Final Year Project 2014/2015

Student Name: Pritpal Sahota

Student ID: 11142375

Supervisor: Dr. Karl Rinne

Course: B.Eng in Computer Engineering

Department: Department of Electronic & Computer Engineering

Title of Project: Arduino-Yun FTP Remote Control / Monitoring of Home

Acknowledgements

I would like to express my deepest appreciation to my project supervisor Dr. Karl Rinne, who has supported me throughout the course of this project, without his supervision and constant help this project would not have been possible.

I would also like to thank technical staff member Jimmy O'Sullivan for providing me with the Hardware components that was required for my project.

I am using this opportunity to express my gratitude to everyone who supported me throughout the course of this project. I am thankful for their aspiring guidance, constructive criticism and friendly advice during the course of this project. I am sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project.

Abbreviations

AES	Advanced Encryption Standard
AHA	Automatic home automation
AREF	Analog Reference
ASCII	American Standard Code for Information Interchange
ATM	Automated Teller Machine
CNG	Compressed natural gas
COM	Communication
DDR2	double-data-rate two
EBCDIC	Extended Binary Coded Decimal Interchange Code
FTP	File Transfer Protocol
FTPS	File Transfer Protocol over SSL
GHZ	Giga Hertz
GND	Ground
GUI	Graphical User Interface
HSC	Heating System Controller
HVAC	*High Voltage Alternating Current*
I/O	Input /Output
I2C	Interface to Communicate
ICSP	In-Circuit Serial Programming
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
KB	Kilo Byte
LED	Light Emitting Diode
LPG	Liquid petroleum gas
mA	Milli Amperes
MB	Mega Byte
MHz	Mega Hertz

MIPS	Million Instructions Per Second
mS	Milli second
NTC	Negative Temperature Coefficient
OS	Operating System
OTP	One Time Programmable
PIR	Passive Infra-Red
PPM	Parts Per Million
PWM	Pulse Width Modulation
RAM	Random Access Memory
RAM	Random Access Memory
RGB	Red, green, and blue
ROM	Read Only Memory
RST	Reset
RX	Receive
SBC	Single-Board Computer
SD Card	Secure Digital Card
SPI	Serial Peripheral Interface
SSL	Secure Sockets Layer
TCP	*Transmission Control Protocol*
TLS	Transport Layer Security
TTL	Transistor-Transistor Logic
TWI	Two Wire Interface
TX	Transmit
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
VPN	Virtual Private Network
WLAN	Wireless Local Area Network

Abstract

With the increasing power of technology, we have access to information from locations that are not within our visibility and are able to communicate this information due to the availability of hardware and mobile devices. This concept of controlling devices in your home by an Android phone and a microcontroller is implemented in this project. Home Automation is a method to have things around the home happen automatically. This project presents the design of Home Automation with low cost and wireless remote control. Such automation can be used to aid the needs of elderly and disabled in homes. The smart home concept improves the standard of living at home.

This current project is designed to sense parameters like temperature, presence of gases, water, motion and humidity, and automate to switch ON or OFF devices like heaters, coolers, etc. The controller selected is Arduino Yun and FTP protocol is enabled to communicate with an Android phone, which is used to monitor and control the sensor parameters.

The implementation is followed by testing and discussion of results. The theory behind the working of different sensors is provided along with their data sheets.

Table of Contents

Chapter 1 – Introduction

A Smart Home is really just a collection of technical home automation concepts that are implamented to meet certain requirements and expectations of the residents in the home. Home automation allows vital home functions to be controlled remotely from anywhere in the world using a computer, iPad, iPhone, etc. connected to the Internet.

The fundamental components of a well-designed home automation system include a computer with the appropriate programming, the various devices and systems to be controlled, interconnecting cables or wireless links, a high-speed Internet connection, and an emergency backup power source for the computer, its peripherals, and the essential home systems.

This project titled '**Arduino-Yun FTP Remote Control / Monitoring of Home**' is designed to sense parameters like temperature, presence of gases, water, motion and humidity, and automate to switch ON or OFF devices like heaters, coolers, etc. The controller selected is Arduino Yun and FTP protocol is enabled to communicate with an Android phone, which is used to monitor and control the sensor parameters.

This report gives the hardware and software, design and implementation details of the Android and Arduino systems along with all the sensors interfaced with them.

Besides the functions already implemented in this project, it can be extended, to remote control telephones and answering machines, fax machines, amateur radios and other communications equipment, and home robots such as automatic vacuum cleaners.

Chapter 2 – Specifications

2.1 Software Specifications

This project may be implemented on any operating system. The software that is required for programming the Arduino-Yun is Arduino IDE and is available for download on their website for free. Ensure to download the latest version of Arduino IDE, which at this time is 1.5.8 version and is compatible with Arduino Yun.

The Arduino IDE 1.5.8 version software can be downloaded for any operating system.

Other software requirements will include any FYP server software such as FileZilla Server to setup your computer or laptop as an FTP server to receive and send files to the FTP client, which in this case is Arduino Yun.

2.2 Hardware Specifications

The hardware requirements needed for the project comprises Arduino Yun Board, SD card to store the monitoring file and controlling to store data, Temperature sensor, resistors, green, red and yellow LED's, breadboard for prototyping the circuit, 5V high level trigger Relay module and jumper wires.

2.3 Android monitoring and control

The parameters to be monitored and the requirements to be met by this project are as follows:

1. **Temperature.**
   Temperature sensor should measure the temperature and based on the measurement control relays to regulate the temperature.
2. **Gas detection**
   Gas sensors should be used to detect the presence of gases in the surrounding environment.
3. **Water detection**
   The presence of water should be detected.
4. **Motion sensor**
   Motion sensors should be used to detect the change of position of objects.

5. **Humidity**

   Humidity sensors should be used to sense and measure the humidity level or moisture content in the air.

6. **Light**

   It should be possible to control the brightness and other parameters related to the lighting device.

2.4 General Requirements

1. Due to shortage of memory available with the AtMega microcontroller, the code for processing the sensor information should be of optimized size. There should be made available sufficient memory for adding bridge libraries, communication with Linux and other relevant utilities.
2. The performance should not degrade with continuous use and software should not crash.
3. Usability: The mobile App should be simple to use and intuitive to understand.
4. The sensors should be limited to simple, low cost ones and the project should serve as a proof of concept for application of microcontrollers to home automation. Advanced home appliances and robots might not be considered due to limitation imposed by (1).
5. The external circuitry with sensors and additional components maybe assembled on a breadboard and not necessarily on a PCB.
6. Design choice in selection of microcontroller and programming language needs to be explained.
7. The results should be consistent and reproducible with repeated tests.
8. Also, events should be marked with timestamps in the files, which are sent to the mobile device to support analysis later.
9. Configuration for relevant parameters should be provided. For example, FTP server details.
10. Data sheets of sensors should be provided.
11. There should be present indicators in Hardware (e.g. LEDs) and on the UI of the mobile device to display the healthiness/ failure of sensors.
12. In specific cases, choice should be provided on the mode of alerting the user on occurrence of alarms. For example, by vibrating the mobile device or by notification.

