59.8ms and the average time for the signal to reach its maximum value is 17.9ms. Therefore, the signal needs to be sampled at least every 9ms to be able to retrieve the full signal pattern accurately. Consequently, the ADC sampling frequency has to be at least 120Hz. Given that all 4 elements are sampled successively, the overall ADC sampling frequency needs to be 4 times that of each individual sampling frequency. Hence, the ADC clock frequency needs to be a minimum of $120 \times 4 \times 13 \approx 6.3$kHz. As the minimum ADC clock frequency of the microcontroller of 50kHz meets the minimum sampling requirements, the program sets 50kHz to be the ADC clock frequency.

Once the analogue to digital conversion has been completed, the program then parses the data to function `calcEnergy`. The function computes the overall energy of the signal. The signal energy is dependent on the force of impact of the raindrop on the surface. This force is dependent on the size of the raindrop. Based on previous research on the mass and velocities of different rain drops,[27] the theoretical impact force can be calculated using the equation $F = \frac{mv}{\delta t}$ and assuming $\delta t = 1$. Since the impact time should be uniform despite rain drop size, the actual value of δt will not affect the overall relationship between impact force and rain drop size.

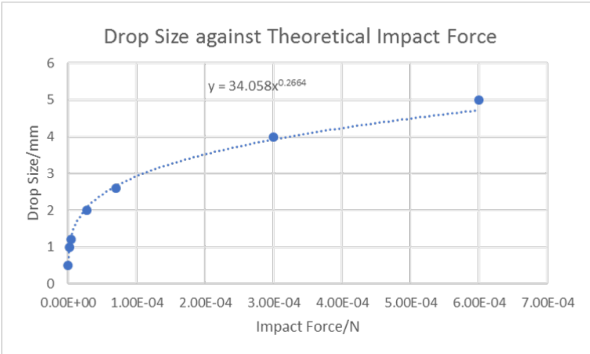


Figure 8: Figure showing theoretical force data

Figure 8 shows the results of this calculation. As seen from the graph, the impact force and rain drop size are related using the power equation, Drop Size $= a \times Force^b$, where parameters a and b require calibration. The impact force is proportional to the energy of the signal, therefore the energy of the signal needs to be determined. The energy of a discrete time signal can be computed as $E \triangleq \sum_n |x[n]|^2$. However, computing the square of every discrete value is computationally intensive and reduces program speed. Therefore, the function `calcEnergy` checks for the maximum value of the signal and uses $E_{peak} \triangleq \sum_n p[n]^2$, where p is the peak of each signal to approximate the energy of the signal. The function also only registers energy values that meet a specific threshold. This reduces the sensitivity of the sensor but helps remove low impact noise from wind or debris. The function also continuously averages the signal energy to significantly reduce the inaccuracies caused by erroneous reading with very low probability of occurrence.

After the program has computed the average energy for a fixed amount of time, the energy is then mapped to a specific rain drop size and rain depth using the function `signalMapper`. This function uses a pre-determined equation derived from test data to convert the signal energy to rain drop size . The calibration framework and test data will be covered in Section 6. The function also uses the rain drop size determined to obtain the rain depth. The rain depth can be derived from the rain drop size using the formula[11]

$$\text{Rain Depth} = \frac{\text{Drop Size}^2}{6 \times \text{Sensor radius}^2}$$

This rain information is then sent to the phone application using bluetooth communication.

3.3.2 Power Saving Functionality

Beyond the main framework, the program has in-built power saving functionalities to reduce power consumption of the sensor. When running the program using the Arduino Uno, the minimum voltage required is 3.8V, with the current consumption being 37mA. Removing the ATMEGA328P from the Arduino and building it on a breadboard reduces the current consumption to 17mA. The operating voltage and current consumption can be further reduced by making use of the microcontroller's sleep mode and reducing the operating clock frequency.

The microcontroller has 6 different sleep modes. The lowest power sleep mode is known as `SLEEP_MODE_PWR_DOWN` and the highest power sleep mode is know as `SLEEP_MODE_IDLE`. The microcontroller is put to sleep at the start of the program to conserve power while there is no rain using the function `sleepModeNoRain`. This sleep can only be interrupted if rain is being sensed by the sensor. This is achieved using the microcontroller's analog comparator. The positive input of the comparator is connected to one of the piezoelectric elements and the the negative input is connected to a very small voltage. This small voltage is set using a voltage divider connected to the supply voltage as can be seen in Figure 5. The potentiometer allows the user to set the desired threshold to prevent the microcontroller from waking up due to low impact disturbances such as wind or falling leaves. Since this sleep mode requires being interrupted by the output of the analog comparator, the microcontroller can only be set to sleep at the 2nd highest power sleep mode, known as `SLEEP_MODE_ADC`.

However, during long rain events, when solar power is limited and the microcontroller is constantly awake, little power is conserved. Therefore, sleep cycles during rain events are implemented using the function `sleepModeRain`. Figure 9 demonstrates the overall flow of the program. The awake and sleep cycles during a rain event prevents the microcontroller from being awake throughout a rain event. As both cycles are timed equally, the microcontroller will only be awake for half of the rain duration. Alas, the awake and sleep cycles have to be timed such that rain information is not lost during sleep. The current program varies the awake and sleep cycle timings from 1 minute to 5 minutes depending on the variation of the rainfall measured. Since the variation of rainfall during a rain event occurs frequently, rain is usually sampled between 1 minute to 15 minutes[28] for research purposes. Therefore, sampling the rain every 1 minute to 5 minutes should be sufficient in extracting the required rain information.

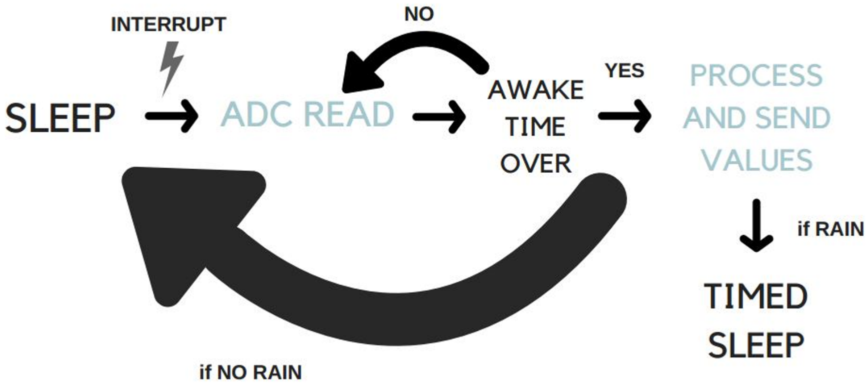


Figure 9: Figure showing program flow

The timing cycles are implemented using the microcontroller's internal watchog timer and the `millis()` function. The internal watchdog timer is referenced from an internal 128kHz clock and can be used to interrupt the lowest power sleep mode once the desired time has been reached.[26] One problem is that the watchdog timer has a maximum period of 8s.[26] This can be overcome by continuously looping the watchdog timer until the required timing is reached. The number of loops is set using the variable `sleepCyclesNo`. Due to timing overheads from waking up and configuring the watchdog timer, the number of loops cannot be computed theoretically but experimentally. The experimental data can be found in Appendix F. The number of loops required for a sleep cycle of 1 minute to 5 minutes is 5 to 32 respectively.

The awake cycle, on the other hand, is configured using the `millis()` function which uses the microcontroller's timer 0 and is inbuilt into the Arduino IDE. Timer 0 has an 8 bit counter register which is incremented every 0.004 milliseconds[26] and the output of `millis()` is the time since the program has started in milliseconds. The awake cycle time is tracked using the variables `timeNow` and `timeStart`. `timeNow` provides the current time of the program after each loop while `timeStart` provides the time since the cycle has started. The awake cycle timing is set using the variable `timeInterval`. This is the required time difference between the start time and the current time for the awake cycle to end. This time interval is set in milliseconds. However, since the counter for time 0 is dependent on the clock frequency, the output of the function `millis()` varies depending on the clock frequency. The function outputs time in microseconds for a clock frequency of 16MHz. For a clock frequency of 2MHz, the actual number of milliseconds will be the function's output multiplied by 8. Hence, the time interval is set between 7500 to 37500 to signify an interval of 1 minute and 5 minutes respectively.

Lastly, the clock frequency is reduced primarily to reduce the operating voltage of the microcontroller. Based on the ATMEGA328P datasheet, the microcontroller can operate at a minimum of 1.8V at 4MHz clock frequency. At 16MHz clock frequency, the minimum operating voltage is 3.8V.[26] The changing of clock frequency is done in setup, simply by setting the clock prescaler bits. For this program, bit 0 and 1 have been set for the clock to operate at 2MHz frequency. Changing the clock frequency also changes the ADC clock frequency. This can be modified easily by changing the ADC prescaler bits. This is also configured in setup such that the ADC clock frequency remains at 50kHz. As can be seen from table 1, the power consumption is greatly reduced by the power saving methods. The data is summarised in Figure 15 in Section 6 . Given the worst case

Table 1: Table showing current consumption of sensor during different modes

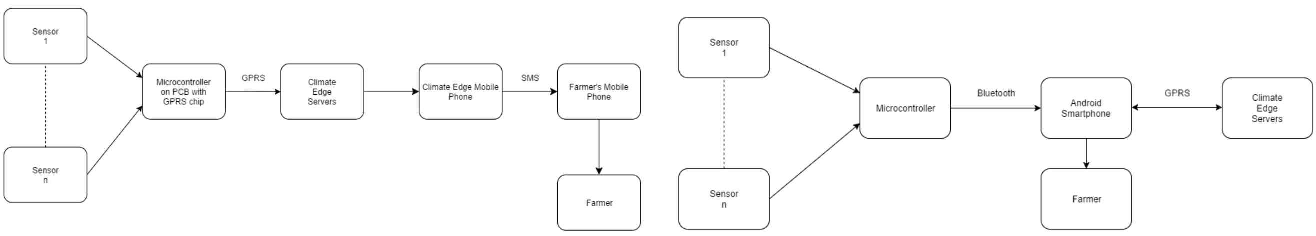
Input Voltage/V	Arduino Uno Current Consumption/mA				Breadboard Current Consumption/mA			
	Awake No Power Save	Interrupt Sleep	Timed Sleep	Awake Cycle	Awake No Power Save	Interrupt Sleep	Timed Sleep	Awake Cylcle
5.4	49	36	23	48	37	5	2	6
5	43	33	22	43	30	4	1	6
4.4	39	32	18	38	23	4	1	5
3.87	37	30	11	36	22	3	0.65	3
3.5	N/A	N/A	N/A	N/A	10	3	0.36	2
2.83	N/A	N/A	N/A	N/A	N/A	0.84	0.23	1
2.05	N/A	N/A	N/A	N/A	N/A	0.65	0.15	0.91
1.8	N/A	N/A	N/A	N/A	N/A	0.61	0.09	0.83

scenario where no rain occurs all year, without any power saving functionality, the sensor consumes 998 Wh per year. With the power saving functionalities, this is reduced significantly by 99% to 9.7 Wh per year.

3.3.3 Data Processing

At the end of the awake cycle, the average signal energy is mapped to rain information. This rain information is then stored in the `EEPROM` before entering sleep mode. This helps preserve data from the awake cycle temporarily in case of sensor shutdown due to insufficient input power. The data processing function, namely `dataReadWrite` also determines the awake and sleep cycle timings. When the program is initiated, the awake and sleep cycle timings are set at 5 minutes. However, if the change in rain between 2 cycles is significant, the cycle timings need to be changed to detect the rainfall variation. This is accomplished by comparing the current awake cycle's values to the previous cycle values that are stored in the `EEPROM`. If the change is greater the 20%, the awake and sleep cycle timings are reduced equally in proportion to the change in rainfall values with the minimum cycle timings being 1 minute. If the change is insignificant, then the function resets the timings to the maximum of 5 minutes. Changing the cycle timings equally also ensures that the power consumption will be the same despite the variation in rainfall but ensures this variation is detected by increasing the sampling frequency.

4 Phone Application Design Outline



(a) Figure showing current model of data communication (b) Figure showing proposed model of data communication

Figure 10: Figure showing data communication models

Figure 10a and Figure 10b show the current and proposed models of data communication respectively. In the proposed model, the sensor data is collected using a microprocessor and stored over a period of time where it is then sent to an Android phone via Bluetooth. The data is then stored and displayed on the phone, then sent to Climate Edge's server via GPRS where it will be processed and feedback is then sent back to the phone so the farmer can act upon it.

There are a number of reasons why using a smart phone is a cheaper and superior alternative to using a custom PCB which is the current implementation . As a PCB is a custom designed piece of hardware used to achieve a unique task, small scale production and revisions are a complex and costly endeavor. Relatively cheap smart phones currently saturate the developing markets and contain all the features of the PCB (built in GPRS) and more (such as large amounts of storage, bluetooth and touch screen displays). The application can be further utilised to transform the current weather station into a smart weather station where farmers can view real-time weather data and access expert analysis through Climate Edge's servers.

In order to design the application, Android Studio was used to create an application showcasing all the sensors connected to the device and readings of each sensor. These readings are shown on the smart phone as a graph of adjustable periods. This data is then uploaded to a Climate Edge server where it is processed and feedback is sent back to the phone. This feedback is then displayed with information on how to improve coffee yields. The full application code can be viewed in Appendix B.

4.1 Transmission of Data to Phone

The sensors are simply connected to the analogue and digital input pins on the arduino used for prototyping. For the precipitation sensor, data is sent via I2C communication and the function `Wire`. To differentiate each sensor's data a tag is concatenated to front of the value taken. The data currently being collected from sensors with their respective tags are: Rain depth (R), Drop Size (D), Soil Temperature (E), Air Temperature (A) and Humidity (H). The code for the microcontroller for sensor reading can be found in Appendix C.

This data is then transmitted using a HC-05 Bluetooth module which is powered from the microprocessor and is periodically triggered by a transmission from the application. This is done by utilising software serial and establishing a serial connection via the bluetooth module. The general process for this transmission is shown in the flow diagram in Figure 11.

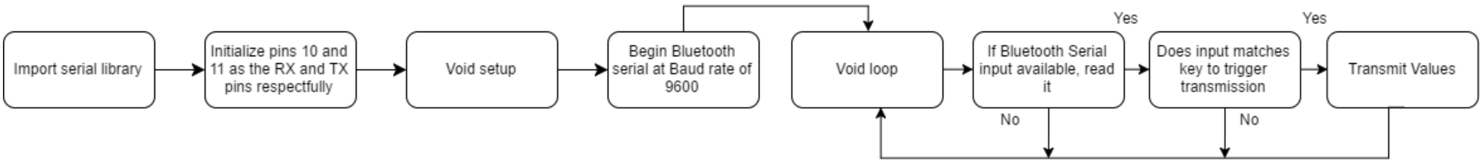


Figure 11: Figure showing how data is transferred from microcontroller to phone

4.2 Receiving of Data by Phone

The receiving of this data is more complex and is handled on the Android device. The application periodically establishes a connection and takes the new data in before ending the connection. Every time a connection is made, the application must check if the device's bluetooth hardware is available before it can connect. Once connected, a thread is created which buffers the incoming data into a file. This thread is then killed once the transmission has been completed and the connection is ended. This is shown in Figure 12.

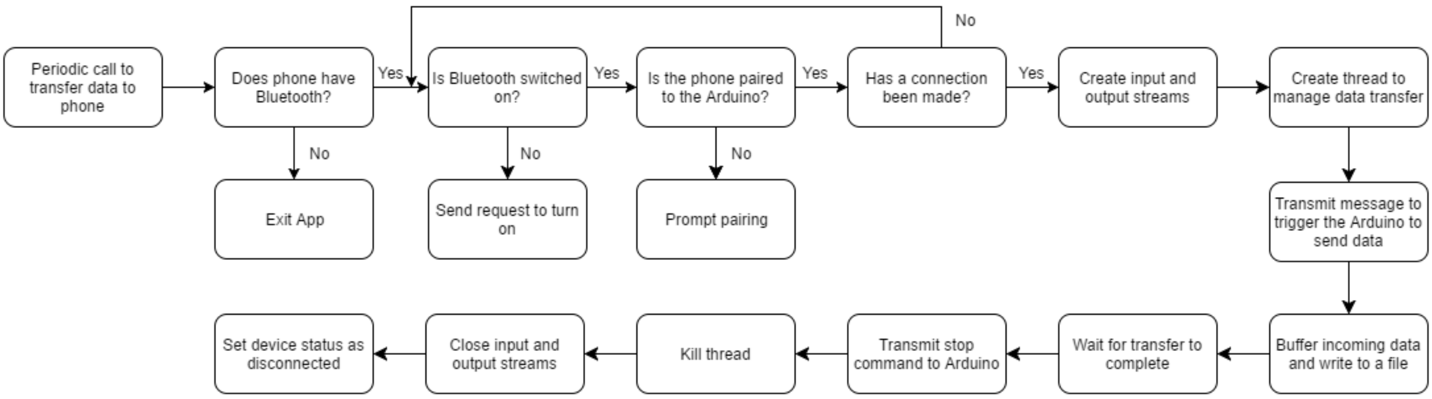


Figure 12: Figure showing how data is received by phone

Once the data has been logged, a graph displaying the latest sensor data and past data in a meaningful way is created using an external Android library called MPAndroidChart. The data is saved offline as a backup on the phone's memory and the application sends the data periodically to Climate Edge's servers.

4.3 Transmission of Data to Servers

The data is sent to servers using Android Volley.[29] Android Volley was selected for use over AsyncTask[30] for server communication, as Volley is specifically designed for the purpose of network accessing. A major advantage of Android Volley over AsyncTask is that you can do multiple requests simultaneously without the need for thread management. Additionally, the retry mechanism is a noteworthy feature of Volley which gives it an edge over AsyncTask.

Volley enables data to be sent in any format to a mySQL database by working in tandem with a PHP file. This is done so by using the `StringRequest()` function[29] in Volley. Using the Singleton Pattern Method, a `RequestQueue` is created that lasts for the lifetime of the application. The `RequestQueue` is used for handling network requests and for dealing with the cache. This is further explained by the 2 cases which Android Volley utilises.

4.3.1 Server Communication Configuration

The first case is when the phone connects to the database for the first time. Once a `StringRequest()` tries to connect to the HTTP Server, it will first go through the cache dispatcher to see if the data the program is trying to obtain already exists in the cache. Upon failure, it sends a "Cache Miss"[31] which will then prompt the dispatcher to send a `Request` to the network thread which correspondingly sends a `Request` to the database hosted on a local server through XAMPP. Given that the server is running, the database will send a `Response` to the network thread which sends the `Response` to the main thread of the application and to the dispatcher to log the `Response` in the cache memory.

The second case is when connection occurs after the first time. The `Request` dispatched by the cache dispatcher to the cache thread will now be successful and will result in a "Cache Hit".[32] This will prompt the `Response` to go back to the dispatcher and consequently back to the main program on the application.

After performing the `Request` method, the data needs to be either read from or stored to corresponding columns. This is referred to as `POST Request`. In Android Studio, these columns can be distinguished by building a hash map[33] that distinguishes each column in the database as a "key" and inputs data to each corresponding "key" or extracts data from each corresponding "key," depending on the application.

4.3.2 Data Transmission

All the sensor data recorded on the application is sent to a text file. The application reads each line of the stored text file and sends the data to the server. This data is then sent to the corresponding columns of the database table using the `INSERT` command in mySQL. This only occurs if the request sent by the application to the the server is successful. The frequency at which this process is repeated is dependent on the user requirements.

5 Costing

The costing of the sensor and application is estimated based on the different components used in the final design. The hidden cost is estimated at 10% of the price of the sensor and is used for tools, adhesives and etc.

Table 2: Table showing breakdown of product costs

Description	Quantity	Unit Price	Cost
Phone Application			
Arduino Uno[1]	1	£15	£15
Bluetooth Module[1]	1	£3.21	£3.21
Precipitation Sensor			
ATMega328P standalone[1,3]	1	≈£5	£5
Piezoelectric Sensors[1]	4	£1	£4
Casing Material	1	£0.28	£0.28
Hidden Cost			£2.75
Total			£30.24

[1]Prices from RS Components Limited
[3]Standalone price includes microcontroller, capacitors, resistors and crystal oscillators

The costing for the sensor and the application satisfies the client's budget of £45. The costs for the phone application can be further reduced if the Arduino Uno is replaced with a standalone microcontroller. Therefore, the final prototype has satisfied the design requirement of affordability.

6 Testing Procedure

6.1 Precipitation Sensor

To ensure optimal performance and for calibration purposes, a few different testing procedures were carried out:

6.1.1 Testing for Sensor Calibration

As the rainfall sensor uses drop size and number of peaks to ascertain total volume of rain, it was important to test the sensor with differing rain drop sizes. The sensor was calibrated using this information as the different rain drop sizes was mapped to different signal energies. This was done by drilling a hole of size ranging from 0.1 mm to 6mm into a plastic bottle cap. It was found that normal rain drops vary between 0.1mm and 5mm.[24] To accurately mimic rain, these drops would have to hit the surface of the sensor at terminal velocity. Terminal velocity varies with the size of the drop and the maximum terminal velocity that can be reached by a raindrop is known to be 12m/s.[34] The minimum height required to attain terminal velocity is 8m which was determined based on the equations of motions, $v = u + at$ and $s = ut + \frac{1}{2}at^2$. Thus, the height used to release the water from the platic bottle was chosen to be 12m or approximately the height from the 5$^{\text{th}}$ storey to the ground in the EEE building in Imperial College London. It was found that the sensor was unable to detect rain drops below 0.5mm due to the limitation of the piezoelectric sensor. However, rainfall with drop sizes below 0.5 mm is uncommon.[24]

6.1.2 Testing for Possible Reading Errors

The second testing procedure was similar but the rain drops were no longer made to hit the sensor at terminal velocity. The reason for this line of testing was to simulate rain drops that do not have a direct path to the sensor or hit other objects (i.e leaves) on their way down. Two non terminal velocity heights were chosen; 1cm and 1m. There were more discrepancies between the results for the same drop sizes at a height of 1cm compared to 1m.

6.1.3 Testing with other Sensor Types

The third testing procedure was used to verify whether the drilling method used to simulate drops of different sizes was in fact producing drops of the correct size. An infrared transmitter and receiver were placed over the rainfall sensor so that it would be in the path of drops falling onto the sensor. As the infrared sensor detects the area of the drop and not the impact force, it could be easily calibrated by initially dropping droplets from close range. From this, it was found that indeed the size of the drops created were between 0.1 mm and 6mm. This testing was also done to reinforce the design choice of using piezoelectric sensors over infrared sensors. It was found that infrared sensors can only sense a much smaller area of rain as compared to the prototype. To overcome this, more transmitters and receivers would have to be used. However, this would not only increase the cost of the product (going against the design specification) but also increase the complexity of data processing.

6.1.4 Functional Testing

Lastly, after improving the prototype based on earlier testing procedures, the final prototype was tested against a rain gauge. This was done to see if the sensor was indeed a viable product compared to a product currently being used in the market. The discrepancies between the sensor and rain gauge results was also used to further calibrate the sensor for improved accuracy. Testing against the rain gauge was done in two stages. Firstly, it was tested indoors by dropping water droplets from the 5th storey of the building. This method allowed it to be tested for differing rain drop sizes. When the performance of the sensor proved to be satisfactory, the sensor was tested for a short period of time in real rain. Due to non ideal weather conditions, this testing method could not be carried out more than once. The sensor itself requires design modifications to be tested properly in real rain for long periods of time. For example, prototyping the sensor and bluetooth module on a breadboard made it difficult to test under rain conditions. A PCB design would be ideal in minimising all component sizes such that they easily fit in the casing. The casing is also currently modelled to be placed on top of Climate Edge's weather station, which provides the sensor further rigidity and stability. In the indoor testing, the sensor was able to detect much smaller rain depths as it is more sensitive. For larger rain depths, both data were similar, therefore proving the sensor's viability as a precipitation sensor.

6.1.5 Calibration Framework

The testing data is then used to calibrate the sensor. A calibration framework is put in place such that the test data can be easily processed. This allows the sensor to be easily modifiable based on future test data. The calibration framework is created in Matlab. The framework pogram simply reads the new test data, merges it with the old test data and finds the optimal parameters for the power equation for calibration. The Matlab function `fit` finds the optimum parameters by minimising the R-squared value. The Matlab code can be found Appendix D.

6.1.6 Power Consumption Testing

The final testing stage would be to monitor the power consumption of the sensor. As there is limited energy supply at the farm site, this is crucial in determining if the product would be viable at a farm site. This is carried out using an oscilloscope and power supply unit. A shunt resistor is used along with the scope to measure current more accurately. The voltage can simply be measured using the power supply unit, or can also be connected to the scope for more accurate readings. This testing was not only performed on the final prototype, but also on previous prototypes that did not include any power saving functionality so as to understand the effect of these power saving functionalities.

6.2 Phone Application

6.2.1 Functional Testing

The testing of the application was carried out in various stages. The first stage is the compatibility testing which checks if the application can be used on older devices and OS versions. As this is a design requirement, the results will be useful in deciding if the application was a success. The second stage involves testing memory usage and file and resource management over a continuous running period of day. This helps reaffirm the reliability of the product and helps ensure that the program does not crash due to memory mismanagement. The third stage checks the integrity of the information transferred to the phone and servers over a period of one day as this is crucial for the usability of the product. The fourth stage of testing involved ensuring that the GUI(graphic user interface) is fully functional and usable by most people. The fifth and most important testing stage involves the feeding of all possible input types/combinations. This tests extreme cases which could potentially cause glitches in the application, thus allowing for fixing of these glitches and making the application more user-friendly and reliable.

6.2.2 Power Consumption Testing

The main drawback of using bluetooth communication is the additional power consumption which is inherent with any form of wireless data transfer. To test if regular bluetooth (BT2.1) would be feasible, testing was done with a HC-05 module and an Arduino Uno. For the HC-05 to be deemed a viable solution, the power consumption has to be kept reasonably low so that it can run off the weather station's solar panel and battery storage. The current and voltage parameters are measured using an oscilloscope, similar to the power consumption testing procedure for the sensor.

6.2.3 Memory Usage Testing

As it isn't possible to directly measure how much drain an individual application is placing on the battery, the two metrics used to measure efficiency were memory usage and CPU utilisation. In both cases the lower the utilisation, the better. This testing procedure was achieved using Android Studio.

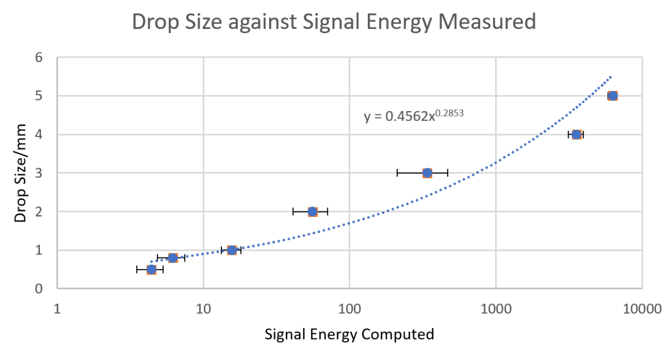
6.3 Sensor and Application Testing

Once both the sensor and the phone application were interfaced with one another, testing was carried out to ensure that the prototype was reliable and functioned as desired. This was done by checking the integrity of data shown on the phone. The actual data output of the sensor was monitored using the Serial Monitor in Arduino IDE. Once the data had been transmitted to the phone, the values are compared to ensure they are the same. Instead of dropping water droplets from a height, this testing was carried out with a water spray. This is because the bluetooth module and intermediary microcontroller are not weather resistant. This was not required by the design specification either as they are to be installed within the weather station which would provide weather protection for these electronics.

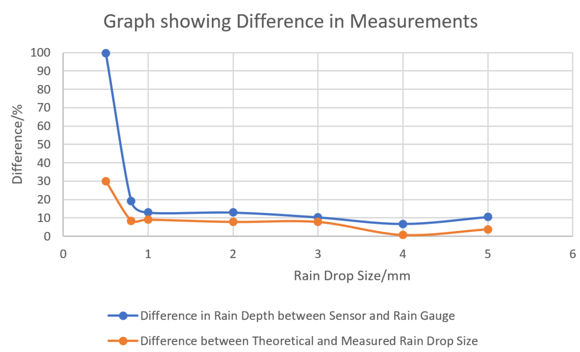
7 Results and Discussion

7.1 Precipitation Sensor

The results of the calibration testing procedure and functional testing procedure are found in Figures 13a and 13b respectively. The raw data can be found in Appendix H, tables 9 and 8 respectively.



(a) Figure showing results from Testing procedure 1



(b) Figure showing results from Testing procedure 4

Figure 13: Figure showing test data

The calibration testing procedure identifies the relationship between the drop size and signal energy. Figure 13a shows the relationship exists through the equation displayed on the graph. This equation is then programmed into the microcontroller. The equation is determined by the calibration framework. Also on the graph are error bars that display the variance in the data set. There is a major overlap between drop sizes of 0.5mm and 0.8mm, primarily due to their similarity in mass and velocity. Moreover, it is not needed to detect the difference between 0.5mm and 0.8mm drops as it makes little difference to the impact on soil erosion. It can be noted that is no overlap between the other measurements, thus proving that the sensor can determine conclusively the size of the drop majority of the time.

The functional testing procedure compares the rain gauge to the sensor in an indoor testing environment. This test is crucial in determining the accuracy of the sensor with respect to the rain gauge. Figure 13b shows the percentage difference in readings. The average difference in sensor reading and rain gauge reading based on 4 tests is generally 10%, except at a rain drop size of 0.5mm where the difference is almost 100%. This is mainly due to the lack of sensitivity of the rain gauge. The rain gauge can only read a minimum of 0.05mm while the sensor can read as low as 1 um of rain depth. The graph also shows the difference between the rain drop size dropped from a height and that registered by the sensor. The average error is approximately 10%. Based on the testing data, it can be seen that the sensor can provide accurate readings for rain drop size and rain depth. Moreover, the sensor has also proved to be more sensitive than the rain gauge while providing more rain information than the rain gauge.

Further testing done to identify the possibility of reading errors showed that for very small drop heights, the discrepancy in data is significant.

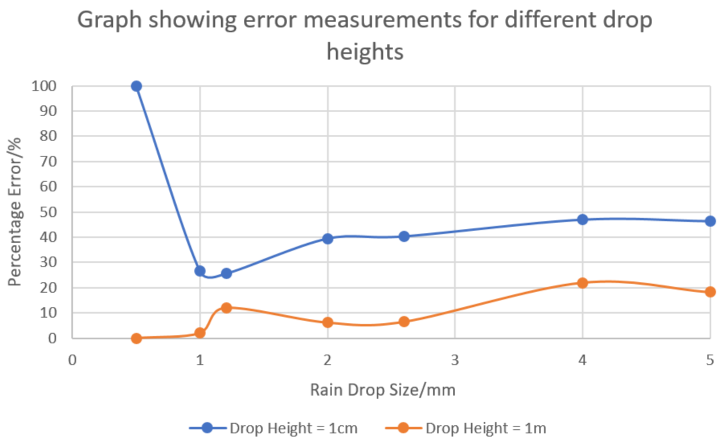


Figure 14: Figure showing error measurements for different drop heights

Figure 14 shows the percentage error between the measured data and theoretical data for drop heights of 1m and 1cm. As can be seen in the figure, the error for drop heights of 1cm is substantial. As explained in the design outline, it is safe to assume these events have a very low probability of occurrence. Therefore, computing the moving average of the energy will suppress the effect of such erroneous readings on the dataset.

The final testing, namely power consumption testing proved the low power capabilities of the sensor. The raw data can be found in Table 1 and as mentioned earlier, the power saving functionalities were a success as they reduced power consumption by 99% to 9.7Wh per year. The 0.03Wh power consumption per day is negligible in comparison to the 70Wh generated by the solar panel daily.

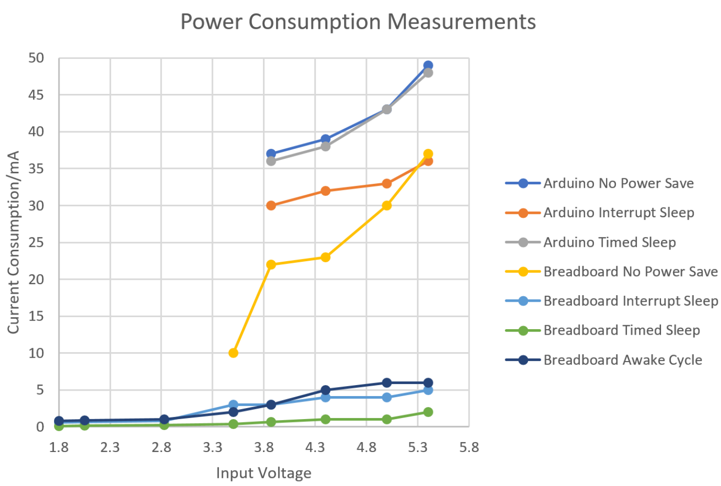


Figure 15: Figure showing input voltage and current consumption with power saving functionalities and without

Figure 15 summarises the power consumption data.

7.2 Phone Application

Due to the fact that the application logs all of its data to a file and any incorrectly stored data is ignored, no issues were found in the functional testing. The application was able to display the correct data and performed satisfactorily when testing extreme conditions.

As for the power consumption, the bluetooth module was initially drawing 42mA of current at an operating voltage of 3.3V. The reason this is so large is due to the fact that the module when unconnected will constantly poll for a device to connect to. Once a connection has been established with an Android device, the current drawn drops to 5mA. Despite this massive improvement, the power saving benefits of leaving the bluetooth module disconnected from the phone outweigh this. Reducing the polling rate can be done using AT commands which can lower the disconnected current draw.[35]

Measuring current drawn when transmitting data is more complex than measuring when idle due to current fluctuations. To test data transfer, values were transmitted with delays of 10ms between transmissions to prevent bluetooth activity crashing. This drew an average of 26mA which is rather high but due to data only being transferred once every hour for a very short period of time, this power consumption spike is negligible.

In an attempt to reduce power consumption, the effect of reducing the voltage to the HC-05 board was investigated. The supply voltage was decreased in 50mV increments until the power LED failed to light up at 2.65V. Even though the LED was lit at 2.70V, an actual connection couldn't be established until supply voltage was increased to 2.80V irrespective of distance between phone and bluetooth module. The effects of this reduction are illustrated in Table 3. The benefit may be slight but it

Table 3: Table showing power consumption data

HC-05 Input Voltage	Idle Current	Transmission Current	Unconnected Current
3.3V	16mA	24mA	38mA
2.8V	14mA	22mA	35mA

could prove significant due to the power constraints. As the distance between the Android device and the weather station isn't of importance, decreasing the voltage shouldn't negatively impact operation. Therefore, unless further testing proves otherwise and a stable 2.8V source can be generated, this reduced voltage should be used to reduce power consumption. The bluetooth module now uses 0.9Wh per day which is still minimal compared to the energy generated.

The main drawback of using an android device as the "brains" of the weatherstation is the power consumption. Part of this can be helped by rooting the device, removing any unnecessary applications or processes and installing a custom kernel. These will help but the application itself must be made as efficient as possible to further reduce power usage.

The unoptimised application refreshed data every 10 seconds for which can be clearly seen by peaks in CPU usage in Figure 16a. At 6005s in, the screen was turned off showing the decreased memory usage caused by not having to draw the user interface. The memory when the screen was switched off fluctuated between 12MB and 26.5MB with CPU utilisation at around 12% by user and 2.5% by the Kernel with occasional even larger peaks.