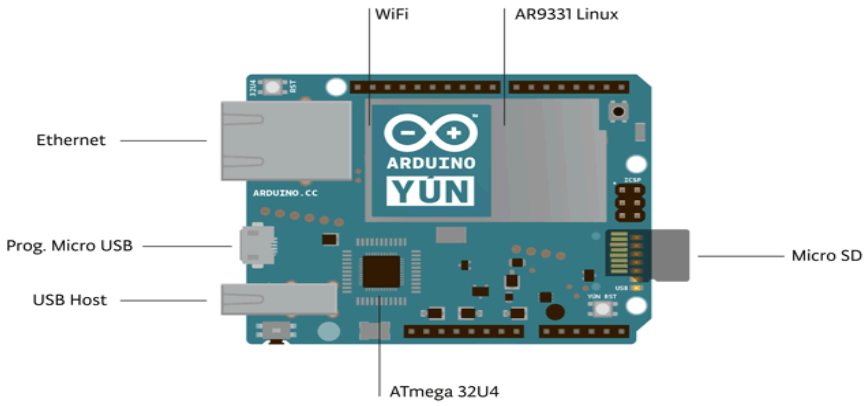
Chapter 3 – Design

3.1 Hardware Design

3.1.1 Arduino Yún System Hardware Description

The Arduino Yún is a microcontroller board (Fig 3.1.1a) based on the ATmega32u4 and the Atheros AR9331. The Atheros processor supports a Linux distribution based on OpenWrt named OpenWrt-Yun. The board has built-in Ethernet and Wi-Fi support, USB port, micro-SD card slot, 20 digital input/output pins (of which 7 can be used as PWM outputs and 12 as analog inputs), a 16 MHz crystal oscillator, a micro USB connection, an ICSP header, and 3 reset buttons.

Fig 3.1.1a - Arduino Yún microcontroller board



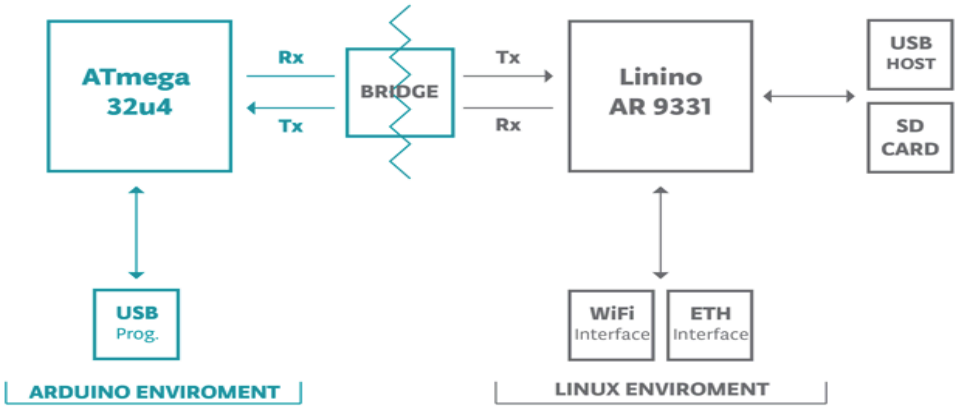
The Yún can communicate with the Atheros AR9331 processor onboard, offering a powerful-networked computer with the ease of Arduino-Yun. With the use of Linux commands like cURL, you can write your own shell and python scripts for robust communications.

The Yún is similar to Arduino Leonardo in that the ATmega32u4 has built-in USB communication port.

The Bridge library as shown in the diagram below enables communication between the ATmega32u4 microcontroller and Linino Linux microprocessor, giving Arduino-Yun sketches the ability to run shell scripts, information from the AR9331 processor. The USB host, network

interfaces and SD card are not connected to the ATmega 32U4, but the AR9331, and the Bridge library enables the Arduino-Yun to interface with those peripherals.

Fig 3.1.1b - Arduino Yún Board Communication Interface



Because the Yún has two processors, the table below shows the characteristics of each one in two separate columns.

AVR Arduino-Yun Microcontroller		**Linux Microprocessor**	
Microcontroller	ATmega32u4	Processor	Atheros AR9331
Operating Voltage	5V	Operating Voltage	3.3V
Input Voltage	5V	Architecture	MIPS @400MHz
Digital I/O Pins	20	Ethernet	IEEE 802.3 10/100Mbit/s
PWM Channels	7	Wi-Fi	IEEE 802.11b/g/n
Analog Input Channels	12	USB Type-A	2.0 Host
DC Current per I/O Pin	40 mA	Card Reader	Micro-SD only
DC Current for 3.3V Pin	50 mA	RAM	64 MB DDR2
Flash Memory	32 KB (of which 4 KB used by bootloader)	Flash Memory	16 MB
SRAM	2.5 KB		
EEPROM	1 KB		
Clock Speed	16 MHz		

Power

The board is powered via:

- Micro-USB connection with 5 VDC (or)
- Regulated 5 VDC to Vin pin

Power pins are as follows:

- V_{IN}: Regulated 5V.
- 5V: Power for microcontrollers and other components on the board. Can come either from V_{IN} or be supplied by USB.
- 3V3: 3.3 V, 50 mA (max) supply generated by the on-board regulator.
- GND: Ground pins
- IO_{REF}: Operating voltage of IO pins of the board (i.e. VCC for the board).

Input and Output

All I/O lines of Atheros AR9331 are tied to the 32U4 and hence not accessible.