The memory usage was particularly unusual with it gradually rising until it reaches the free memory limit where it then drops down again. These sudden drops in memory usage can be explained by Java's built in feature called Garbage Collection which is a form of automatic memory management of objects no longer being used.[36] Having garbage collection events occurring so frequently is a sign of poor optimisation. These events are occurring because the bluetooth connection is continuously connected and objects are repeatedly being created.

To improve the application's performance the continuous connection was replaced with a periodic connection, interlaced with periods of system sleep, where data is transmitted every time a bluetooth connection is established. Another optimisation which was applied was to prefer static final to define constants so they can be referenced later on using field lookups.[37] The optimised metrics can be seen in Figure 16b.

(a) Figure showing results for unoptimised application

(b) Figure showing results for optimised application

Figure 16: Figure showing memory usage and CPU utilisation of application

The memory when the screen was switched off remained relatively steady around 10MB with CPU utilisation approximately nonexistent except at periodic peaks with similar usage as the unoptimised application during these peaks. If this system was to actually be implemented, these peaks would only occur every hour making the average CPU usage incredibly low, and therefore making the application very efficient.

The final optimisation was to ensure that data sent to the server via GPRS was kept to a minimum and done in batches as transmitting data over long distances will always be power consuming. This was achieved by identifying data from different sensors using single character tags rather than strings. The actual data transfer is then done once an hour when all the new data will be sent together.

8 Limitations

8.1 Precipitation Sensor

The main limitation of the rainfall sensor is the lack of calibration data. The sensor is currently calibrated to the surrounding conditions (London) where it was tested. However, there will be a disparity with the conditions on the farm sites. The disparity stems from different rainfall patterns but also difference in temperature, humidity and wind patterns which will have an effect on sensor readings. General weather conditions of Nicaragua can be found but this is not specific enough to the farm site, thus it will still lead to calibration errors. Thus, further testing and tweaking will need to be carried out to ensure optimal performance.

Another limitation faced was the lack of testing facilities. Rainfall simulation laboratories will allow easy testing of the precipitation sensor. Different rainfall types can also be simulated more accurately. Although the method of manually dropping water from a certain height was sufficient for calibrating the sensor effectively, it was time consuming and difficult to execute. Therefore, it will be useful to carry out further testing in a rainfall simulation laboratory.

From research, it was discovered that the piezoelectric sensor has the limitation of not being able to sense very tiny drops of rain.[38] The cut off below which it is unable to sense readings is 0.5mm rain drops. This was later verified in testing. Raindrops typically vary between 0.1mm and 5mm thus there is small window of readings which the piezoelectric sensor will not sense. However, below 0.5mm the rain pattern is a very light drizzle and as such will not have an effect on the farm sites or in an agricultural context. Therefore, this limitation is acceptable and negligible.

The piezoelectric sensor is also subject to radius effect which describes that the further away the rain drop from the centre of the sensor, the weaker the signal read. This means that the strength of the signal from different locations on the sensor will be different. This non uniformity will lead to errors in calculating the total volume of rain measured.

Piezoelectric sensors also experience the puddle effect. Water collecting on the surface of the sensor will lead to errors in future readings. This happens because subsequent drops hitting the collected water (puddle) are more likely to splatter. Secondly, the puddle also dampens the force of subsequent drops leading to a lower reading than what is expected. Although means to aid water roll off have been implemented, the roll off is slow and not instantaneous. As such, there will still be some errors due to the puddle effect.

8.2 Phone Application

From the perspective of the application, there are several major obstacles that need to be overcome before a full “roll-out” will be possible.

The main limitation of the phone application is the power consumption. Although power consumption is a major client requirement, it is difficult to reduce power consumed by smart phones to that of PCBs with minimal electronics due to the myriad of background processes. Moreover, using older versions of smart phones make it even more difficult to implement as they do not have power saving functionalities such as `Doze` and `App Standby` that were recently released for Anroid 6.0.[39]

Moreover, the application is still very much under development in terms of its overall look/functionality, as well as stability. While the individual modules such as graphs, server communication, GUI work well, the ensemble functionality still has flaws and bugs to iron out. In terms of the latter, the application is prone to occasional crashing. Although the stability has been greatly improved as part of the design process, there are still tweaks to be made. As one of our core design specifications is reliability and robustness, the aforementioned problems would have to be fully resolved before rolling out the system commercially.

Following up on the previous point, the application has only really been tested on one phone model. As there are myriads of android OS versions and device models available (and since Climate Edge is targeting them all), substantial testing will have to be carried out on as many phone models as possible, particularly the older versions which are more likely to be used by farmers in developing areas. Such tests would yield greater insight into factors such as: power consumption, performance, reliability and long term stability of the system. Using the results of these trials, compatibility issues would be ironed out prior to commercial roll out.

Furthermore, in terms of the server side communication/database, it would be beneficial and even perhaps imperative to set up and host a dedicated server as the current prototype operates in an ad-hoc fashion. Having a dedicated system in place would allow for more control over the system from Climate Edge's perspective. Additionally, while the current database system works well, it would be worth collaborating with a specialist in order to optimise the system by making it more stable and efficient than it currently is.

In addition, an effective method of delivering software updates would have to be determined. While having the application on the Google store allows for seamless updates in normal cases, the fact that the phones will be placed in weather stations renders this void. Perhaps setting up periodic maintenance sessions every few months would be an adequate measure.

Finally, the application does not support the capability to add new sensors in a "plug and play fashion". Despite being written in a modular way, the app still requires a certain amount of development effort in order to add additional sensors to its capability (GUI, interfacing sensor with app). Creating an easy to use, inbuilt method of configuring new sensors would greatly benefit the end user, allow for more customizability of weather stations and reduce the amount of technical support by the company to its users, thus cutting costs.

9 Future Work

9.1 Precipitation Sensor

Moving forward, the sensor should be tested under controlled conditions in a rainfall simulation laboratory. With this, the readings taken for different drop sizes of rain will be more precise as the differing conditions can be simulated to exactly mimic real rain. This is an improvement on the indoor testing procedure as it more closely resembles rain and is better than testing in real rain as external factors can be controlled. The effects of changing these external factors such as wind, temperature and humidity on the performance of the sensor will be noted. Using a rainfall simulation lab will also give more flexibility in testing allowing for repeated readings as there will be no need to wait for a rainy day to proceed.

Currently, the sensor prototype utilizes a strip board and breadboard to house the electronics and for connection to the Arduino. This is not feasible for mass productions and affects the compactness of the design. A single module is instead needed to ensure reproducibility. Thus, the use of a PCB will downsize connections by combining the different components into one unit and as such achieving the desired compactness. This will remove any exposed connections as the PCB can be housed inside the sensor casing itself preventing any damage to the electronics caused by rainfall and other weather conditions. This is an important future work because resistance to weather and scalability was a key client requirement as part of the design specification.

9.2 Phone Application

Firstly, the main improvement would come in form of a modularisation of the sensor/application interface that would allow the farmer/user to add new sensors in a "plug and play" fashion. Currently, adding new sensor is relatively easy, but still requires substantial programming and interfacing effort. The improvement would aim to eliminate this hassle and open up possibilities of using both new sensor types and manufacturers, thus expanding Climate Edge's capabilities and making the business more competitive.

Another major area of improvement would be expanding the current device support range to older phones known as "dumb" phones. A device such as Nokia 3310 possesses hardware that is more than adequate for the task of logging, displaying and transferring sensor data via GPRS.[40] Additionally, such devices are physically rugged, highly accessible and affordable in poorer countries and consume significantly less power than the cumbersome and multi-functional smart phone prototypes, thus making them ideal for a farm setting. Although some redevelopment would have to be done, coding for such devices is not particularly challenging and would greatly expand Climate Edge's capabilities at a relatively low effort and material cost.

When dealing with smartphones, another possibility would be trying to conserve the power by gaining root access of an android based device (known as "rooting") and disabling unessential and power consuming services. The biggest challenge with this step would be the difficulty of reaching many customers. Rooting is not an easy task, especially for a layman. Perhaps such a service could be made available only for customers willing to pay for it.

Finally, Climate Edge's service could be further expanded by providing the farmers with tailored recommendations based on the data obtained. With substantial coverage, Climate Edge has the potential of obtaining large amounts of agricultural/environmental data spread across a very wide geographical area. This opens up several lucrative opportunities within the fields of Data Analytics and Machine Learning. In other words, Climate Edge could use the data to both advise individual farmers that are already part of the system, as well as make predictions and recommendations to both new customers and as a separate service to 3rd party customers.

10 Project Management and Organisation

The project was initially split into 2 teams, one that focused on the precipitation sensor and on that focused on the phone application. Meetings for each team were held individually with less frequent group meetings to discuss progress. Near the end of the project, both teams rejoined to work on interfacing the sensor with the phone application. The following gantt chart in Figure 17 enabled the team to keep track of internal and external deadlines.

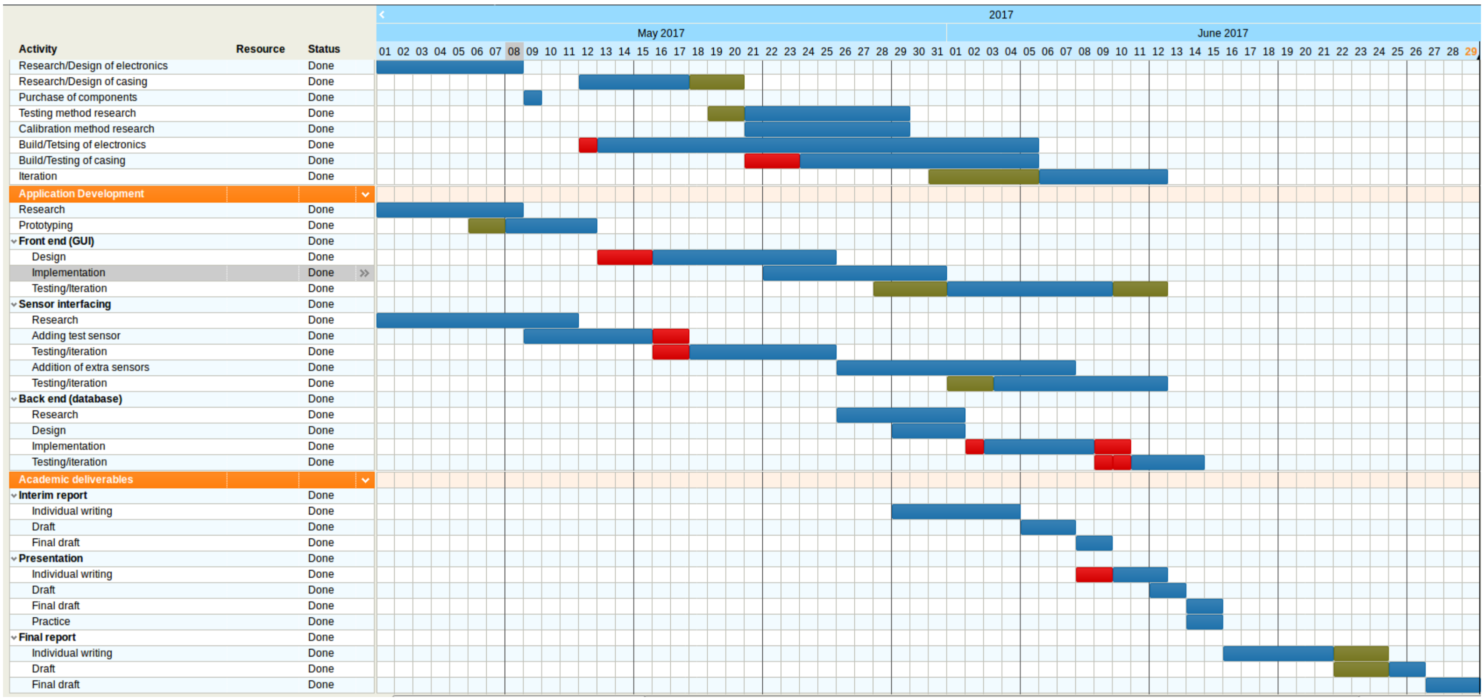


Figure 17: Figure showing gantt chart used for project organisation

11 Conclusion

In conclusion, the final product has shown to meet most of the design requirements. For the precipitation sensor, testing results have shown that the data is reliable and the sensor itself consumes negligible power. Moreover, the material of the casing and simplicity in casing design ensures the sensor's scalability and durability. The overall cost also reflects the affordability of the product.

As for the phone application, compatibility with older smart phones and suitability for use in the weather station have been achieved. However, power consumption of the smart phone may not necessarily be supported by the solar panel. A possible alternative would be to switch to "dumbphones" that consume much less power. The implementation of the application in the weather station depends on what Climate Edge deems as more important, power consumption or cost.

Although the idea of measuring and processing weather data and sending the information to a phone application is not novel, the methods carried out by this project to minimise cost and power consumption ensures great potential for this integrated product on the market. Moreover, this integrated product enables Climate Edge to empower farmers in developing countries, who do not have the advanced resources of farmers from developed countries. With future work into improved testing procedures, usage of PCBs, and rooting of smart phones, this project can soon become a reality.

References

[1] "Bringing data to action." http://climate-edge.co.uk/. Accessed: 2017-05-04.

[2] "Climate change dries up nicaragua." https:http://www.ipsnews.net/2016/04/climate-change-dries-up-nicaragua/. Accessed: 2017-06-19.

[3] "10w solar panel kit." https:http:http://www.maplin.co.uk/p/10w-solar-panel-kit-l14br. Accessed: 2017-06-27.

[4] F. M. DaMatta, C. P. Ronchi, M. Maestri, and R. S. Barros, "Ecophysiology of coffee growth and production," 2007.

[5] "Coffee and climate." https://www.climate.gov/news-features/climate-and/climate-coffee. Accessed: 2017-05-05.

[6] "Climate change predicted to halve coffee growing area that supports 120m people." https://www.theguardian.com/environment/2016/aug/29/climate-change-predicted-to-halve-coffee-growing-area-that-supports-12 Accessed: 2017-05-20.

[7] A. Moussouni, L. Mouzai, and M. Bouhadef, "The effect of raindrop kinetic energy on soil erodibility," 2014.

[8] "Rain gauge." https://en.wikipedia.org/wiki/Rain_gauge#Tipping_bucket_rain_gauge. Accessed: 2017-05-04.

[9] E. Trono, M. Guico, N. Libatique, G. Tangonan, D. Baluyot, T. Cordero, F. Geronimo, and A. Parrenas, "Rainfall monitoring using acoustic sensors," 2012.

[10] "Hydreon optical rain sensor." http://rainsensors.com/. Accessed: 2017-05-04.

[11] S. de Jong, "Low cost disdrometer." http://www.citg.tudelft.nl/index.php?id=20113&L=1. Accessed: 2017-05-04.

[12] "Introducing arable mark." http:http://arable.com/. Accessed: 2017-06-08.

[13] "Smart vineyard." http://smartvineyard.com/home/. Accessed: 2017-06-08.

[14] "Pic12f1840 datasheet." http://ww1.microchip.com/downloads/en/DeviceDoc/40001441F.pdf. Accessed: 2017-06-07.

[15] "Bluetooth low energy." https://developer.android.com/guide/topics/connectivity/bluetooth-le.html. Accessed: 2017-05-05.

[16] "Android developers dashboard." https://developer.android.com/about/dashboards/index.html. Accessed: 2017-05-03.

[17] "Bluetooth 4.0 ble module datasheet v507." http://wiki.microduinoinc.com/images/f/fc/Bluetooth40_en.pdf. Accessed: 2017-05-18.

[18] "Lg nexus 4 e960." http://www.gsmarena.com/lg_nexus_4_e960-5048.php. Accessed: 2017-05-05.

[19] "Smartphone headset standards: Apple iphone, ahj and omtp." https://longtailproducts.zendesk.com/hc/en-us/articles/207970396-Smartphone-Headset-Standards-Apple-iPhone-AHJ-CTIA-OMTP. Accessed: 2017-05-05.

[20] "Piezoelectricity." https://en.wikipedia.org/wiki/Piezoelectricity. Accessed: 2017-05-01.

[21] Z. Chu and S. Seeger, "Superamphiphobic surfaces," 31st January 2014.

[22] J. E. Lane, T. Kasparis, P. T. Metzger, and W. L. Jones, "Laser calibration of an impact disdrometer," 2016.

[23] "High impact polystyrene, british plastic foundation." http://www.bpf.co.uk/plastipedia/polymers/HIPS.aspx. Accessed: 2017-06-20.

[24] D. R. Weast, Robert C.and Lide, "Characteristics of particles and particle dispersoids," 1981.

[25] "Hot melt adhesive." https://en.wikipedia.org/wiki/Hot-melt_adhesive. Accessed: 2017-06-10.

[26] "Atmega328/p datasheet." http://www.atmel.com/Images/Atmel-42735-8-bit-AVR-Microcontroller-ATmega32 Datasheet.pdf. Accessed: 2017-06-05.

[27] S.C.Warude, P.R.Unhale, S.P.Khandagale, A.D.Waykar, and S.S.Gaonkar, “Harnessing of kinetic energy of raindrops,” 2015.

[28] D. Dunkerley, “Rain event properties in nature and in rainfall simulation experiments: a comparative review with recommendations for increasingly systematic study and reporting,” 2008.

[29] “Transmitting network data using volley.” https://developer.android.com/training/volley/index.html. Accessed: 2017-04-23.

[30] “Asynctask.” https://developer.android.com/reference/android/os/AsyncTask.html. Accessed: 2017-04-23.

[31] “Transmitting network data using volley.” https://www.techopedia.com/definition/6308/cache-miss. Accessed: 2017-04-23.

[32] “Cache hit.” https://www.techopedia.com/definition/6306/cache-hit. Accessed: 2017-04-23.

[33] “Hash map.” https://developer.android.com/reference/java/util/HashMap.html. Accessed: 2017-04-23.

[34] G. Foote and P. Toit, “Terminal velocity of raindrops aloft,” 1969.

[35] “Hc-03/05 embedded bluetooth serial communication module at command set.” http://www.linotux.ch/arduino/HC-0305_serial_module_AT_commamd_set_201104_revised.pdf. Accessed: 2017-04-23.

[36] “Java garbage collection basics.” http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html. Accessed: 2017-05-15.

[37] “Performance tips.” https://developer.android.com/training/articles/perf-tips.html. Accessed: 2017-05-10.

[38] G. Kathiravelu and a. P. N. Terry Lucke, “Rain drop measurement techniques: A review,” 2016.

[39] “Optimising doze and app standby.” https://developer.android.com/training/monitoring-device-state/doze-standby.html. Accessed: 2017-06-22.

[40] “Nokia 3310 full specs.” https://www.nokia.com/en_int/phones/nokia-3310. Accessed: 2017-06-15.

Appendix

A Sensor Code

```
/*
Project: Climate Edge
Group: 9
Author: Vanita K
Function: Rain Sensor

Overall Function:

The program reads from 4 piezoelectric sensors and transforms the signal into useful rain information

Functions:
ISR (ANALOG_COMP_vect)- interrupt vector for analog comparator
ISR(WDT_vect)- interrupt vector for watchdog timer(internal timer)
configure_wdt- configuration of watchdog timer(internal timer)
configure_analog_comp- configuration of analog comparator
sleepModeRain- sleep mode while raining
sleepModeNoRain- sleep mode when no rain
calcEnergy- claculates energy of signal
signalMapper- maps signal energy to raindrop size and rain depth
dataReadWrite- temporarily stores data for one cycle and varies read write time cylces


*/


#include <avr/wdt.h>            // library for default watchdog functions
#include <avr/interrupt.h>      // library for interrupts handling
#include <avr/sleep.h>          // library for sleep
#include <avr/power.h>          // library for power control
#include <EEPROM.h>             // library for storing data in EEPROM
#include <SoftwareSerial.h>     // library for bluetooth data transfer
#include <Wire.h>

#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit)) // define functions for ADC sampling cycle set
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))

/* Global Variables */

/* Sleep Timers */
int sleepCyclesLeft;  // tracks sleep cycles left
int sleepCyclesNo=1; // sets number of sleep cycles (1 min to 5 min---> 5 to 32)

/* Awake Cycle Timers */
unsigned long timeStart = 0; // tracks start time of each awake cycle
unsigned long timeNow;  // tracks current time
unsigned long timeInterval=3000; // sets time for awake cycle -> time scaled w.r.t clock (1 min to 5 min---> 7500 to 37500)
volatile boolean triggered = false; // checks for interrupt from analog comparator

/* Energy Computation Variables */
unsigned int piezoE = 0; // energy of signal
int peakCount = 0;       // number of signals
const int peakThresh = 3; // threshold for signal
int piezoMaxArray[] = {0,0,0,0}; // tracks maximum peak value
```

```
float dropSizeInst = 0.00; // tracks drop size
float rainDepth = 0.0000; // tracks overall rain volumev
float rainConst = 0.3; // rain constant dependent on sensor area /

//analog comparator interrupt vector
ISR (ANALOG_COMP_vect)
  {
  triggered = true;
  }

// watchdog timer interrupt vector
ISR(WDT_vect)
{
 wdt_reset();
}

//configures analog comparator
void configure_analog_comp(){

  ADCSRB = 0;           // disable analog comparator multiplexer enable
  ACSR =  bit (ACI)     // clear analog comparator interrupt flag
        | bit (ACIE)
        | bit (ACIS1);  // select analog comparator interrupt mode- trigger on falling edge
}

// configures watchdog timer
void configure_wdt(void)
{

  cli();                              // disable interrupts for changing the registers

  MCUSR = 0;                          // reset status register flag
  WDTCSR |= 0b00011000;
  WDTCSR =  0b01000000 | 0b100001; // set watchdog timer to 8 seconds per cycle

  sei();                              // re-enable interrupts

}

// put the Arduino to sleep while raining
void sleepModeRain(int sleepCycles)
{
 // configure the watchdog
  configure_wdt();

  sleepCyclesLeft = sleepCycles; // defines how many cycles should sleep

  // Set sleep to full power down
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);

  while (sleepCyclesLeft > 0){ // while some cycles left, sleep

  // Enable sleep and enter sleep mode
  sleep_enable();
  sleep_mode();

  // When awake, disable sleep mode
  sleep_disable();

  // Reduce number of sleep cycles left
  sleepCyclesLeft = sleepCyclesLeft - 1;
```

```

  }

  // set timings to determine time program samples
  //Serial.println("Waking Up");
  timeStart = millis();
}


// Put Arduino to sleep when not raining
void sleepModeNoRain(){

  // Set sleep to idle power down - enables analog comparator interrupt
  set_sleep_mode (SLEEP_MODE_IDLE);

  // Turn off everything while asleep
  power_all_disable();

  // Enable sleep and enter sleep mode.
  sleep_enable();
  sleep_mode();

  if( triggered ){

    // When awake, disable sleep mode
    triggered = false;
    sleep_disable();
    power_all_enable();

    //Serial.println("Triggered");
    //delay(100);
    }

  }

// Compute approximate energy of signal
void calcEnergy(int piezoV, int sensorNo){
  if(piezoV != 0) {//checks for signal
    if( piezoV > piezoMaxArray[sensorNo]) {
      piezoMaxArray[sensorNo] = piezoV; // finds maximum value of signal
    }
  } else {
    if (piezoMaxArray[sensorNo] > peakThresh) {// ensures maximum peak value above noise threshold
      peakCount++;  // increments number of peaks
      piezoE= (sq(piezoMaxArray[sensorNo]) + piezoE)/2; // computes moving average signal energy(simplified form)
    }
    piezoMaxArray[sensorNo]=0; // sets peak value tracker back to 0 once peak detected

  }
}

// Map signal energy to drop size and rain depth
void signalMapper(){

   if(piezoE != 0){
    dropSizeInst = (0.4562*pow(piezoE,0.2853)); // compute instantaneous drop size
    rainDepth = rainConst*pow(dropSizeInst, 3)*peakCount; // compute total rainfall depth in um(micro metre)
   }
```

```
  }


// Read data from awake cycle and store value
void dataReadWrite(int piezoValue0, int piezoValue1, int piezoValue2){


  int EEPROMval0 = (int) EEPROM.read(0); // read previous drop size
  int EEPROMval1 = (int) EEPROM.read(1); // read previous peak count
  int EEPROMval2 = (int) EEPROM.read(2); // read previous rain depth
  int diff = (EEPROMval2 - piezoValue2) / EEPROMval2; // check if there is a difference in
      rain depth

   // send data via I2c bus to microcontroller
   Wire.beginTransmission (42);
   I2C_writeAnything (piezoValue2);
   I2C_writeAnything (piezoValue0);
   Wire.endTransmission ();
   delay (200);

  if (piezoValue0 == 0) { // no peaks --> no rain --> sleep

    // configure analog comparator again for sleep
    configure_analog_comp();
    delay(200);

    // sleep until rain interrupts
    digitalWrite(13, LOW);
    sleepModeNoRain();

  }

  else if(EEPROMval1 == 0){ // input first value
    EEPROM.write(0, (byte) piezoValue0);
    EEPROM.write(1, (byte) piezoValue1);
    EEPROM.write(2, (byte) piezoValue2);
    }

  else if(diff > 0.2 or diff < -0.2){ // large difference in measurements
    EEPROM.update(0, (byte) (piezoValue0 + EEPROMval0)); // update measurements
    EEPROM.update(1, (byte) (piezoValue1 + EEPROMval1));
    EEPROM.update(2, (byte) (piezoValue2 + EEPROMval2));
    timeInterval = max(7500, (1 - diff)*timeInterval); // compute new cycle timings
    sleepCyclesNo = (1 - diff)*sleepCyclesNo;
    }
  else if(diff < 0.2 or diff > -0.2){ // no difference in measurements
    EEPROM.update(0, (byte) (piezoValue0 + EEPROMval0));
    EEPROM.update(1, (byte) (piezoValue1 + EEPROMval1));
    EEPROM.update(2, (byte) (piezoValue2 + EEPROMval2));
    timeInterval = 37500; // new cycle timings set back to maximum
    sleepCyclesNo = 32;
    }
  }

// Clear data at startup
void EEPROM_clear(){
  for(int i=0; i < 3; i++){
      EEPROM.write(i,0);
    }
  }

```

```
// Convert float data into bytes for I2C
unsigned int I2C_writeAnything (const &value)
  {
  Wire.write((byte *) &value, sizeof (value));
  return sizeof (value);
  }

// Setup at switch on
void setup(){

  //set baud rate for serial port
  Serial.begin(76800);

  // configure analog comparator
  configure_analog_comp();

  // clock prescaler setup
  CLKPR = _BV(CLKPCE);  // enable change of the clock prescaler
  CLKPR = _BV(CLKPS0) | _BV(CLKPS1) ;  // divide frequency by 4, 4MHz clock

  // clear EEPROM
  EEPROM_clear();

  // turn off brown-out enable (low voltage detection)
  MCUCR = bit (BODS) | bit (BODSE);
  MCUCR = bit (BODS);

  //set inputs and outputs
  pinMode(6, INPUT); // set ADC comparator pins as inputs
  pinMode(7, INPUT);
  pinMode(13, OUTPUT); // set pin 13 as output
  pinMode(A0, INPUT); // set ADC pins as inputs
  pinMode(A1, INPUT);
  pinMode(A2, INPUT);
  pinMode(A3, INPUT);

  // flash to signal end of configuration
  digitalWrite(13, HIGH);
  delay(100);
  digitalWrite(13, LOW);

  // sleep until interrupt detected/starts raining
  sleepModeNoRain();

  // set ADC clock in acceptable range (50-200kHz)
  sbi(ADCSRA, ADPS2);
  cbi(ADCSRA, ADPS1);
  sbi(ADCSRA, ADPS0);

  // check time at start of program
  timeStart = millis();

  // light ON to signal awake cycle
  digitalWrite(13, HIGH);

  // set I2C bus for data transfer
  Wire.begin ();
}
```

```

// loop time dependent on clock time and program time
void loop(){

    timeNow=millis(); // set current time during program

    //perform ADC
    digitalWrite(12, HIGH);
    calcEnergy(analogRead(A0), 0);
    digitalWrite(12, LOW);
    calcEnergy(analogRead(A1), 1);
    digitalWrite(12, HIGH);
    calcEnergy(analogRead(A2), 2);
    digitalWrite(12, LOW);
    calcEnergy(analogRead(A3), 3);

    if( (timeNow - timeStart) >= timeInterval ){ // if awake cycle time over

    signalMapper(); // map the signal energy and peak count to rain variables

    dataReadWrite(dropSizeInst, peakCount, rainDepth); // write variables to EEPROM

    digitalWrite(13, LOW); // light OFF to signal sleep cycle

    sleepModeRain(sleepCyclesNo); // go to sleep during rain

    digitalWrite(13, HIGH); // light ON to signal awake cycle

    peakCount = 0; // set variables to 0 for next awake cycle
    dropSizeInst = 0;
    rainDepth = 0;
    piezoE = 0;
  }

}
```

B Phone Application Code

B.1 Main Activity

```
package com.example.keyan.cet;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.DialogInterface;
import android.content.Intent;
import android.graphics.Color;
import android.os.Build;
import android.os.Handler;
import android.support.design.widget.TabLayout;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v4.view.PagerAdapter;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;

import android.support.v4.app.Fragment;
```

```java
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;
import android.support.v4.view.ViewPager;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;

import android.widget.Button;
import android.widget.EditText;
import android.widget.Switch;
import android.widget.TextView;
import android.widget.Toast;

import com.android.volley.AuthFailureError;
import com.android.volley.Request;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.StringRequest;
import com.github.mikephil.charting.charts.LineChart;
import com.github.mikephil.charting.components.Legend;
import com.github.mikephil.charting.components.XAxis;
import com.github.mikephil.charting.components.YAxis;
import com.github.mikephil.charting.data.Entry;
import com.github.mikephil.charting.data.LineData;
import com.github.mikephil.charting.data.LineDataSet;
import com.github.mikephil.charting.interfaces.datasets.ILineDataSet;
import com.github.mikephil.charting.utils.ColorTemplate;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import java.util.UUID;

import static android.support.v4.view.PagerAdapter.POSITION_NONE;
import static java.lang.Boolean.FALSE;
import static java.lang.Boolean.TRUE;

public class MainActivity extends AppCompatActivity {

    /**
     * The {@link android.support.v4.view.PagerAdapter} that will provide
     * fragments for each of the sections. We use a
     * {@link FragmentPagerAdapter} derivative, which will keep every
     * loaded fragment in memory. If this becomes too memory intensive, it
     * may be best to switch to a
     * {@link android.support.v4.app.FragmentStatePagerAdapter}.
     */
    private SectionsPagerAdapter mSectionsPagerAdapter;

```

```
    /**
     * The {@link ViewPager} that will host the section contents.
     */
    private ViewPager mViewPager;

    private static final UUID PORT_UUID = UUID.fromString("00001101-0000-1000-8000-00805
        F9B34FB");
    private static String DEVICE_ADDRESS = "00:14:03:06:21:FA"; //Unique to Bluetooth Module
    private BluetoothDevice device;
    private BluetoothSocket socket;
    private OutputStream outputStream;
    private InputStream inputStream;

    EditText editText;
    TextView textView,txtArduino;
    Button startButton, sendButton,clearButton,stopButton,resetButton;

    boolean deviceConnected=false;
    Thread thread;
    byte buffer[];
    int bufferPosition;
    boolean stopThread;
    private LineChart mChart;
    private Boolean fabChecked = TRUE;

    //=====================DATABASE RELATED VARIABLES, DO NOT CHANGE
        --------------------------------
    //database variables, change here for different Apache connections:
    String server_url = "http://129.31.179.170/climateedgedb.php"; //edit the IP address
        here.
    //variables to send to the database
//    final String curr_val,tag;
    AlertDialog.Builder builder;




    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        doTheAutoRefresh();
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        // Create the adapter that will return a fragment for each of the three
        // primary sections of the activity.
        mSectionsPagerAdapter = new SectionsPagerAdapter(getSupportFragmentManager());


        // Set up the ViewPager with the sections adapter.
        mViewPager = (ViewPager) findViewById(R.id.container);
        mViewPager.setAdapter(mSectionsPagerAdapter);

        TabLayout tabLayout = (TabLayout) findViewById(R.id.tabs);
        tabLayout.setupWithViewPager(mViewPager);

        final FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
```

```

            if(fabChecked){
                if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
                    //Toast.makeText(getApplicationContext(), "Pressed Switch", Toast.
                        LENGTH_SHORT).show();
                    fab.setImageDrawable(getResources().getDrawable(R.drawable.
                        ic_stop_white_48dp, getApplicationContext().getTheme()));
                    fabChecked = FALSE;
                }
                onClickStart();

            }else {
                onClickClear();
                try {
                    onClickStop();
                } catch (IOException e) {
                    e.printStackTrace();
                }
                if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
                    Toast.makeText(getApplicationContext(), "Disconnected", Toast.
                        LENGTH_SHORT).show();
                    fab.setImageDrawable(getResources().getDrawable(R.drawable.
                        ic_play_arrow_white_48dp, getApplicationContext().getTheme()));
                    fabChecked = TRUE;
                }
            }





            try {

                FileInputStream fileInputStream = openFileInput("ArduinoData.txt");
                InputStreamReader inputStreamReader = new InputStreamReader((
                    fileInputStream));
                BufferedReader bufferedReader = new BufferedReader(inputStreamReader);
                StringBuffer stringBuffer = new StringBuffer();

                String lines;
                while ((lines = bufferedReader.readLine()) != null) {

                    stringBuffer.append(lines + "\n");

        }
        } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }


            refresher();


        }
    });


    resetButton = (Button) findViewById(R.id.resetButton);