Features of the 20 digital I/O pins:

- Used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions.
- Operates at 5 volts.
- Source / sink current: 40 mA (max)
- Internal pull-up resistor (disconnected by default) of 20-50 kΩ.
- Pins with specialized functions:

Serial	0 (RX) and 1 (TX). Receive & transmit TTL serial data using ATmega32U4
TWI	2 (SDA) and 3 (SCL). Support TWI communication using the Wire library
Ext. Interrupts	3 (int. 0), 2 (int. 1), 0 (int. 2), 1 (int. 3) & 7 (int. 4). These pins can be configured to trigger an interrupt
PWM	3,5,6,9,10,11, and 13. Provide 8-bit PWM output with the analogWrite() function
SPI	On the ICSP header. Supports SPI communication using the SPI library.
LED	13. Pin is HIGH value: LED is on; pin is LOW: LED off.
Analog Inputs:	A0 - A5, A6 - A11 on digital pins (4, 6, 8, 9, 10, and 12). 12 analog inputs (also digital I/Os). Upper end of their range can be measured using the AREF pin and the analogReference() function.

AREF	Reference voltage for analog inputs. Used with analogReference().

3 reset buttons with different functions on the board:

- **Yún RST -** Bring this line LOW to reset the AR9331 microprocessor.
- **32U4 RST -** Bring this line LOW to reset the ATmega32U4 microcontroller.
- **WLAN RST -** To restore the Wi-Fi to the factory configuration.

Communication

- UART TTL (5V) serial communication.
- I2C (TWI) and SPI communication
- Keyboard and mouse
- Ethernet and Wi-Fi.
- USB host.

Programming

Arduino Yun can be programmed with Arduino IDE software.

3.1.2 Temperature Sensor

3.1.2.1 Theory of Temperature Sensor

Temperature sensors measure the temperature of its surroundings or of the surface on which they are placed, and then translates it into a form that can be understood by the observer or the information is passed on to another device or computer for further processing / action. The most common example of a temperature sensor is thermometer. Temperature sensors are broadly classified as Contact and Non-Contact sensors. Contact sensors measures the temperature of the body to which the sensor is in contact with. For example, temperature of systems in aerospace are measured by placing thermistors on the electronics packages using adhesive. Contact sensors work efficiently only when there is a thermal between the body and the sensor.

Non-Contact sensors measure the heat radiations from the environments within a given area without being in actual contact with the body. Examples are temperature sensors that are used to measure the temperature of a medium that are controlled using air-conditioning equipment.

Applications

Examples of applications of temperature sensors are in Thermostatic Controls, Refrigerators and boilers in HVAC systems, industrial systems like chemical factories to maintain process temperatures, consumer products like PCs and servers, medical equipment like thermometers and massaging equipment. They are also used in Automation in home and industry, like the project in this report. In many applications it is used as a closed loop control of temperature like water heating geysers, wind power and compressed air.

3.1.2.2 Theory of Heating System Controller

Theory of the heating system controller as discussed here as it utilizes the output of the temperature sensor for it's functioning.

Heating System Controller (HSC) is a device used to control temperature. HSC takes an input from a temperature sensor and has an output that is connected to a control component such as a heater or fan.

Heating System Controller relies upon as an input. It compares the actual temperature to the desired control temperature (set point), and it provides an output signal that maintains a desired value to a control component such as a heater or fan through reley.

This Heating System Controller is based on On/Off Control. An On/Off controller is the simplest form of temperature control device. The output from the Controller is either on or off, with no middle state. An On/Off controller will switch the output only when the temperature crosses the set point. The output of the Heating System below the set point, and off when it's above the set point. Since the temperature crosses the set point to change the output state, the process of continually cycling of the temperature going from below set point to above and back below will occur. In cases where this cycling occurs rapidly, and to prevent damage to devices such Relays, an On/Off differential or "hysteresis" is added to the operations of the Controller. This differential value is set so that the output will turn off or on when the temperature exceeds the set

point by a certain amount. On/Off differential value prevents the output from making fast, continual switches if the cycling above and below the set point occurs very fast.

On/Off control is usually used where a precise control is not necessary, in systems, which cannot handle having the energy turned on and off frequently.

This type of Heating Control System typically uses a Relay to shut down a process when a certain temperature is reached, which is simple mechanism construction and is widely used.

(Fig 3.1.2a) below illustrates how the Heating System Controller works and (Fig 3.1.2b) is a block diagram of the Heating System Controller.

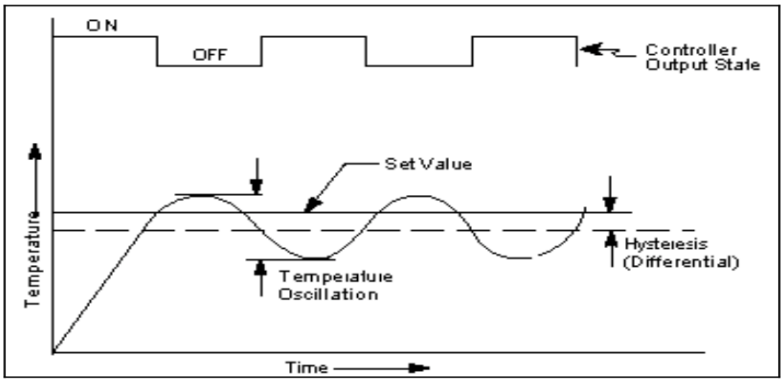


Fig 3.1.2a: Working Principle of Heating System Controller

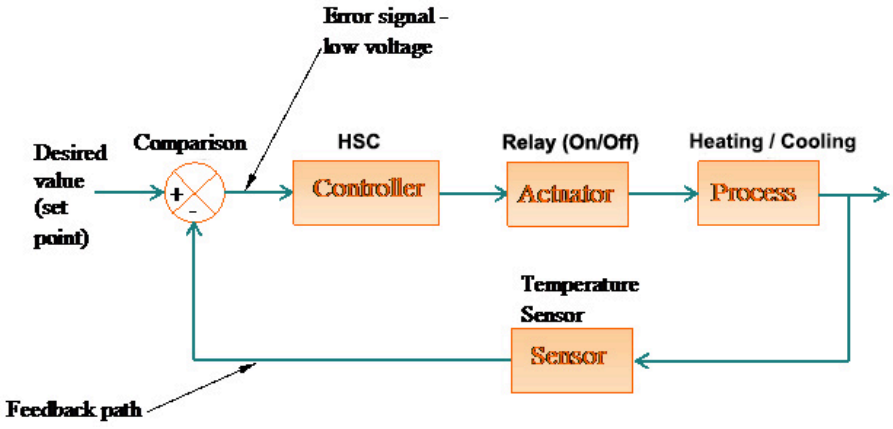


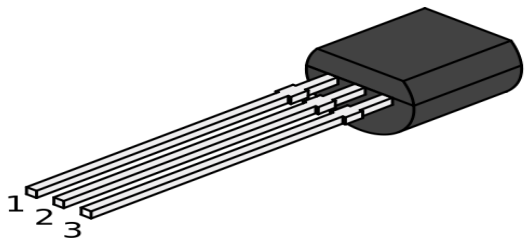
Fig 3.1.2b: Block diagram of Heating System Controller

3.1.2.3 DS18B20 Temperature Sensor specification

Description

The DS18B20 digital thermometer provides 9-bit to 12-bit Celsius temperature measurements and has an alarm function with nonvolatile user- programmable upper and lower trigger points. It communicates over a 1-Wire bus that by definition requires only one data line (and ground) for communication with a central microprocessor. It has an operating temperature range of -55°C to +125°C and is accurate to ±0.5°C over the range of -11°C to +85°C. In addition, the DS18B20 can derive power directly from the data line, eliminating the need for an external power supply.
Each DS18B20 has a unique 64-bit serial code, which allows multiple DS18B20s to function on the same 1-Wire bus. Thus, it is simple to use one microprocessor to control many DS18B20s distributed over a large area. Applications that can benefit from this feature include HVAC environmental controls, temperature monitoring systems inside buildings, equipment, or machinery, and process monitoring and control systems.

Diagram



Specifications

- Communication Interface: 1-Wire® Interface requiring a single port pin
- Each Device has a Unique 64-Bit Serial Code Stored in an On-Board ROM
- Multidrop Capability Simplifies Distributed Temperature-Sensing Applications
- Can Be Powered from Data Line; Voltage: 3.0 V to 5.5 V
- Temperature range: -55°C to +125°C

- Resolution: 9 to 12 Bits
- Accuracy: ±0.5°C from -10°C to +85°C
- Output: 12-Bit Digital Word in 750 ms (Max)

Pin assignment

1. GND
2. DQ
3. V_{DD}

Dimensions: 1.8 cm x 0.4 cm x 0.3 cm

Weight: 28 g

Fig 3.1.2c: Circuit details of Temperature Sensor, Cooling and Heating System Controller

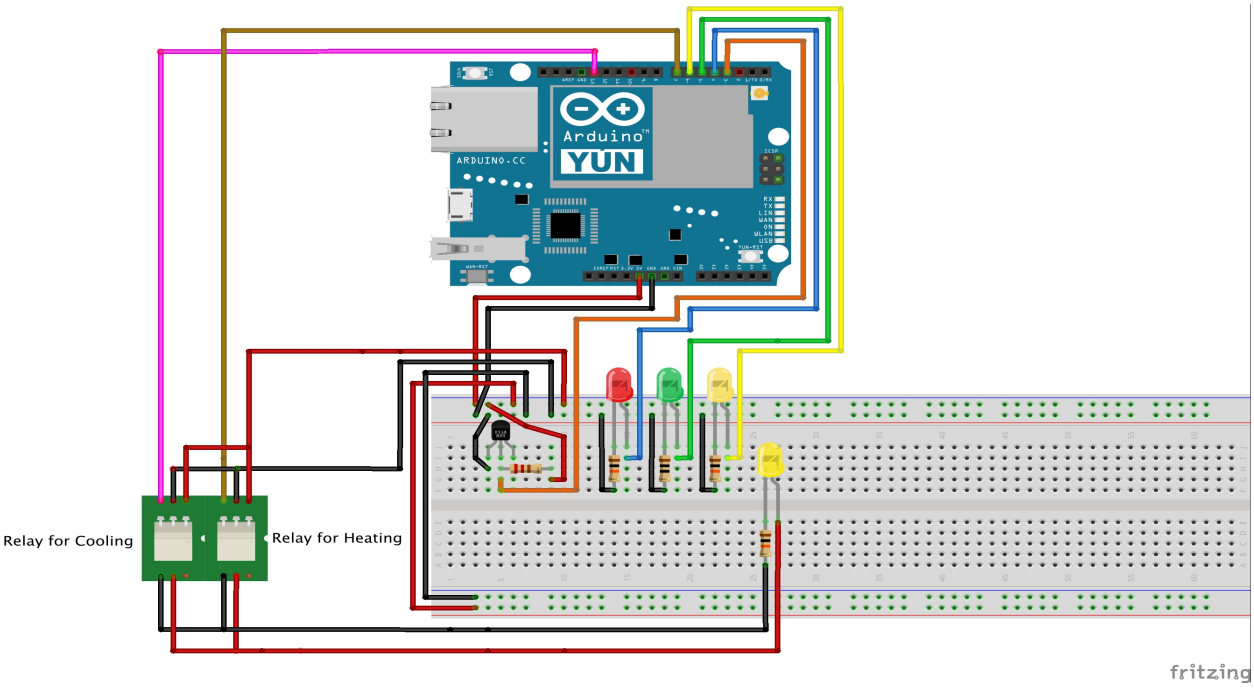
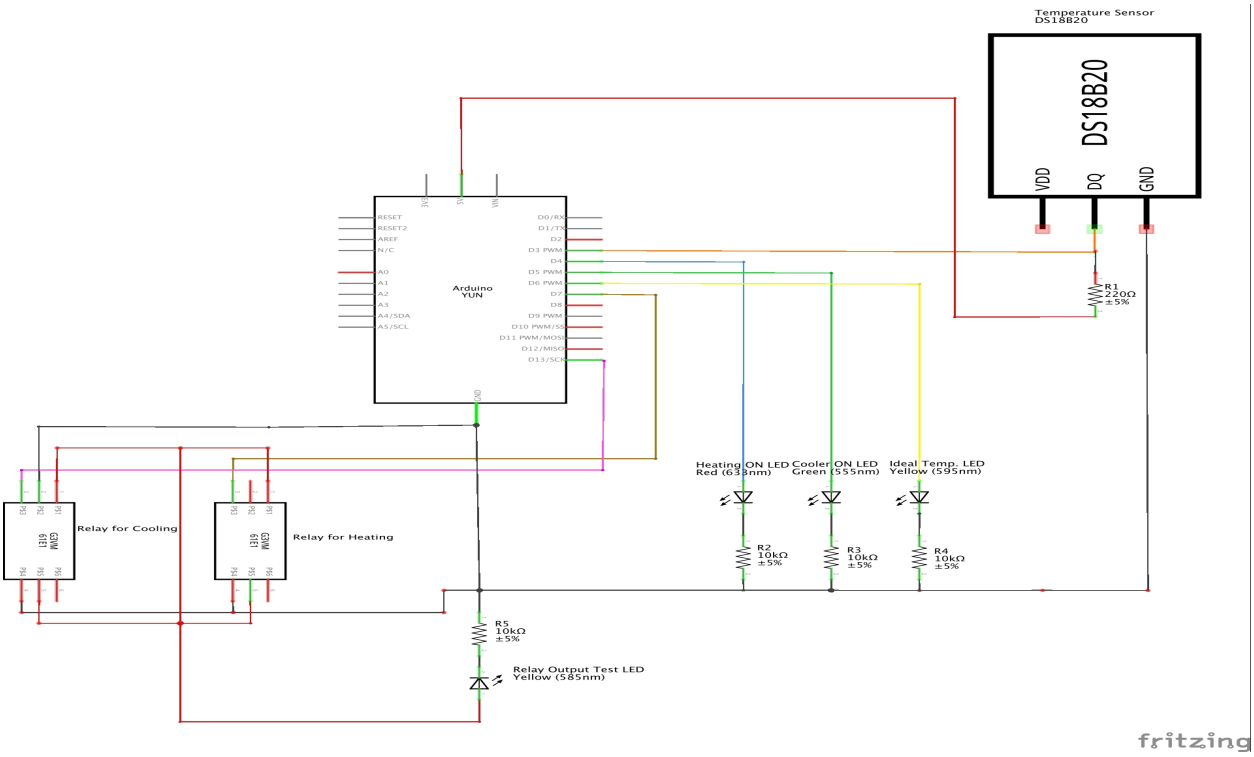