```

```
        mChart = (LineChart) findViewById(R.id.line_chart);

    }

        private LineDataSet createSet() {

            LineDataSet set = new LineDataSet(null, "Dynamic␣Data");
            set.setAxisDependency(YAxis.AxisDependency.LEFT);
            set.setColor(ColorTemplate.getHoloBlue());
            //set.setCircleColor(Color.WHITE);
            //set.setLineWidth(2f);
            set.setDrawCircles(false);
            set.setCircleRadius(4f);
            set.setFillAlpha(65);
            set.setFillColor(ColorTemplate.getHoloBlue());


            //This sets the values being highlighted by tap gesture
            set.setHighlightEnabled(false);

            //set.setHighLightColor(Color.rgb(244, 117, 117));
            set.setValueTextColor(Color.WHITE);
            set.setValueTextSize(9f);
            set.setDrawValues(false);


            return set;
        }

    public void setUiEnabled(boolean bool) {
        startButton.setEnabled(!bool);
        sendButton.setEnabled(bool);
        stopButton.setEnabled(bool);
        txtArduino.setEnabled(bool);

    }

    public boolean BTinit()
    {
        boolean found=false;
        BluetoothAdapter bluetoothAdapter=BluetoothAdapter.getDefaultAdapter();
        if (bluetoothAdapter == null) {
            Toast.makeText(getApplicationContext(),"Device␣doesnt␣Support␣Bluetooth",Toast.
                LENGTH_SHORT).show();
        }
        if(!bluetoothAdapter.isEnabled())
        {
            Intent enableAdapter = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableAdapter, 0);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        Set<BluetoothDevice> bondedDevices = bluetoothAdapter.getBondedDevices();
        if(bondedDevices.isEmpty())
        {
            Toast.makeText(getApplicationContext(),"Please␣Pair␣the␣Device␣first",Toast.
                LENGTH_SHORT).show();
        }
```

```
        else
        {
            for (BluetoothDevice iterator : bondedDevices)
            {
                if(iterator.getAddress().equals(DEVICE_ADDRESS))
                {
                    device=iterator;
                    found=true;
                    break;
                }
            }
        }
        return found;
    }

    public boolean BTconnect()
    {
        boolean connected=true;
        try {
            socket = device.createRfcommSocketToServiceRecord(PORT_UUID);
            socket.connect();
        } catch (IOException e) {
            e.printStackTrace();
            connected=false;
        }
        if(connected)
        {
            try {
                outputStream=socket.getOutputStream();
            } catch (IOException e) {
                e.printStackTrace();
            }
            try {
                inputStream=socket.getInputStream();
            } catch (IOException e) {
                e.printStackTrace();
            }

        }


        return connected;
    }

    public void onClickStart() {
        if(BTinit())
        {
            if(BTconnect())
            {
                //setUiEnabled(true);
                deviceConnected=true;
                beginListenForData();
                //txtArduino.append("\nConnection Opened!\n");
                Toast.makeText(getApplicationContext(), "Connected",Toast.LENGTH_SHORT).show
                    ();

                onClickSend();

            }

        }
```

```
    }

    void beginListenForData()
    {
        final Handler handler = new Handler();
        stopThread = false;
        buffer = new byte[1024];
        Thread thread  = new Thread(new Runnable()
        {
            public void run()
            {
                while(!Thread.currentThread().isInterrupted() && !stopThread)
                {
                    try
                    {
                        int byteCount = inputStream.available();
                        if(byteCount > 0)
                        {
                            byte[] rawBytes = new byte[byteCount];
                            inputStream.read(rawBytes);
                            final String string=new String(rawBytes,"UTF-8");
                            handler.post(new Runnable() {
                                public void run()
                                {


                                    try {
                                        FileOutputStream fileOutputStream = openFileOutput("ArduinoData.txt", MODE_APPEND);
                                        fileOutputStream.write(string.getBytes());
                                        fileOutputStream.close();

                                    } catch (FileNotFoundException e) {
                                        e.printStackTrace();
                                    } catch (IOException e) {
                                        e.printStackTrace();
                                    }

                                }
                            });

                        }
                    }
                    catch (IOException ex)
                    {
                        stopThread = true;
                    }
                }
            }
        });

        thread.start();
    }

    public void onClickSend() {
        try {
            outputStream.write("1".getBytes());
        } catch (IOException e) {
            e.printStackTrace();
        }
            }
```

```

    public void onClickStop() throws IOException {
        stopThread = true;
        outputStream.flush();
        outputStream.close();
        inputStream.close();
        socket.close();
        deviceConnected=false;
    }

    public void onClickClear() {
        try {
            outputStream.write("0".getBytes());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }


    public void random(View view){
        Toast.makeText(getApplicationContext(), "Random Value Added",Toast.LENGTH_SHORT).
           show();
            String Mytextmessage = editText.getText().toString();
        LineData data = mChart.getData();

        if (data != null) {

            ILineDataSet set = data.getDataSetByIndex(0);

            if (set == null) {
                set = createSet();
                data.addDataSet(set);
            }

            data.addEntry(new Entry(set.getEntryCount(), (float) (Math.random() * 40) + 30f)
                , 0);
            data.notifyDataChanged();

            // let the chart know it's data has changed
            mChart.notifyDataSetChanged();

            // limit the number of visible entries
            mChart.setVisibleXRangeMaximum(120);


            // move to the latest entry
            mChart.moveViewToX(data.getEntryCount());

            // this automatically refreshes the chart (calls invalidate())

        }
    }

    public void fromArduino(String theText){
        LineData data = mChart.getData();

        if (data != null) {

            ILineDataSet set = data.getDataSetByIndex(0);

            if (set == null) {
```

```
                set = createSet();
                data.addDataSet(set);
            }

            data.addEntry(new Entry(set.getEntryCount(), (float) (Float.valueOf(theText))),
                0);
            data.notifyDataChanged();

            // let the chart know it's data has changed
            mChart.notifyDataSetChanged();

            // limit the number of visible entries
            mChart.setVisibleXRangeMaximum(120);

            // move to the latest entry
            mChart.moveViewToX(data.getEntryCount());

            // this automatically refreshes the chart (calls invalidate())

        }
    }
    private Menu menu;
    private Boolean pressed =Boolean.TRUE;



    @Override
            public boolean onCreateOptionsMenu(Menu menu) {
                // Inflate the menu; this adds items to the action bar if it is present.
                this.menu = menu;
                getMenuInflater().inflate(R.menu.menu_main, menu);
                return true;
            }

            @Override
            public boolean onOptionsItemSelected(MenuItem item) {
                // Handle action bar item clicks here. The action bar will
                // automatically handle clicks on the Home/Up button, so long
                // as you specify a parent activity in AndroidManifest.xml.
                int id = item.getItemId();

                //noinspection SimplifiableIfStatement
                if (id == R.id.action_settings) {
                    refresher();
                    return true;
                }

                if(id == R.id.myswitch){

                    Switch menuswitch = (Switch) findViewById(R.id.myswitch);

                    if(menuswitch.isChecked()){
                        Toast.makeText(getApplicationContext(), "Pressed_Switch", Toast.
                            LENGTH_LONG).show();
                    }

                    Toast.makeText(getApplicationContext(), "Pressed_Switch", Toast.
                        LENGTH_LONG).show();
                    return true;
                }

```

```java
//If the server button is pressed.
if(id == R.id.Server){
    final String curr_val,tag;
    curr_val = "val";
    tag ="tag";
    builder = new AlertDialog.Builder(MainActivity.this);

    StringRequest stringRequest = new StringRequest(Request.Method.POST, server_url,
            new Response.Listener<String>() {
                @Override
                public void onResponse(String response) {
                    builder.setTitle("Server Response");
                    builder.setMessage("Response :"+response);
                    builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {

                            System.out.println("Hello World");


                        }
                    });
                    AlertDialog alertDialog = builder.create();
                    alertDialog.show();
                }
            }
            , new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {

            Toast.makeText(MainActivity.this,"Error...",Toast.LENGTH_SHORT).show();
            error.printStackTrace();

        }
    }){ //hash map to sift through database
        @Override
        protected Map<String, String> getParams() throws AuthFailureError {
            Map<String,String> params = new HashMap<String, String>();
            params.put("tag",tag);
            params.put("value",curr_val);
            return params;
        }
    };

    MySingleton.getInstance(MainActivity.this).addTorequestque(stringRequest);
}

if ((id == R.id.onClickStart)&&pressed) {
    Toast.makeText(getApplicationContext(), "Pressed Start", Toast.LENGTH_LONG).show();
    onClickStart();
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
        menu.getItem(0).setIcon(getResources().getDrawable(R.drawable.ic_stop_white_48dp, getApplicationContext().getTheme()));
    }
    pressed = FALSE;
```

```
                return true;
            }

            if ((id == R.id.onClickStart)&!pressed) {
                Toast.makeText(getApplicationContext(), "Pressed Stop", Toast.
                    LENGTH_LONG).show();
                onClickClear();
                try {
                    onClickStop();
                } catch (IOException e) {
                    e.printStackTrace();
                }
                if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
                    menu.getItem(0).setIcon(getResources().getDrawable(R.drawable.
                        ic_play_arrow_white_48dp, getApplicationContext().getTheme()));
                }
                pressed = TRUE;
                return true;
            }

            if (id == R.id.onClickStop) {
                Toast.makeText(getApplicationContext(), "Pressed Stop", Toast.
                    LENGTH_LONG).show();
                try {
                    onClickStop();
                } catch (IOException e) {
                    e.printStackTrace();
                }
                return true;
            }

            if (id == R.id.onClickClear) {
                Toast.makeText(getApplicationContext(), "Pressed Clear", Toast.
                    LENGTH_LONG).show();
            try {
                    FileOutputStream fileOutputStream = openFileOutput("ArduinoData.txt"
                        , MODE_PRIVATE);
                    fileOutputStream.close();
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();}
                return true;
            }

            if (id == R.id.onClickSend) {
                Toast.makeText(getApplicationContext(), "Pressed Send", Toast.
                    LENGTH_LONG).show();
                onClickSend();
                return true;
            }



            return super.onOptionsItemSelected(item);
        }



        public void refresher(){
            mSectionsPagerAdapter.notifyDataSetChanged();
```

```
            }

private  final Handler handler = new Handler();

    private void doTheAutoRefresh() {
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
                // Write code for your refresh logic
                refresher();

               doTheAutoRefresh();
            }
        }, 5000);   //this is for 5mins do 3600000 for hourly
    }


           /**
             * A {@link FragmentPagerAdapter} that returns a fragment corresponding to
             * one of the sections/tabs/pages.
             */




            public class SectionsPagerAdapter extends FragmentPagerAdapter {

                public SectionsPagerAdapter(FragmentManager fm) {
                    super(fm);
                }

                @Override
                public Fragment getItem(int position) {
                    // getItem is called to instantiate the fragment for the given page.
                    // Return a PlaceholderFragment (defined as a static inner class below).
                    //return PlaceholderFragment.newInstance(position + 1);

                    switch (position) {
                        case 0:
                            Tab6 tab6 = new Tab6();
                            return tab6;
                        case 1:
                            Tab1 tab1 = new Tab1();
                            return tab1;
                        case 2:
                            Tab2 tab2 = new Tab2();
                            return tab2;
                        case 3:
                            Tab3 tab3 = new Tab3();
                            return tab3;
                        case 4:
                            Tab4 tab4 = new Tab4();
                            return tab4;
                        case 5:
                            Tab5 tab5 = new Tab5();
                            return tab5;
                        default:
                            return null;
                    }
```

```
                }

                public int getItemPosition(Object object) {
                    return POSITION_NONE;
                }

                @Override
                public int getCount() {
                    // Show 5 total pages.
                    return 6;
                }

                @Override
                public CharSequence getPageTitle(int position) {
                    switch (position) {
                        case 0:
                            return "Home";
                        case 1:
                            return getString(R.string.SECTION_1);
                        case 2:
                            return getString(R.string.SECTION_2);
                        case 3:
                            return getString(R.string.SECTION_3);
                        case 4:
                            return getString(R.string.SECTION_4);
                        case 5:
                            return getString(R.string.SECTION_5);
                    }
                    return null;
                }


            }



        }
```

B.2 Fragment 1

```
package com.example.keyan.cet;
import android.content.Context;
import android.graphics.Color;
import android.graphics.Paint;
import android.support.v4.app.Fragment;
import android.os.Bundle;
import android.support.v4.app.FragmentTransaction;
import android.support.v4.view.PagerAdapter;
import android.text.format.Time;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.github.mikephil.charting.charts.LineChart;
import com.github.mikephil.charting.components.Legend;
import com.github.mikephil.charting.components.XAxis;
import com.github.mikephil.charting.components.YAxis;
```

```
import com.github.mikephil.charting.data.Entry;
import com.github.mikephil.charting.data.LineData;
import com.github.mikephil.charting.data.LineDataSet;
import com.github.mikephil.charting.interfaces.datasets.ILineDataSet;
import com.github.mikephil.charting.utils.ColorTemplate;


import org.json.JSONException;
import org.json.JSONObject;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.text.SimpleDateFormat;
import java.util.Calendar;

import static android.content.Context.MODE_APPEND;
import static android.content.Context.MODE_PRIVATE;

/**
 * Created by Keyan on 26/05/2017.
 */

public class Tab1 extends Fragment {

    int lastpressed;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
         final View rootView = inflater.inflate(R.layout.tab1, container, false);

        final LineChart mChart = (LineChart) rootView.findViewById(R.id.line_chart);
        final TextView OneD = (TextView) rootView.findViewById(R.id.textView2);
        final TextView OneM = (TextView) rootView.findViewById(R.id.textView3);
        final TextView ThreeM = (TextView) rootView.findViewById(R.id.textView4);
        final TextView SixM = (TextView) rootView.findViewById(R.id.textView5);
        final TextView OneY = (TextView) rootView.findViewById(R.id.textView6);
        final TextView All = (TextView) rootView.findViewById(R.id.textView7);

        OneD.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                lastpressed = 1;
                one_day(OneD,OneM,ThreeM,SixM,OneY,All);
                Toast.makeText(getActivity().getApplicationContext(), "One Day", Toast.
                    LENGTH_SHORT).show();
            }
        });

        OneM.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                lastpressed = 2;
                one_month(OneD,OneM,ThreeM,SixM,OneY,All);
```

```
                Toast.makeText(getActivity().getApplicationContext(), "One Month", Toast.
                    LENGTH_SHORT).show();
            }
        });

        ThreeM.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                lastpressed = 3;
                three_month(OneD,OneM,ThreeM,SixM,OneY,All);
                Toast.makeText(getActivity().getApplicationContext(), "Three Months", Toast.
                    LENGTH_SHORT).show();
            }
        });

        SixM.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                lastpressed = 4;
                six_month(OneD,OneM,ThreeM,SixM,OneY,All);
                Toast.makeText(getActivity().getApplicationContext(), "Six Months", Toast.
                    LENGTH_SHORT).show();
            }
        });

        OneY.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                lastpressed = 5;
                one_year(OneD,OneM,ThreeM,SixM,OneY,All);
                Toast.makeText(getActivity().getApplicationContext(), "One Year", Toast.
                    LENGTH_SHORT).show();
            }
        });

        All.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                lastpressed = 6;
                all_time(OneD,OneM,ThreeM,SixM,OneY,All);
                Toast.makeText(getActivity().getApplicationContext(), "All", Toast.
                    LENGTH_SHORT).show();
            }
        });

        switch (lastpressed) {
            case 1:  one_day(OneD,OneM,ThreeM,SixM,OneY,All);
                break;
            case 2:  one_month(OneD,OneM,ThreeM,SixM,OneY,All);
                break;
            case 3:  three_month(OneD,OneM,ThreeM,SixM,OneY,All);
                break;
            case 4:  six_month(OneD,OneM,ThreeM,SixM,OneY,All);
                break;
            case 5:  one_year(OneD,OneM,ThreeM,SixM,OneY,All);
                break;
            case 6:  all_time(OneD,OneM,ThreeM,SixM,OneY,All);
                break;
            default:
                break;
        }
```

```java


        mChart.getDescription().setEnabled(false);

        // enable touch gestures
        mChart.setTouchEnabled(true);

        // enable scaling and dragging
        mChart.setDragEnabled(true);
        mChart.setScaleEnabled(true);
        mChart.setDrawGridBackground(false);

        // if disabled, scaling can be done on x- and y-axis separately
        mChart.setPinchZoom(true);

        // set an alternative background color
        mChart.setBackgroundColor(Color.WHITE);

        final LineData data = new LineData();
        data.setValueTextColor(Color.WHITE);

        // add empty data

        mChart.setData(data);

        // get the legend (only possible after setting data)
        Legend l = mChart.getLegend();

        // modify the legend ...
        l.setForm(Legend.LegendForm.LINE);
        //l.setTypeface(mTfLight);
        l.setTextColor(Color.BLUE);
        l.setEnabled(false);



        XAxis xl = mChart.getXAxis();
        xl.setPosition(XAxis.XAxisPosition.BOTTOM);
        xl.setTextColor(Color.BLUE);
        xl.setDrawGridLines(false);
        xl.setAvoidFirstLastClipping(true);
        xl.setEnabled(true);




        YAxis leftAxis = mChart.getAxisLeft();
        leftAxis.setTextColor(Color.BLUE);
        leftAxis.setAxisMinimum(0f);
        leftAxis.setDrawGridLines(true);

        YAxis rightAxis = mChart.getAxisRight();
        rightAxis.setEnabled(false);



        try {

            FileInputStream fileInputStream = getActivity().openFileInput("ArduinoData.txt")
                ;
            InputStreamReader inputStreamReader = new InputStreamReader((fileInputStream));
```

```
            BufferedReader bufferedReader = new BufferedReader(inputStreamReader);
            StringBuffer stringBuffer = new StringBuffer();

            String lines;
            while ((lines = bufferedReader.readLine()) != null) {
                if (lines.charAt(0)=='R'){
                    stringBuffer.append(lines.substring(1,lines.length()) + "\n");


                if (data != null) {

                    ILineDataSet set = data.getDataSetByIndex(0);


                    if (set == null) {
                        set = createSet();
                        data.addDataSet(set);
                    }

                    Float RandFloat = Float.parseFloat(lines.substring(1,lines.length()));


                    data.addEntry(new Entry(set.getEntryCount(), (float) (RandFloat)), 0);

                    data.notifyDataChanged();

                    // let the chart know it's data has changed
                    mChart.notifyDataSetChanged();

                    // limit the number of visible entries
                    mChart.setVisibleXRangeMaximum(24);
                    mChart.setVisibleXRange(0,24);

                    // move to the latest entry
                    mChart.moveViewToX(data.getEntryCount());
                }}

            }
            //textBox.setText(stringBuffer);
            fileInputStream.close();


        } catch (FileNotFoundException e) {
            e.printStackTrace();


        } catch (IOException e) {
            e.printStackTrace();

        }

            return rootView;
    }



    private LineDataSet createSet() {

        LineDataSet set = new LineDataSet(null, "");
        set.setAxisDependency(YAxis.AxisDependency.LEFT);
        set.setColor(ColorTemplate.getHoloBlue());
```

```
        set.setDrawCircles(false);
        set.setCircleRadius(4f);
        set.setFillAlpha(65);
        set.setFillColor(ColorTemplate.getHoloBlue());
        set.setDrawFilled(true);

        //This sets the values being highlighted by tap gesture
        set.setHighlightEnabled(false);

        set.setValueTextColor(Color.WHITE);
        set.setValueTextSize(9f);
        set.setDrawValues(false);


        return set;
    }

    public void one_day(TextView OneD,TextView OneM,TextView ThreeM,TextView SixM,TextView OneY,TextView All) {

        OneD.setTextColor(Color.parseColor("#FF4081"));
        OneD.setPaintFlags(OneD.getPaintFlags() |   Paint.UNDERLINE_TEXT_FLAG);
        OneM.setTextColor(Color.parseColor("#808080"));
        OneM.setPaintFlags(View.INVISIBLE);
        ThreeM.setTextColor(Color.parseColor("#808080"));
        ThreeM.setPaintFlags(View.INVISIBLE);
        SixM.setTextColor(Color.parseColor("#808080"));
        SixM.setPaintFlags(View.INVISIBLE);
        OneY.setTextColor(Color.parseColor("#808080"));
        OneY.setPaintFlags(View.INVISIBLE);
        All.setTextColor(Color.parseColor("#808080"));
        All.setPaintFlags(View.INVISIBLE);
    }

    public void one_month(TextView OneD,TextView OneM,TextView ThreeM,TextView SixM,TextView OneY,TextView All){
        OneM.setTextColor(Color.parseColor("#FF4081"));
        OneM.setPaintFlags(OneD.getPaintFlags() |   Paint.UNDERLINE_TEXT_FLAG);
        OneD.setTextColor(Color.parseColor("#808080"));
        OneD.setPaintFlags(View.INVISIBLE);
        ThreeM.setTextColor(Color.parseColor("#808080"));
        ThreeM.setPaintFlags(View.INVISIBLE);
        SixM.setTextColor(Color.parseColor("#808080"));
        SixM.setPaintFlags(View.INVISIBLE);
        OneY.setTextColor(Color.parseColor("#808080"));
        OneY.setPaintFlags(View.INVISIBLE);
        All.setTextColor(Color.parseColor("#808080"));
        All.setPaintFlags(View.INVISIBLE);
    }

    public void three_month(TextView OneD,TextView OneM,TextView ThreeM,TextView SixM,TextView OneY,TextView All){
        ThreeM.setTextColor(Color.parseColor("#FF4081"));
        ThreeM.setPaintFlags(OneD.getPaintFlags() |   Paint.UNDERLINE_TEXT_FLAG);
        OneD.setTextColor(Color.parseColor("#808080"));
        OneD.setPaintFlags(View.INVISIBLE);
        OneM.setTextColor(Color.parseColor("#808080"));
        OneM.setPaintFlags(View.INVISIBLE);
        SixM.setTextColor(Color.parseColor("#808080"));
        SixM.setPaintFlags(View.INVISIBLE);
        OneY.setTextColor(Color.parseColor("#808080"));
```

```
        OneY.setPaintFlags(View.INVISIBLE);
        All.setTextColor(Color.parseColor("#808080"));
        All.setPaintFlags(View.INVISIBLE);
    }

    public void six_month(TextView OneD,TextView OneM,TextView ThreeM,TextView SixM,TextView
         OneY,TextView All){
        SixM.setTextColor(Color.parseColor("#FF4081"));
        SixM.setPaintFlags(OneD.getPaintFlags() |   Paint.UNDERLINE_TEXT_FLAG);
        OneD.setTextColor(Color.parseColor("#808080"));
        OneD.setPaintFlags(View.INVISIBLE);
        ThreeM.setTextColor(Color.parseColor("#808080"));
        ThreeM.setPaintFlags(View.INVISIBLE);
        OneM.setTextColor(Color.parseColor("#808080"));
        OneM.setPaintFlags(View.INVISIBLE);
        OneY.setTextColor(Color.parseColor("#808080"));
        OneY.setPaintFlags(View.INVISIBLE);
        All.setTextColor(Color.parseColor("#808080"));
        All.setPaintFlags(View.INVISIBLE);
    }

    public void one_year(TextView OneD,TextView OneM,TextView ThreeM,TextView SixM,TextView
        OneY,TextView All){
        OneY.setTextColor(Color.parseColor("#FF4081"));
        OneY.setPaintFlags(OneD.getPaintFlags() |   Paint.UNDERLINE_TEXT_FLAG);
        OneD.setTextColor(Color.parseColor("#808080"));
        OneD.setPaintFlags(View.INVISIBLE);
        ThreeM.setTextColor(Color.parseColor("#808080"));
        ThreeM.setPaintFlags(View.INVISIBLE);
        SixM.setTextColor(Color.parseColor("#808080"));
        SixM.setPaintFlags(View.INVISIBLE);
        OneM.setTextColor(Color.parseColor("#808080"));
        OneM.setPaintFlags(View.INVISIBLE);
        All.setTextColor(Color.parseColor("#808080"));
        All.setPaintFlags(View.INVISIBLE);
    }

    public void all_time(TextView OneD,TextView OneM,TextView ThreeM,TextView SixM,TextView
        OneY,TextView All){
        All.setTextColor(Color.parseColor("#FF4081"));
        All.setPaintFlags(OneD.getPaintFlags() |   Paint.UNDERLINE_TEXT_FLAG);
        OneD.setTextColor(Color.parseColor("#808080"));
        OneD.setPaintFlags(View.INVISIBLE);
        ThreeM.setTextColor(Color.parseColor("#808080"));
        ThreeM.setPaintFlags(View.INVISIBLE);
        SixM.setTextColor(Color.parseColor("#808080"));
        SixM.setPaintFlags(View.INVISIBLE);
        OneY.setTextColor(Color.parseColor("#808080"));
        OneY.setPaintFlags(View.INVISIBLE);
        OneM.setTextColor(Color.parseColor("#808080"));
        OneM.setPaintFlags(View.INVISIBLE);
    }




}
```

B.3 Fragment 2

```java
package com.example.keyan.cet;
import android.content.Context;
import android.graphics.Color;
import android.support.v4.app.Fragment;
import android.os.Bundle;
import android.support.v4.app.FragmentTransaction;
import android.support.v4.view.PagerAdapter;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.github.mikephil.charting.charts.LineChart;
import com.github.mikephil.charting.components.Legend;
import com.github.mikephil.charting.components.XAxis;
import com.github.mikephil.charting.components.YAxis;
import com.github.mikephil.charting.data.Entry;
import com.github.mikephil.charting.data.LineData;
import com.github.mikephil.charting.data.LineDataSet;
import com.github.mikephil.charting.interfaces.datasets.ILineDataSet;
import com.github.mikephil.charting.utils.ColorTemplate;


import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;


import static android.content.Context.MODE_APPEND;
import static android.content.Context.MODE_PRIVATE;
import static com.example.keyan.cet.R.id.bottom;
import static com.example.keyan.cet.R.id.resetButton;


/**
 * Created by Keyan on 26/05/2017.
 */

public class Tab2 extends Fragment {


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        final View rootView = inflater.inflate(R.layout.tab2, container, false);

        Button randButton = (Button) rootView.findViewById(R.id.randButton);
        Button resetButton = (Button) rootView.findViewById(R.id.resetButton);
        TextView textBox =  (TextView) rootView.findViewById(R.id.textBox);
        final LineChart mChart = (LineChart) rootView.findViewById(R.id.line_chart);

        mChart.getDescription().setEnabled(true);
        mChart.getDescription().setText("Rainfall");
```

```

        // enable touch gestures
        mChart.setTouchEnabled(true);

        // enable scaling and dragging
        mChart.setDragEnabled(true);
        mChart.setScaleEnabled(true);
        mChart.setDrawGridBackground(false);

        // if disabled, scaling can be done on x- and y-axis separately
        mChart.setPinchZoom(true);

        // set an alternative background color
        mChart.setBackgroundColor(Color.WHITE);

        final LineData data = new LineData();
        data.setValueTextColor(Color.WHITE);

        // add empty data

        mChart.setData(data);

        // get the legend (only possible after setting data)
        Legend l = mChart.getLegend();

        // modify the legend ...
        l.setForm(Legend.LegendForm.LINE);
        l.setTextColor(Color.BLUE);



        XAxis xl = mChart.getXAxis();
        xl.setTextColor(Color.BLUE);
        xl.setDrawGridLines(false);
        xl.setAvoidFirstLastClipping(true);
        xl.setEnabled(true);



        YAxis leftAxis = mChart.getAxisLeft();
        leftAxis.setTextColor(Color.BLUE);
        leftAxis.setAxisMinimum(0f);
        leftAxis.setDrawGridLines(true);

        YAxis rightAxis = mChart.getAxisRight();
        rightAxis.setEnabled(false);

        try {

            FileInputStream fileInputStream = getActivity().openFileInput("ArduinoData.txt")
                ;
            InputStreamReader inputStreamReader = new InputStreamReader((fileInputStream));
            BufferedReader bufferedReader = new BufferedReader(inputStreamReader);
            StringBuffer stringBuffer = new StringBuffer();

            String lines;
            while ((lines = bufferedReader.readLine()) != null) {
                if (lines.charAt(0)=='A'){
                    stringBuffer.append(lines.substring(1,lines.length()) + "\n");


```

```
                    if (data != null) {

                        ILineDataSet set = data.getDataSetByIndex(0);

                        if (set == null) {
                            set = createSet();
                            data.addDataSet(set);

                        }

                        Float RandFloat = Float.parseFloat(lines.substring(1,lines.length())
                            );

                        data.addEntry(new Entry(set.getEntryCount(), (float) (RandFloat)),
                            0);
                        data.notifyDataChanged();

                        // let the chart know it's data has changed
                        mChart.notifyDataSetChanged();

                        // limit the number of visible entries
                        mChart.setVisibleXRangeMaximum(120);

                        // move to the latest entry
                        mChart.moveViewToX(data.getEntryCount());
                    }}

            }
            textBox.setText(stringBuffer);
            fileInputStream.close();


        } catch (FileNotFoundException e) {
            e.printStackTrace();


        } catch (IOException e) {
            e.printStackTrace();

        }


        randButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                // enable description text



                if (data != null) {

                    ILineDataSet set = data.getDataSetByIndex(0);

                    if (set == null) {
                        set = createSet();
                        data.addDataSet(set);

                    }

                    Float RandFloat = (float) Math.random() * 40;
```

```java

                    data.addEntry(new Entry(set.getEntryCount(), (float) (RandFloat)), 0);
                    data.notifyDataChanged();

                    // let the chart know it's data has changed
                    mChart.notifyDataSetChanged();

                    // limit the number of visible entries
                    mChart.setVisibleXRangeMaximum(120);

                    // move to the latest entry
                    mChart.moveViewToX(data.getEntryCount());


                    try {
                        FileOutputStream fileOutputStream = getActivity().openFileOutput("
                            textBoxNew2.txt", MODE_APPEND);
                        fileOutputStream.write(RandFloat.toString().getBytes());
                        fileOutputStream.write("\n".getBytes());
                        fileOutputStream.close();
                    } catch (FileNotFoundException e) {
                        e.printStackTrace();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }

                }

            }

        });

        resetButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {


                try {
                    FileOutputStream fileOutputStream = getActivity().openFileOutput("
                        textBoxNew2.txt", MODE_PRIVATE);
                    fileOutputStream.close();
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                }

                Toast.makeText(getActivity().getApplicationContext(), "Reset, switch to 
                    section 3 or beyond and back to refresh graph", Toast.LENGTH_SHORT).show
                    ();




            }

        });


        return rootView;
    }
```

```



    private LineDataSet createSet() {

        LineDataSet set = new LineDataSet(null, "Dynamic Data");
        set.setAxisDependency(YAxis.AxisDependency.LEFT);
        set.setColor(ColorTemplate.getHoloBlue());
        set.setDrawCircles(false);
        set.setCircleRadius(4f);
        set.setFillAlpha(65);
        set.setFillColor(ColorTemplate.getHoloBlue());
        set.setDrawFilled(true);

        //This sets the values being highlighted by tap gesture
        set.setHighlightEnabled(false);

        set.setValueTextColor(Color.WHITE);
        set.setValueTextSize(9f);
        set.setDrawValues(false);


        return set;
    }







}
```

B.4 Fragment 3

```
package com.example.keyan.cet;
import android.graphics.Color;
import android.graphics.Paint;
import android.support.v4.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import org.w3c.dom.Text;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import static android.content.Context.MODE_PRIVATE;
import static com.example.keyan.cet.R.color.colorPrimary;


/**
 * Created by Keyan on 26/05/2017.
 */

```

```java
public class Tab3 extends Fragment {

    int lastpressed;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.tab3, container, false);

        final TextView OneD = (TextView) rootView.findViewById(R.id.textView2);
        final TextView OneM = (TextView) rootView.findViewById(R.id.textView3);
        final TextView ThreeM = (TextView) rootView.findViewById(R.id.textView4);
        final TextView SixM = (TextView) rootView.findViewById(R.id.textView5);
        final TextView OneY = (TextView) rootView.findViewById(R.id.textView6);
        final TextView All = (TextView) rootView.findViewById(R.id.textView7);

        OneD.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                lastpressed = 1;
                one_day(OneD,OneM,ThreeM,SixM,OneY,All);
                Toast.makeText(getActivity().getApplicationContext(), "One Day", Toast.LENGTH_SHORT).show();
            }
        });

        OneM.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                lastpressed = 2;
                one_month(OneD,OneM,ThreeM,SixM,OneY,All);
                Toast.makeText(getActivity().getApplicationContext(), "One Month", Toast.LENGTH_SHORT).show();
            }
        });

        ThreeM.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                lastpressed = 3;
                three_month(OneD,OneM,ThreeM,SixM,OneY,All);
                Toast.makeText(getActivity().getApplicationContext(), "Three Months", Toast.LENGTH_SHORT).show();
            }
        });

        SixM.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                lastpressed = 4;
                six_month(OneD,OneM,ThreeM,SixM,OneY,All);
                Toast.makeText(getActivity().getApplicationContext(), "Six Months", Toast.LENGTH_SHORT).show();
            }
        });

        OneY.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                lastpressed = 5;
                one_year(OneD,OneM,ThreeM,SixM,OneY,All);
```

```
                Toast.makeText(getActivity().getApplicationContext(), "One Year", Toast.LENGTH_SHORT).show();
            }
        });

        All.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                lastpressed = 6;
                all_time(OneD,OneM,ThreeM,SixM,OneY,All);
                Toast.makeText(getActivity().getApplicationContext(), "All", Toast.LENGTH_SHORT).show();
            }
        });

        switch (lastpressed) {
            case 1:  one_day(OneD,OneM,ThreeM,SixM,OneY,All);
                break;
            case 2:  one_month(OneD,OneM,ThreeM,SixM,OneY,All);
                break;
            case 3:  three_month(OneD,OneM,ThreeM,SixM,OneY,All);
                break;
            case 4:  six_month(OneD,OneM,ThreeM,SixM,OneY,All);
                break;
            case 5:  one_year(OneD,OneM,ThreeM,SixM,OneY,All);
                break;
            case 6:  all_time(OneD,OneM,ThreeM,SixM,OneY,All);
                break;
            default:
                break;
        }


        return rootView;
    }

    public void one_day(TextView OneD,TextView OneM,TextView ThreeM,TextView SixM,TextView OneY,TextView All) {

        OneD.setTextColor(Color.parseColor("#FF4081"));
        OneD.setPaintFlags(OneD.getPaintFlags() |   Paint.UNDERLINE_TEXT_FLAG);
        OneM.setTextColor(Color.parseColor("#808080"));
        OneM.setPaintFlags(View.INVISIBLE);
        ThreeM.setTextColor(Color.parseColor("#808080"));
        ThreeM.setPaintFlags(View.INVISIBLE);
        SixM.setTextColor(Color.parseColor("#808080"));
        SixM.setPaintFlags(View.INVISIBLE);
        OneY.setTextColor(Color.parseColor("#808080"));
        OneY.setPaintFlags(View.INVISIBLE);
        All.setTextColor(Color.parseColor("#808080"));
        All.setPaintFlags(View.INVISIBLE);
    }

    public void one_month(TextView OneD,TextView OneM,TextView ThreeM,TextView SixM,TextView OneY,TextView All){
        OneM.setTextColor(Color.parseColor("#FF4081"));
        OneM.setPaintFlags(OneD.getPaintFlags() |   Paint.UNDERLINE_TEXT_FLAG);
        OneD.setTextColor(Color.parseColor("#808080"));
        OneD.setPaintFlags(View.INVISIBLE);
        ThreeM.setTextColor(Color.parseColor("#808080"));
        ThreeM.setPaintFlags(View.INVISIBLE);
```

```
        SixM.setTextColor(Color.parseColor("#808080"));
        SixM.setPaintFlags(View.INVISIBLE);
        OneY.setTextColor(Color.parseColor("#808080"));
        OneY.setPaintFlags(View.INVISIBLE);
        All.setTextColor(Color.parseColor("#808080"));
        All.setPaintFlags(View.INVISIBLE);
    }

    public void three_month(TextView OneD,TextView OneM,TextView ThreeM,TextView SixM,
       TextView OneY,TextView All){
        ThreeM.setTextColor(Color.parseColor("#FF4081"));
        ThreeM.setPaintFlags(OneD.getPaintFlags() |   Paint.UNDERLINE_TEXT_FLAG);
        OneD.setTextColor(Color.parseColor("#808080"));
        OneD.setPaintFlags(View.INVISIBLE);
        OneM.setTextColor(Color.parseColor("#808080"));
        OneM.setPaintFlags(View.INVISIBLE);
        SixM.setTextColor(Color.parseColor("#808080"));
        SixM.setPaintFlags(View.INVISIBLE);
        OneY.setTextColor(Color.parseColor("#808080"));
        OneY.setPaintFlags(View.INVISIBLE);
        All.setTextColor(Color.parseColor("#808080"));
        All.setPaintFlags(View.INVISIBLE);
    }

    public void six_month(TextView OneD,TextView OneM,TextView ThreeM,TextView SixM,TextView
        OneY,TextView All){
        SixM.setTextColor(Color.parseColor("#FF4081"));
        SixM.setPaintFlags(OneD.getPaintFlags() |   Paint.UNDERLINE_TEXT_FLAG);
        OneD.setTextColor(Color.parseColor("#808080"));
        OneD.setPaintFlags(View.INVISIBLE);
        ThreeM.setTextColor(Color.parseColor("#808080"));
        ThreeM.setPaintFlags(View.INVISIBLE);
        OneM.setTextColor(Color.parseColor("#808080"));
        OneM.setPaintFlags(View.INVISIBLE);
        OneY.setTextColor(Color.parseColor("#808080"));
        OneY.setPaintFlags(View.INVISIBLE);
        All.setTextColor(Color.parseColor("#808080"));
        All.setPaintFlags(View.INVISIBLE);
    }

    public void one_year(TextView OneD,TextView OneM,TextView ThreeM,TextView SixM,TextView
       OneY,TextView All){
        OneY.setTextColor(Color.parseColor("#FF4081"));
        OneY.setPaintFlags(OneD.getPaintFlags() |   Paint.UNDERLINE_TEXT_FLAG);
        OneD.setTextColor(Color.parseColor("#808080"));
        OneD.setPaintFlags(View.INVISIBLE);
        ThreeM.setTextColor(Color.parseColor("#808080"));
        ThreeM.setPaintFlags(View.INVISIBLE);
        SixM.setTextColor(Color.parseColor("#808080"));
        SixM.setPaintFlags(View.INVISIBLE);
        OneM.setTextColor(Color.parseColor("#808080"));
        OneM.setPaintFlags(View.INVISIBLE);
        All.setTextColor(Color.parseColor("#808080"));
        All.setPaintFlags(View.INVISIBLE);
    }

    public void all_time(TextView OneD,TextView OneM,TextView ThreeM,TextView SixM,TextView
       OneY,TextView All){
        All.setTextColor(Color.parseColor("#FF4081"));
        All.setPaintFlags(OneD.getPaintFlags() |   Paint.UNDERLINE_TEXT_FLAG);
        OneD.setTextColor(Color.parseColor("#808080"));
```

```
        OneD.setPaintFlags(View.INVISIBLE);
        ThreeM.setTextColor(Color.parseColor("#808080"));
        ThreeM.setPaintFlags(View.INVISIBLE);
        SixM.setTextColor(Color.parseColor("#808080"));
        SixM.setPaintFlags(View.INVISIBLE);
        OneY.setTextColor(Color.parseColor("#808080"));
        OneY.setPaintFlags(View.INVISIBLE);
        OneM.setTextColor(Color.parseColor("#808080"));
        OneM.setPaintFlags(View.INVISIBLE);
    }
}
```