Fig 3.1.2d: Schematic of Temperature Sensor, Cooling and Heating System Controller



Details of LEDs – Coloring Scheme:

- **Red:** The red LED indicates that sensor temperature is less then (desired temperature – hysteresis) and hence the relay for heating must be turned on to switch on the heating. When the sensor temperature reaches the (desired temperature – hysteresis), then Red LED is turned off.
- **Green:** When the sensor temperature goes above the (desired temperature + hysteresis), then green LED is turned on, relay for heating is switched off to stop the heating process and relay for cooling is switched on to start the cooling process to bring the sensor temperature to be below (desired temperature + hysteresis).
- **Yellow:** The yellow LED is used to notify when the sensor temperature is just ideal, i.e. it is less then (desired temperature + hysteresis) and greater then (desired temperature – hysteresis).
- **Yellow (Relay Output Test LED):** This yellow LED is a relay output test LED that is connected to the output of relay and it signals when the relay is switched on.

3.1.3 Gas Detector Sensor

3.1.3.1 Theory of Gas Sensor

Gas sensors are used to detect the presence of gases in its surroundings. The output of the gas sensor is used as an input to a control system to start or stop a process automatically. As gas sensors are used in hazardous environment also, the output of the gas can be used to sound an alarm for appropriate action.

Monitoring of gases produced is a very critical requirement in home and industrial appliances like air conditioners, chimneys and safety systems that monitor gases. The gas sensors are small in size and shaped like a nose and housed in a steel case; they react instantaneously to the gas present, thus keeping the system updated about any modifications that occur in the molecules at gaseous state.

The specifications of the gas sensor vary based on the type of the gas it is designed to sense, their sensitivity levels and physical dimensions of the sensor. The sensor works by ionizing the gas near the sensing element, into its constituents and they are then adsorbed by the sensing element. Based on the amount of adsorption a potential difference exists in the element, which results in a current (also called heating current) at the connecting leads. This current is the input to the processor.

Applications

Gas sensors are used in life saving scenarios to detect combustible, flammable and toxic gases and oxygen depletion, in industrial scenarios like oilrigs, refineries and manufacture processes for monitoring. They are also applied in fire-fighting, and waste-water treatment facilities and to monitor smoke in vehicles and homes.

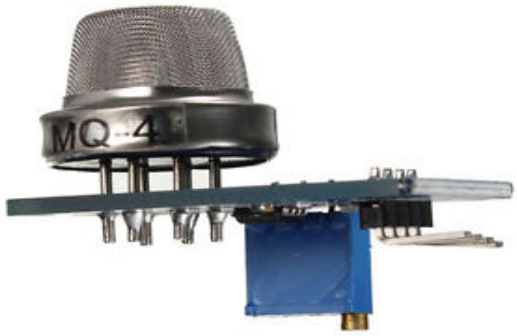
3.1.3.2 MQ-4 DC5V Gas Detector Sensor Module for Arduino

Description

The MQ-4 DC5V Gas Detector Sensor Module is used for detecting combustible gases such as LPG, butane, methane, alcohol, propane, hydrogen and smoke. It has a both analog output and TTL level outputs. It is highly sensitive to LPG, gas, alcohol, hydrogen and many other gases

too. As it has fast response and recovery features along with a long service life, it is highly suitable for home or factory gas leakage detection.

Diagram



Specifications

- Voltage: 5 V DC
- Detecting Range: 300 to 10000ppm
- Output: TTL signal – Active low

Dimensions: 32 cm x 22 cm x 27 cm

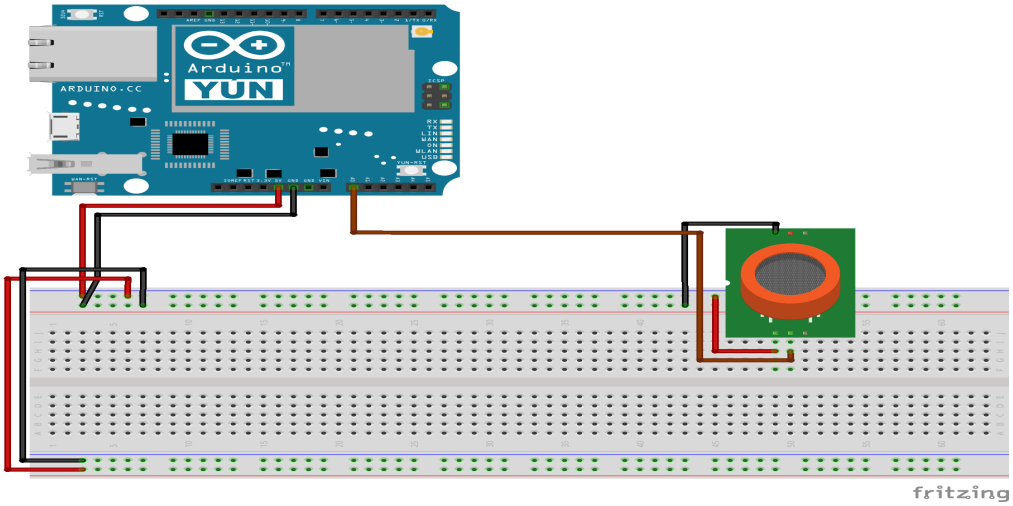
The MQ series of gas sensors use a small heater inside with an electro-chemical sensor. They are sensitive for a range of gasses and are used indoors at room temperature. They can be calibrated more or less, but a known concentration of the measured gas or gasses is needed for that.