B.5 Fragment 4

```
package com.example.keyan.cet;
import android.content.Context;
import android.graphics.Color;
import android.support.v4.app.Fragment;
import android.os.Bundle;
import android.support.v4.app.FragmentTransaction;
import android.support.v4.view.PagerAdapter;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.github.mikephil.charting.charts.LineChart;
import com.github.mikephil.charting.components.Legend;
import com.github.mikephil.charting.components.XAxis;
import com.github.mikephil.charting.components.YAxis;
import com.github.mikephil.charting.data.Entry;
import com.github.mikephil.charting.data.LineData;
import com.github.mikephil.charting.data.LineDataSet;
import com.github.mikephil.charting.interfaces.datasets.ILineDataSet;
import com.github.mikephil.charting.utils.ColorTemplate;


import org.json.JSONException;
import org.json.JSONObject;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;


import static android.content.Context.MODE_APPEND;
import static android.content.Context.MODE_PRIVATE;


/**
 * Created by Keyan on 26/05/2017.
 */

```

```
public class Tab4 extends Fragment {


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        final View rootView = inflater.inflate(R.layout.tab4, container, false);

        Button randButton = (Button) rootView.findViewById(R.id.randButton);
        Button resetButton = (Button) rootView.findViewById(R.id.resetButton);
        Button refreshButton = (Button) rootView.findViewById(R.id.refreshButton);
        final TextView textBox =  (TextView) rootView.findViewById(R.id.textBox);
        final LineChart mChart = (LineChart) rootView.findViewById(R.id.line_chart);

        mChart.getDescription().setEnabled(true);
        mChart.getDescription().setText("Rainfall");

        // enable touch gestures
        mChart.setTouchEnabled(true);

        // enable scaling and dragging
        mChart.setDragEnabled(true);
        mChart.setScaleEnabled(true);
        mChart.setDrawGridBackground(false);

        // if disabled, scaling can be done on x- and y-axis separately
        mChart.setPinchZoom(true);

        // set an alternative background color
        mChart.setBackgroundColor(Color.WHITE);

        final LineData data = new LineData();
        data.setValueTextColor(Color.WHITE);

        // add empty data

        mChart.setData(data);

        // get the legend (only possible after setting data)
        Legend l = mChart.getLegend();

        // modify the legend ...
        l.setForm(Legend.LegendForm.LINE);
        l.setTextColor(Color.BLUE);



        XAxis xl = mChart.getXAxis();
        xl.setTextColor(Color.BLUE);
        xl.setDrawGridLines(false);
        xl.setAvoidFirstLastClipping(true);
        xl.setEnabled(true);



        YAxis leftAxis = mChart.getAxisLeft();
        leftAxis.setTextColor(Color.BLUE);
                leftAxis.setAxisMinimum(0f);
        leftAxis.setDrawGridLines(true);

        YAxis rightAxis = mChart.getAxisRight();
```

```
        rightAxis.setEnabled(false);


        try {

            FileInputStream fileInputStream = getActivity().openFileInput("ArduinoData.txt");
            InputStreamReader inputStreamReader = new InputStreamReader((fileInputStream));
            BufferedReader bufferedReader = new BufferedReader(inputStreamReader);
            StringBuffer stringBuffer = new StringBuffer();

            String lines;
            while ((lines = bufferedReader.readLine()) != null) {
                if (lines.charAt(0)=='S'){
                    stringBuffer.append(lines.substring(1,lines.length()) + "\n");


                    if (data != null) {

                        ILineDataSet set = data.getDataSetByIndex(0);

                        if (set == null) {
                            set = createSet();
                            data.addDataSet(set);

                        }

                        Float RandFloat = Float.parseFloat(lines.substring(1,lines.length()));

                        data.addEntry(new Entry(set.getEntryCount(), (float) (RandFloat)), 0);
                        data.notifyDataChanged();

                        // let the chart know it's data has changed
                        mChart.notifyDataSetChanged();

                        // limit the number of visible entries
                        mChart.setVisibleXRangeMaximum(120);

                        // move to the latest entry
                        mChart.moveViewToX(data.getEntryCount());
                    }}

            }
            textBox.setText(stringBuffer);
            fileInputStream.close();


        } catch (FileNotFoundException e) {
            e.printStackTrace();


        } catch (IOException e) {
            e.printStackTrace();

        }


        refreshButton.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View v) {

                // enable description text

                try {

                    FileInputStream fileInputStream = getActivity().openFileInput("Data1.txt
                        ");
                    InputStreamReader inputStreamReader = new InputStreamReader((
                        fileInputStream));
                    BufferedReader bufferedReader = new BufferedReader(inputStreamReader);
                    StringBuffer stringBuffer = new StringBuffer();


                    String lines;
                    while ((lines = bufferedReader.readLine()) != null) {

                        stringBuffer.append(lines + "\n");
                        if (data != null) {

                            ILineDataSet set = data.getDataSetByIndex(0);

                            if (set == null) {
                                set = createSet();
                                data.addDataSet(set);

                            }

                            Float RandFloat = Float.parseFloat(lines);

                            data.addEntry(new Entry(set.getEntryCount(), (float) (RandFloat)
                                ), 0);
                            data.notifyDataChanged();

                            // let the chart know it's data has changed
                            mChart.notifyDataSetChanged();

                                                                                }


                    }
                    textBox.setText(stringBuffer);

                    fileInputStream.close();


                } catch (FileNotFoundException e) {
                    e.printStackTrace();


                } catch (IOException e) {
                    e.printStackTrace();

                }

            }

        });
```

```java

        randButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                // enable description text



                if (data != null) {

                    ILineDataSet set = data.getDataSetByIndex(0);

                    if (set == null) {
                        set = createSet();
                        data.addDataSet(set);

                    }

                    Float RandFloat = (float) Math.random() * 40;

                    data.addEntry(new Entry(set.getEntryCount(), (float) (RandFloat)), 0);
                    data.notifyDataChanged();

                    // let the chart know it's data has changed
                    mChart.notifyDataSetChanged();

                    // limit the number of visible entries
                    mChart.setVisibleXRangeMaximum(120);
                    // mChart.setVisibleYRange(30, AxisDependency.LEFT);

                    // move to the latest entry
                    mChart.moveViewToX(data.getEntryCount());


                    try {
                        FileOutputStream fileOutputStream = getActivity().openFileOutput("
                            ArduinoData.txt", MODE_APPEND);
                        fileOutputStream.write(RandFloat.toString().getBytes());
                        fileOutputStream.write("\n".getBytes());
                        fileOutputStream.close();
                    } catch (FileNotFoundException e) {
                        e.printStackTrace();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }

                }

            }

        });

        resetButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {


                try {
                    FileOutputStream fileOutputStream = getActivity().openFileOutput("
                        ArduinoData.txt", MODE_PRIVATE);
```

```
                    //fileOutputStream.write("".getBytes());
                    fileOutputStream.close();
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                }

                Toast.makeText(getActivity().getApplicationContext(), "Reset,␣switch␣to␣section␣3␣or␣beyond␣and␣back␣to␣refresh␣graph", Toast.LENGTH_SHORT).show();




            }

        });


        return rootView;
    }



    private LineDataSet createSet() {

        LineDataSet set = new LineDataSet(null, "Dynamic␣Data");
        set.setAxisDependency(YAxis.AxisDependency.LEFT);
        set.setColor(ColorTemplate.getHoloBlue());
        set.setDrawCircles(false);
        set.setCircleRadius(4f);
        set.setFillAlpha(65);
        set.setFillColor(ColorTemplate.getHoloBlue());
        set.setDrawFilled(true);

        //This sets the values being highlighted by tap gesture
        set.setHighlightEnabled(false);

        set.setValueTextColor(Color.WHITE);
        set.setValueTextSize(9f);
        set.setDrawValues(false);


        return set;
    }







}
```

B.6 Fragment 5

```
package com.example.keyan.cet;
import android.content.Context;
import android.graphics.Color;
import android.support.v4.app.Fragment;
```

```
import android.os.Bundle;
import android.support.v4.app.FragmentTransaction;
import android.support.v4.view.PagerAdapter;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.github.mikephil.charting.charts.LineChart;
import com.github.mikephil.charting.components.Legend;
import com.github.mikephil.charting.components.XAxis;
import com.github.mikephil.charting.components.YAxis;
import com.github.mikephil.charting.data.Entry;
import com.github.mikephil.charting.data.LineData;
import com.github.mikephil.charting.data.LineDataSet;
import com.github.mikephil.charting.interfaces.datasets.ILineDataSet;
import com.github.mikephil.charting.utils.ColorTemplate;


import org.json.JSONException;
import org.json.JSONObject;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;


import static android.content.Context.MODE_APPEND;
import static android.content.Context.MODE_PRIVATE;


/**
 * Created by Keyan on 26/05/2017.
 */

public class Tab5 extends Fragment {


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        final View rootView = inflater.inflate(R.layout.tab5, container, false);

        Button randButton = (Button) rootView.findViewById(R.id.randButton);
        Button resetButton = (Button) rootView.findViewById(R.id.resetButton);
        Button refreshButton = (Button) rootView.findViewById(R.id.refreshButton);
        final TextView textBox =  (TextView) rootView.findViewById(R.id.textBox);
        final LineChart mChart = (LineChart) rootView.findViewById(R.id.line_chart);

        mChart.getDescription().setEnabled(true);
        mChart.getDescription().setText("Rainfall");

        // enable touch gestures
        mChart.setTouchEnabled(true);
```

```

        // enable scaling and dragging
        mChart.setDragEnabled(true);
        mChart.setScaleEnabled(true);
        mChart.setDrawGridBackground(false);

        // if disabled, scaling can be done on x- and y-axis separately
        mChart.setPinchZoom(true);

        // set an alternative background color
        mChart.setBackgroundColor(Color.WHITE);

        final LineData data = new LineData();
        data.setValueTextColor(Color.WHITE);

        // add empty data

        mChart.setData(data);

        // get the legend (only possible after setting data)
        Legend l = mChart.getLegend();

        // modify the legend ...
        l.setForm(Legend.LegendForm.LINE);
        //l.setTypeface(mTfLight);
        l.setTextColor(Color.BLUE);



        XAxis xl = mChart.getXAxis();
        xl.setTextColor(Color.BLUE);
        xl.setDrawGridLines(false);
        xl.setAvoidFirstLastClipping(true);
        xl.setEnabled(true);



        YAxis leftAxis = mChart.getAxisLeft();
        leftAxis.setTextColor(Color.BLUE);
        leftAxis.setAxisMinimum(0f);
        leftAxis.setDrawGridLines(true);

        YAxis rightAxis = mChart.getAxisRight();
        rightAxis.setEnabled(false);


        try {

            FileInputStream fileInputStream = getActivity().openFileInput("ArduinoData.txt")
                ;
            InputStreamReader inputStreamReader = new InputStreamReader((fileInputStream));
            BufferedReader bufferedReader = new BufferedReader(inputStreamReader);
            StringBuffer stringBuffer = new StringBuffer();

            String lines;
            while ((lines = bufferedReader.readLine()) != null) {
                if (lines.charAt(0)=='H'){
                    stringBuffer.append(lines.substring(1,lines.length()) + "\n");


                    if (data != null) {
```

```

                        ILineDataSet set = data.getDataSetByIndex(0);

                        if (set == null) {
                            set = createSet();
                            data.addDataSet(set);

                        }

                        Float RandFloat = Float.parseFloat(lines.substring(1,lines.length()));

                        data.addEntry(new Entry(set.getEntryCount(), (float) (RandFloat)), 0);
                        data.notifyDataChanged();

                        // let the chart know it's data has changed
                        mChart.notifyDataSetChanged();

                        // limit the number of visible entries
                        mChart.setVisibleXRangeMaximum(100);

                        // move to the latest entry
                        mChart.moveViewToX(data.getEntryCount());
                    }}

            }
            textBox.setText(stringBuffer);
            fileInputStream.close();


        } catch (FileNotFoundException e) {
            e.printStackTrace();


        } catch (IOException e) {
            e.printStackTrace();

        }


            refreshButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                // enable description text

                try {

                    FileInputStream fileInputStream = getActivity().openFileInput("Data1.txt");
                    InputStreamReader inputStreamReader = new InputStreamReader((fileInputStream));
                    BufferedReader bufferedReader = new BufferedReader(inputStreamReader);
                    StringBuffer stringBuffer = new StringBuffer();


                    String lines;
                    while ((lines = bufferedReader.readLine()) != null) {

                        stringBuffer.append(lines + "\n");
```

```
                    if (data != null) {

                        ILineDataSet set = data.getDataSetByIndex(0);

                        if (set == null) {
                            set = createSet();
                            data.addDataSet(set);

                        }

                        Float RandFloat = Float.parseFloat(lines);

                        data.addEntry(new Entry(set.getEntryCount(), (float) (RandFloat)
                            ), 0);
                        data.notifyDataChanged();

                        // let the chart know it's data has changed
                        mChart.notifyDataSetChanged();


                        }

                }
                textBox.setText(stringBuffer);

                fileInputStream.close();


            } catch (FileNotFoundException e) {
                e.printStackTrace();


            } catch (IOException e) {
                e.printStackTrace();

            }

        }

    });





    randButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {

            // enable description text



            if (data != null) {

                ILineDataSet set = data.getDataSetByIndex(0);

                if (set == null) {
                    set = createSet();
                    data.addDataSet(set);

```

```
                    }

                    Float RandFloat = (float) Math.random() * 40;

                    data.addEntry(new Entry(set.getEntryCount(), (float) (RandFloat)), 0);
                    data.notifyDataChanged();

                    // let the chart know it's data has changed
                    mChart.notifyDataSetChanged();

                    // limit the number of visible entries
                    mChart.setVisibleXRangeMaximum(120);

                    // move to the latest entry
                    mChart.moveViewToX(data.getEntryCount());


                    try {
                        FileOutputStream fileOutputStream = getActivity().openFileOutput("
                            ArduinoData.txt", MODE_APPEND);
                        fileOutputStream.write(RandFloat.toString().getBytes());
                        fileOutputStream.write("\n".getBytes());
                        fileOutputStream.close();
                    } catch (FileNotFoundException e) {
                        e.printStackTrace();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }

                }

            }

        });

        resetButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {


                try {
                    FileOutputStream fileOutputStream = getActivity().openFileOutput("
                        ArduinoData.txt", MODE_PRIVATE);
                  fileOutputStream.close();
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                }

                Toast.makeText(getActivity().getApplicationContext(), "Reset, switch to 
                    section 3 or beyond and back to refresh graph", Toast.LENGTH_SHORT).show
                    ();




            }

        });

```

```

        return rootView;
    }



    private LineDataSet createSet() {

        LineDataSet set = new LineDataSet(null, "Dynamic␣Data");
        set.setAxisDependency(YAxis.AxisDependency.LEFT);
        set.setColor(ColorTemplate.getHoloBlue());
        set.setDrawCircles(false);
        set.setCircleRadius(4f);
        set.setFillAlpha(65);
        set.setFillColor(ColorTemplate.getHoloBlue());
        set.setDrawFilled(true);

        //This sets the values being highlighted by tap gesture
        set.setHighlightEnabled(false);

        set.setValueTextColor(Color.WHITE);
        set.setValueTextSize(9f);
        set.setDrawValues(false);


        return set;
    }






}
```