MQ-4 Gas sensor

The MQ-4 gas sensor is sensitive for Methane and CNG Gas
Pin out details:

- AOUT - Analog value output (Analog value of detected gas, can be remapped to %)
- DOUT - Digital value output (Turned on when gas exceed a certain level). It may not be used.
- VCC - +5V
- GND – Ground

Fig 3.1.3a: Circuit details of Gas Sensor



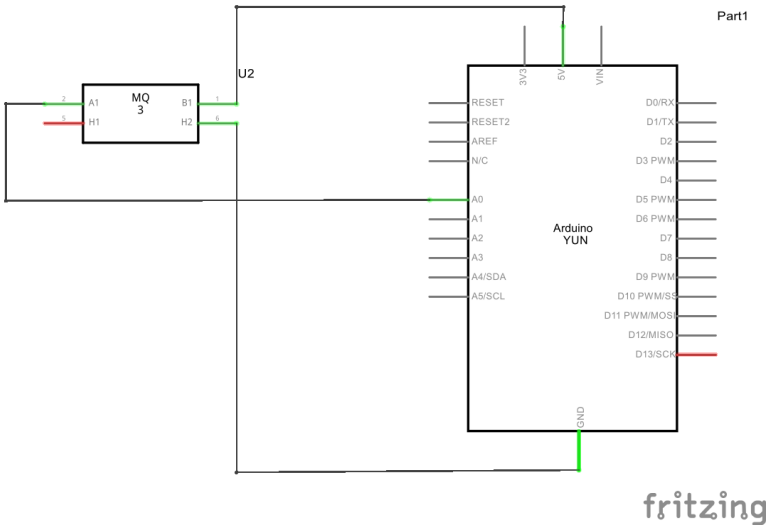


Fig 3.1.3b: Schematic of Gas Sensor

3.1.4 Water Detector

3.1.4.1 Theory of Water Detector

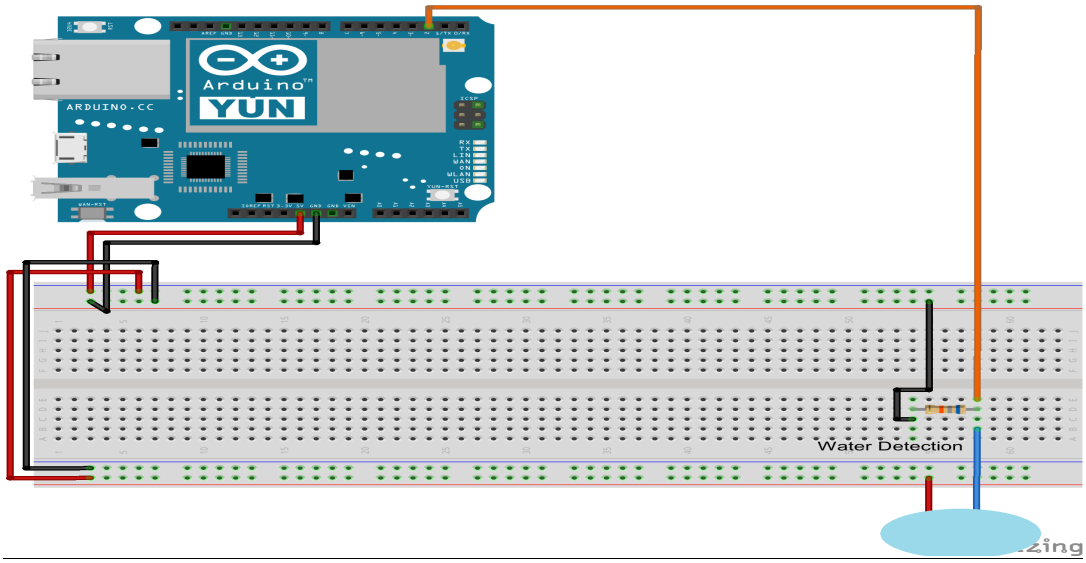
Tap water is a good conductor of electricity. This concept is used to detect the presence of water. Two open wires at the extreme ends of a resistor in series with a DC voltage serve as a water

detector. The current that flows through the resistor when the 2 wire-ends are immersed in water gives an indication of water.

A 68k Ω wire wound resistor in series with 5 V supply with open-ended wires are used as a water detector.

When the blue and red wire are in the water at the same time, as tap water being a good conductor of electricity, a current flows through the resistor and Arduino-Yun reads HIGH on the second pin.

Fig 3.1.4a: Circuit details of Water Detector



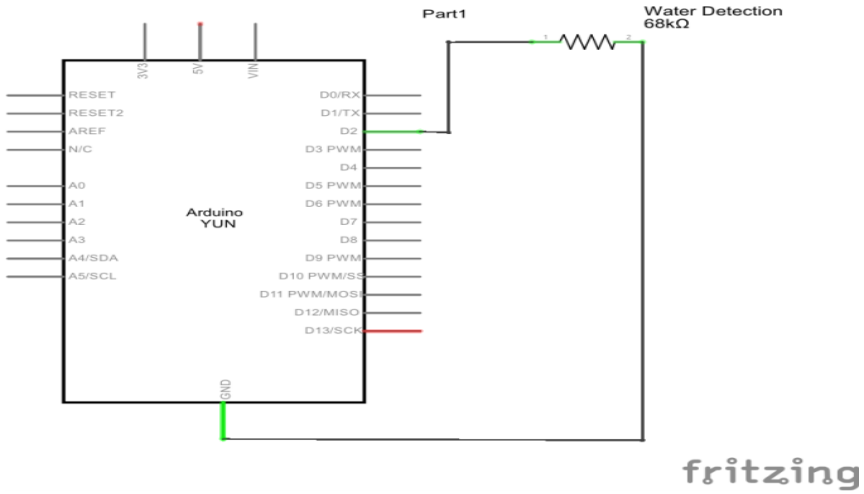


Fig 3.1.4b: Schematic of Water Detector

3.1.5 Motion Sensor

3.1.5.1 Theory of Motion Sensor

The motion sensor is designed to sense and measure the changes in position, velocity, and acceleration of moving objects. They work by using the properties of ultrasound to measure the position of persons or things when they move. Motion sensors may be IR human detection sensor or factory automation sensors used with factory equipment.