B.7 Fragment 6

```
package com.example.keyan.cet;

import android.graphics.Color;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.StringRequest;
import com.android.volley.toolbox.Volley;
import com.github.mikephil.charting.charts.LineChart;
import com.github.mikephil.charting.components.Legend;
import com.github.mikephil.charting.components.XAxis;
import com.github.mikephil.charting.components.YAxis;
import com.github.mikephil.charting.data.Entry;
import com.github.mikephil.charting.data.LineData;
```

```
import com.github.mikephil.charting.data.LineDataSet;
import com.github.mikephil.charting.interfaces.datasets.ILineDataSet;
import com.github.mikephil.charting.utils.ColorTemplate;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;

import static android.content.Context.MODE_APPEND;
import static android.content.Context.MODE_PRIVATE;


/**
 * Created by Keyan on 26/05/2017.
 */

public class Tab6 extends Fragment {
    String feedbackstringR;
    String feedbackstringD;
    String feedbackstringE;
    String feedbackstringA;
    String feedbackstringH;


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        final View rootView = inflater.inflate(R.layout.tab6, container, false);

        TextView textBoxRain = (TextView) rootView.findViewById(R.id.textBoxRain);
        TextView textBoxDrop = (TextView) rootView.findViewById(R.id.textBoxDrop);
        TextView textBoxEarth = (TextView) rootView.findViewById(R.id.textBoxEarth);
        TextView textBoxAir = (TextView) rootView.findViewById(R.id.textBoxAir);
        TextView textBoxHum = (TextView) rootView.findViewById(R.id.textBoxHum);


        getfeedbackFromDatabaseR();
        textBoxRain.setText(feedbackstringR);

        getfeedbackFromDatabaseD();
        textBoxDrop.setText(feedbackstringD);

        getfeedbackFromDatabaseE();
        textBoxEarth.setText(feedbackstringE);


        getfeedbackFromDatabaseA();
        textBoxAir.setText(feedbackstringA);

        getfeedbackFromDatabaseH();
        textBoxHum.setText(feedbackstringH);


        return rootView;
```

```
    }

    public void getfeedbackFromDatabaseR(){

        String feedback_url = Config.DATA_URL+'R';

        StringRequest stringRequest = new StringRequest(feedback_url,
                new Response.Listener<String>() {
                    public void onResponse(String response) {
                        showJSONR(response);
                    }
                }
                , new Response.ErrorListener() {
            public void onErrorResponse(VolleyError error) {

                Toast.makeText(getActivity().getApplicationContext(),"Error...",Toast.
                    LENGTH_SHORT).show();
                error.printStackTrace();

            }
        });
        RequestQueue requestQueue = Volley.newRequestQueue(getActivity().
            getApplicationContext());
        requestQueue.add(stringRequest);

    }

    private void showJSONR(String response){
        String feedbackR = "";
        try {
            JSONObject jsonObject = new JSONObject(response);
            JSONArray result = jsonObject.getJSONArray(Config.JSON_ARRAY);
            JSONObject climatEdgedata = result.getJSONObject(0);


            feedbackR = climatEdgedata.getString(Config.FEEDBACK);

        } catch (JSONException e) {
            e.printStackTrace();
        }

        feedbackstringR = feedbackR;

    }

    public void getfeedbackFromDatabaseD(){

        String feedback_url = Config.DATA_URL+'D';

        StringRequest stringRequest = new StringRequest(feedback_url,
                new Response.Listener<String>() {
                    public void onResponse(String response) {
                        showJSOND(response);
                    }
                }
                , new Response.ErrorListener() {
            public void onErrorResponse(VolleyError error) {

                Toast.makeText(getActivity().getApplicationContext(),"Error...",Toast.
                    LENGTH_SHORT).show();
                error.printStackTrace();
```

```
            }
        });
        RequestQueue requestQueue = Volley.newRequestQueue(getActivity().
            getApplicationContext());
        requestQueue.add(stringRequest);

    }

    private void showJSOND(String response){
        String feedbackD = "";
        try {
            JSONObject jsonObject = new JSONObject(response);
            JSONArray result = jsonObject.getJSONArray(Config.JSON_ARRAY);
            JSONObject climatEdgedata = result.getJSONObject(0);


            feedbackD = climatEdgedata.getString(Config.FEEDBACK);

        } catch (JSONException e) {
            e.printStackTrace();
        }

        feedbackstringD = feedbackD;

    }

    public void getfeedbackFromDatabaseE(){

        String feedback_url = Config.DATA_URL+'E';

        StringRequest stringRequest = new StringRequest(feedback_url,
                new Response.Listener<String>() {
                    public void onResponse(String response) {
                        showJSONE(response);
                    }
                }
                , new Response.ErrorListener() {
            public void onErrorResponse(VolleyError error) {

                Toast.makeText(getActivity().getApplicationContext(),"Error...",Toast.
                    LENGTH_SHORT).show();
                error.printStackTrace();

            }
        });
        RequestQueue requestQueue = Volley.newRequestQueue(getActivity().
            getApplicationContext());
        requestQueue.add(stringRequest);

    }

    private void showJSONE(String response){
        String feedbackE = "";
        try {
            JSONObject jsonObject = new JSONObject(response);
            JSONArray result = jsonObject.getJSONArray(Config.JSON_ARRAY);
            JSONObject climatEdgedata = result.getJSONObject(0);


            feedbackE = climatEdgedata.getString(Config.FEEDBACK);
```

```
        } catch (JSONException e) {
            e.printStackTrace();
        }

        feedbackstringE = feedbackE;

    }

    public void getfeedbackFromDatabaseA(){

        String feedback_url = Config.DATA_URL+'A';

        StringRequest stringRequest = new StringRequest(feedback_url,
                new Response.Listener<String>() {
                    public void onResponse(String response) {
                        showJSONA(response);
                    }
                }
                , new Response.ErrorListener() {
            public void onErrorResponse(VolleyError error) {

                Toast.makeText(getActivity().getApplicationContext(),"Error...",Toast.
                   LENGTH_SHORT).show();
                error.printStackTrace();

            }
        });
        RequestQueue requestQueue = Volley.newRequestQueue(getActivity().
           getApplicationContext());
        requestQueue.add(stringRequest);

    }

    private void showJSONA(String response){
        String feedbackA = "";
        try {
            JSONObject jsonObject = new JSONObject(response);
            JSONArray result = jsonObject.getJSONArray(Config.JSON_ARRAY);
            JSONObject climatEdgedata = result.getJSONObject(0);


            feedbackA = climatEdgedata.getString(Config.FEEDBACK);


        } catch (JSONException e) {
            e.printStackTrace();
        }

        feedbackstringA = feedbackA;

    }

    public void getfeedbackFromDatabaseH(){

        String feedback_url = Config.DATA_URL+'H';

        StringRequest stringRequest = new StringRequest(feedback_url,
                new Response.Listener<String>() {
                    public void onResponse(String response) {
```

```
                        showJSONH(response);
                    }
                }
                , new Response.ErrorListener() {
            public void onErrorResponse(VolleyError error) {

                Toast.makeText(getActivity().getApplicationContext(),"Error...",Toast.
                    LENGTH_SHORT).show();
                error.printStackTrace();

            }
        });
        RequestQueue requestQueue = Volley.newRequestQueue(getActivity().
            getApplicationContext());
        requestQueue.add(stringRequest);

    }

    private void showJSONH(String response){
        String feedbackH = "";
        try {
            JSONObject jsonObject = new JSONObject(response);
            JSONArray result = jsonObject.getJSONArray(Config.JSON_ARRAY);
            JSONObject climatEdgedata = result.getJSONObject(0);


            feedbackH = climatEdgedata.getString(Config.FEEDBACK);


        } catch (JSONException e) {
            e.printStackTrace();
        }

        feedbackstringH = feedbackH;

    }

}
```

B.8 Singleton

```
package com.example.keyan.cet;

import android.content.Context;

import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.toolbox.Volley;

public class MySingleton {
    private static MySingleton mInstance;
    private RequestQueue requestQueue;
    private static Context mCtx;

    private MySingleton(Context context){
        mCtx = context;
        requestQueue = getRequestQueue();
    }

    public static synchronized MySingleton getInstance(Context context){
```

```
        if(mInstance == null){
            mInstance = new MySingleton(context);
        }
        return mInstance;
    }

    public RequestQueue getRequestQueue(){
        if(requestQueue == null){
            requestQueue = Volley.newRequestQueue(mCtx.getApplicationContext());
        }
        return requestQueue;
    }

    public <T>void addTorequestque(Request<T> request){
        requestQueue.add(request);
    }
}
```

C Microcontroller Code

```
#include <SoftwareSerial.h>// import the serial library
#include <SoftWire.h>
#include <HIH61xx.h>
#include <AsyncDelay.h>
#include <OneWire.h>
#include <DallasTemperature.h>

SoftwareSerial Genotronex(10, 11); // RX, TX
int ledpin=13; // led on D13 will show blink on / off

int inByte = 0;
int count;
// Set up variables for Rain Sensor
volatile float rainDepth;
volatile float rainDropSize;
volatile boolean haveData = false;

HIH61xx hih;
AsyncDelay samplingInterval;

// Data wire is plugged into pin 2 on the Arduino
#define ONE_WIRE_BUS 2

// Setup a oneWire instance to communicate with any OneWire devices
// (not just Maxim/Dallas temperature ICs)
OneWire oneWire(ONE_WIRE_BUS);

// Pass our oneWire reference to Dallas Temperature.
DallasTemperature sensors(&oneWire);


void setup() {
  Genotronex.begin(9600);
  hih.initialise(A4, A5);
  samplingInterval.start(3000, AsyncDelay::MILLIS);
  sensors.begin();
  Wire.begin(42);  // join i2c bus
  Wire.onReceive (receiveEvent);
}
```

```

bool printed = true;
void loop() {
if (Genotronex.available() > 0) {

inByte = Genotronex.read();

}

if (samplingInterval.isExpired() && !hih.isSampling()) {
    hih.start();
    printed = false;
    samplingInterval.repeat();
    //Serial.println("Sampling started");
    }

  hih.process();

if(inByte==49){

  //Put print commands here
  printE(sensors.getTempCByIndex(0)/4); //-> Earth temp

  if (haveData) \\sensor data read
    {
        printR(rainDepth);
    printD(rainDropSize);
    haveData = false;
    }  // end if haveData

  if (hih.isFinished() && !printed) {
    float temp = hih.getAmbientTemp() / 100.0;

    if (temp <100){
    printA(temp); }//-> Air Temperature
    printH(hih.getRelHumidity() / 100.0); //-> Humidity
  }
delay(2000);

}

}

void printA(float value) {
    Genotronex.write('A');
    Genotronex.println(value);
}

void printR(float value) {
    Genotronex.write('R');
    Genotronex.println(value);
}

void printE(float value) {
    Genotronex.write('E');
    Genotronex.println(value);
}

void printH(float value) {
    Genotronex.write('H');
    Genotronex.println(value);
```

```
}

void printD(float value) {
    Genotronex.write('D');
    Genotronex.println(value);
}

void receiveEvent (int howMany)
 {
 if (howMany >= (sizeof rainDepth) + (sizeof rainDropSize) )
   {
   I2C_readAnything (rainNum);
   I2C_readAnything (rainDropSize);
   haveData = true;
   }  // end if have enough data
 }  // end of receiveEvent

unsigned int I2C_readAnything(& value)
  {
    byte * p = (byte*) &value;
    unsigned int i;
    for (i = 0; i < sizeof value; i++)
          *p++ = Wire.read();
    return i;
  }  // end of I2C_readAnything
```

D Calibration Framework Code

```
fitType = fittype('power1');
prompt = 'Please input data in file with row 1 as theoretical drop size and rows below as test energy data. Filename for Data?';

fileName = input(prompt, 's');
fileData = importdata(fileName);
[m,n] = size(fileData);


prompt1 = 'Would you like to merge data? Y/N';
mergeData = input(prompt1, 's');

if mergeData == 'Y'
    existingData = xlsread('dataOriginal.xlsx');
    [M,N] = size(existingData);

    tempSet = double.empty;

    for i = 1:N
    tempSet = [tempSet fileData(2:m,fileData(1,:)==existingData(1,i))];
    end
    tempSet = padarray(tempSet,[0,N-size(tempSet,2)],'post');
    existingData = [existingData; tempSet];

    Xdata = mean(existingData(2:end,:));
    Ydata = existingData(1,:);
    f = fit(Xdata', Ydata', 'power1');

    figure;
    plot(f,Xdata,Ydata);
    text(3000,4,sprintf('%.3f * x^{%.3f}',f.a,f.b));
```

```

elseif mergeData == 'N'
    Xdata = mean(fileData(2:end,:));
    Ydata = fileData(1,:);

    f = fit(Xdata', Ydata', 'power1');

    figure;
    plot(f,Xdata,Ydata);
    text(3000,4,sprintf('%.3f * x^{%.3f}',f.a,f.b));
else
    sprintf('Invalid Answer');
end
```

E Rain Drop Signal Timings

Table 4: Table showing data for signal timings

Start	545	1590	2581	3597	4580	5571	6579	7597	8592	9641	10645	11623
Peak	569	1608	2599	3614	4596	5590	6596	7617	8611	9661	10670	11648
End	619	1661	2639	3654	4637	5617	6623	7655	8625	9699	10714	11691
Start-End	74	71	58	57	57	46	44	58	33	58	69	68
Start-Peak	24	18	18	17	16	19	17	20	19	20	25	25

Table 5: Table showing data for signal timings

Start	12621	13578	14455	19010	15386	16268	17198	18138
Peak	12633	13603	14469	19021	15397	16280	17222	18149
End	12675	13645	14512	19064	15454	16337	17264	18206
Start-End	54	67	57	54	68	69	66	68
Start-Peak	12	25	14	11	11	12	24	11

F Watchdog Timer Timings

Table 6: Table showing actual watchdog timer timings

Number of Loops	Time/s
1	10
3	36
5	53
6	63
10	102
15	144
20	189
25	240
30	279
32	297

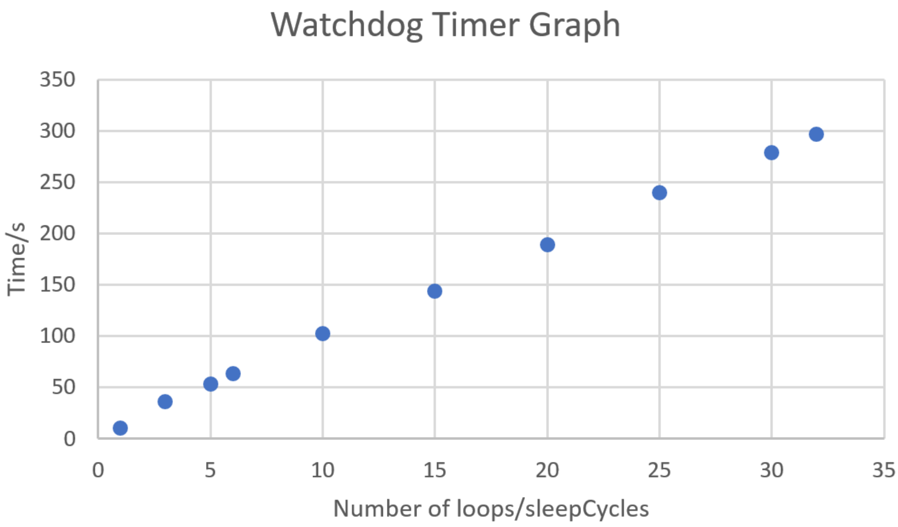


Figure 18: Figure showing the actual watcdog timer timings

G Design Selection Matrix

Table 7: Table showing ranking of different design concepts

	Power Consumption	Affordability	Reliability	Innovation	Total
Acoustic	2	2	3	2	9
Optical	3	3	1	3	10
Piezoelectric	1	1	2	1	5

H Test Data

Table 8: Table showing test data

TEST 1					
Time/s	Theoretical Drop Size/mm	Rain Gauge	Sensor		
		Rain Depth/mm	Rain Depth/mm	Drop Size/mm	
60	0.5	0.1	0.131	0.68	0.31
60	0.8	0.2	0.206	0.77	0.03
60	1	0.55	0.585	1.12	0.063636
60	2	2.5	2.787	1.94	0.1148
60	3	12	13.313	3.21	0.109417
30	4	11	10.682	4.06	-0.02891
30	5	25.5	27.44	4.93	0.076078
TEST 2					
Time/s	Theoretical Drop Size/mm	Rain Gauge	Sensor		
		Rain Depth/mm	Rain Depth/mm	Drop Size/mm	
60	0.5	0.05	0.073	0.56	0.46
60	0.8	0.2	0.249	0.82	0.245
60	1	0.6	0.633	1.15	0.055
60	2	3	3.24	2.04	0.08
60	3	13.5	15.681	3.39	0.161556
30	4	11	9.987	3.97	-0.09209
30	5	23	23.562	5.01	0.024435
TEST 3					
Time/s	Theoretical Drop Size/mm	Rain Gauge	Sensor		
		Rain Depth/mm	Rain Depth/mm	Drop Size/mm	
60	0.5	0.05	0.029	0.41	-0.42
60	0.8	0.25	0.329	0.9	0.316
60	1	0.5	0.346	0.94	-0.308
60	2	4	5.081	2.37	0.27025
60	3	12	11.42	3.05	-0.04833
30	4	9	10.139	3.99	0.126556
30	5	24.5	19.206	4.68	-0.21608
TEST 4					
Time/s	Theoretical Drop Size/mm	Rain Gauge	Sensor		
		Rain Depth/mm	Rain Depth/mm	Drop Size/mm	
60	0.5	0.05	0.19	0.77	2.8
60	0.8	0.3	0.351	0.92	0.17
60	1	0.5	0.455	1.03	-0.09
60	2	4	3.793	2.15	-0.05175
60	3	13	14.203	3.28	0.092538
30	4	10	10.215	4	0.0215
30	5	21	18.839	4.65	-0.1029

Table 9: Table showing more test data

Drop Size	0.5	0.8	1	2	3	4	5
Measured Energy	4.88	5.75	12.89	38.88	181.02	3417.57	6723.45
	3.56	5.1	13.55	74.96	532.51	2757.95	6082.11
	5.23	6.78	19.83	52.55	204.39	3595.34	6871.77
	4.68	9.09	16.44	33.68	340.63	3495.23	5877.56
	5	4.69	19.18	46.91	310.26	3141.73	6864.57
	2.35	6.8	14.16	76.58	381.35	3930.02	5564.69
	4.9	5.92	15.06	57.22	532.82	3985.03	6324.82
	4.64	5.25	14.27	65.04	230.72	4033.69	5966.4
Average Energy	4.405	6.1725	15.6725	55.7275	339.2125	3544.57	6284.421
Variance in Energy	0.906449	1.309387	2.42522	14.79629	128.6644	415.9338	461.1868