Types of motion sensors

Motion sensors are classified based on the principle used to sense motion. They are Passive Infrared motion sensors (also called PIR motion sensor) and Area Reflective motion sensors,

The Passive Infrared motion sensors are so named because they are capable of detecting the amount of change in IR rays when a person or object with a differential temperature with reference to the surroundings, moves. It's field-of-view is large but detection distance is not settable. It can also detect human presence and the motion of people by their body temperature.

The Area Reflective motion sensors also derive their name from their functionality. The sensor emits IR rays from an LED and receives back the reflected rays to the sensor. The forward and reflected light are correlated to measure the distance to the person or object and detects whether or not it is exists within a specific distance. This type of sensor can be used for detection within specific ranges and the resolution achieved is 1 cm in 5 cm to 10 cm range and 10 cm in 20 cm to 200 cm range. The Passive Infra-Red motion sensor also called as PIR motion sensor is a pyro electric device that detects motion by sensing changes in the infrared (radiant heat) levels emitted by surrounding objects.

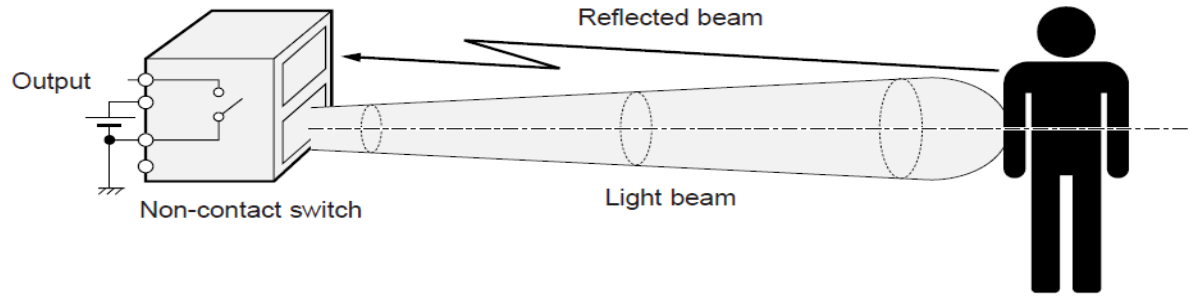


Fig 3.1.5a: Sensor detecting the presence of Human Body

As shown in Fig 3.1.5a, the sensor sends out a ray of light toward the human body, and then uses the reflected light to measure the distance and determine whether there is a person within a given distance of the sensor. If the sensor detects that there is a person within the given distance, it sets an output on non-contact switch to turn on.

Applications

The signal from the motion sensor is used to automatically turn indoor lighting on and off, to enable the monitor camera for door entrance and to switch ON/OFF the entrance / sink / desk /dresser light and also computer monitor and TV. This leads to energy saving. As a verbal guidance machine for the blind it serves a societal cause.

In hospitality areas the motion sensors can be utilized for the control of air conditioner louver, detection of toilet use, automatic water faucets and as non-contact switch for the toilet.

In the public areas it is used as customer sensor in automatic ticket gate, parking meter, and customer detecting sensors for automated teller machines

3.1.5.2 Pyro electric Infrared PIR Motion Sensor Detector

Description

The Motion sensor detector is used to measure position, velocity, and acceleration of moving objects. It uses ultrasound to measure the position of carts, balls, people, and other objects. It can measure objects as close as 15 cm and as far away as 6 m. The PIR (Passive Infra-Red) Sensor is a pyro electric device that detects motion by sensing changes in the infrared (radiant heat) levels emitted by surrounding objects.

Diagram

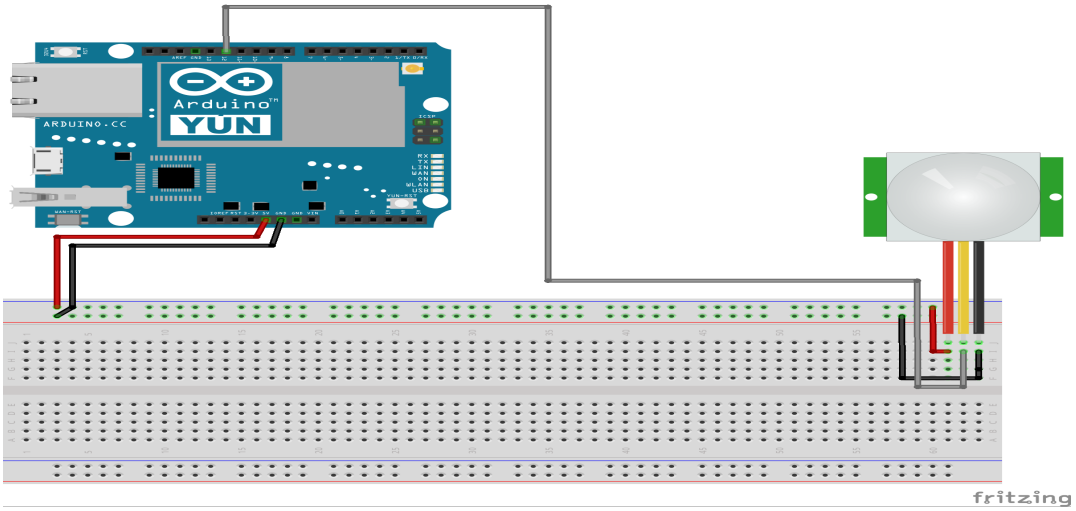
Specifications

- Material: Semiconductor + PVC
- Voltage: DC 5 to 20 V
- Quiescent Current: 65μA
- Output Voltage: High 3V / Low 0V
- Delay time: 0.3 to 18 S (adjustable)
- Blockade Time: 0.2 S
- Trigger Method: L: unrepeatable trigger / H: repeatable trigger
- Detecting Range: 7m, less than 120 degree angle
- Working Temperature: -15° to 70° C
- Screw hole pitch: 2.8mm, screw hole diameter: 2mm

Dimensions: 3.2 cm x 2.4 cm x 2.5 cm

Weight: 6 g

Fig 3.1.5b: Circuit details of Motion Sensor



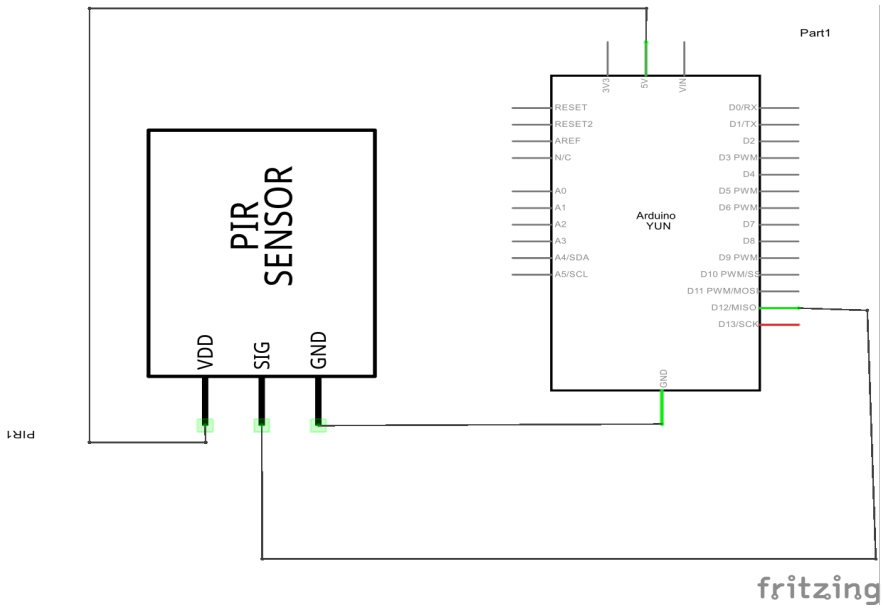


Fig 3.1.5c: Schematic of Motion Sensor

3.1.6 Humidity Sensor

3.1.6.1 Theory of Humidity Sensor

Humidity sensors sense and measure the humidity level or moisture content in the air. If humidity level is less than a specific number, it leads to generation of static electricity leading to fire or damage of electronic components, which are ESD sensitive. On the other hand consequences of high level of humidity are loss of material due to bacterial and fungal growth.

The data from the humidity sensor serves as input to the humidity controller to increase or decrease the humidity in the area as the case may be. Humidity sensors may be of the capacitive type or resistive type based on their working principle.

Types of Humidity Sensors

Dielectric constant is directly proportional to the relative humidity in immediate environment. This property of the material used in the 'Capacitive Humidity Sensors' to measure the humidity of the area after calibration. Resistive humidity sensors use Semiconductor devices like lithium

chloride that can measure resistance and this property is dependent on the presence / level of humidity in the material. Also these materials have the property of absorbing water vapor in the air. Hence when they are used as a humidity sensor, the resistance of the material is measured and this gives an indication of the level of humidity. These sensors are capable of working in high temperatures.

Applications

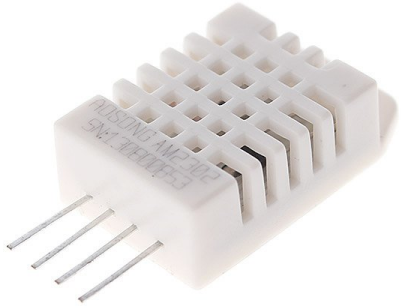
Humidity sensors are used to sense and control the humidity level in Electronic labs to make them ESD safe areas. They are used in Transmission lines, antennas, and waveguides used in telecommunications for the same reason. They are also used in HVAC systems in the industry to maintain optimum humidity levels.

3.1.6.2 AM2302 Temperature Humidity Sensor Module

Description

The AM2302 digital temperature and humidity sensor gives a calibrated digital signal output of the temperature and humidity parameters. It uses dedicated digital modules and temperature and humidity sensor technology. The sensor includes a capacitive humidity sensing element and an NTC temperature measurement device, and it is connected with a high-performance 8-bit microcontroller connected for good quality. It has fast response, anti-interference ability, and other advantages. Each AM2302 sensor is accurately calibrated for humidity in a calibration chamber. Calibration coefficients are stored in the OTP program memory. The internal sensor of the detection signal processing process calls these calibration coefficients. Single wire serial interface allows quick and easy system integration. Other features like ultra-small size, low power consumption and signal transmission range of 20 meters, makes it the most useful device for the related applications.

Diagram



Specifications

- Power supply: 3.3-5.5V DC
- Output signal: digital signal via 1-wire bus
- Sensing element: Polymer humidity capacitor
- Operating range: 0-100%RH
- Accuracy: ±2%RH (Max ±5%RH)
- Resolution: 0.1%RH

Pin assignment

1. VDD power supply: 3.5-5.5V (recommended supply voltage is 5V)
2. SDA serial data, bidirectional
3. NC
4. Ground

Sensitivity parameters

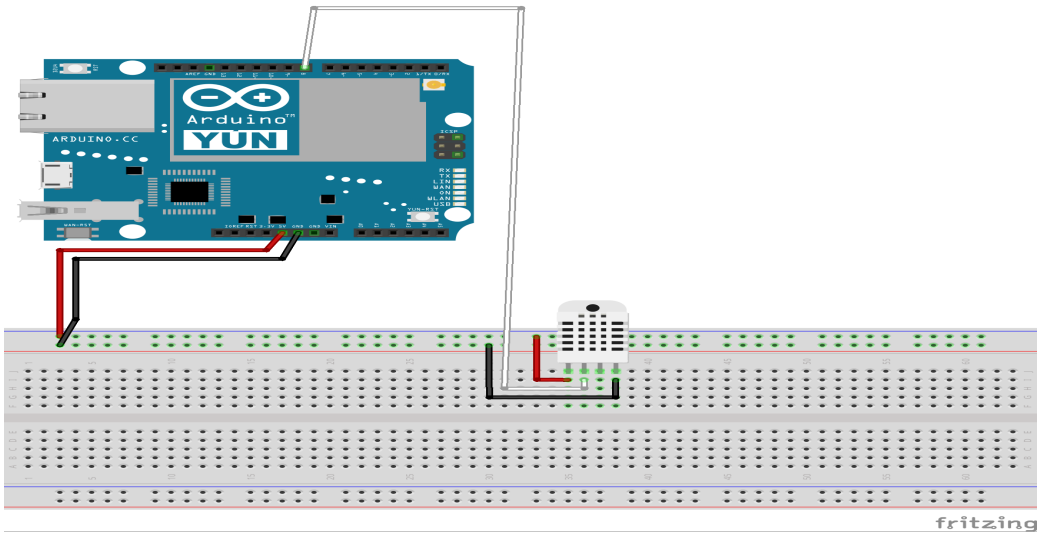
Relative Humidity: 0 ~ 100% RH

Temperature: -20 ~ 80

Dimensions: 3.3 cm x 1.5 cm x 0.8 cm

Weight: 5 g

Fig 3.1.6a: Circuit details of Humidity Sensor



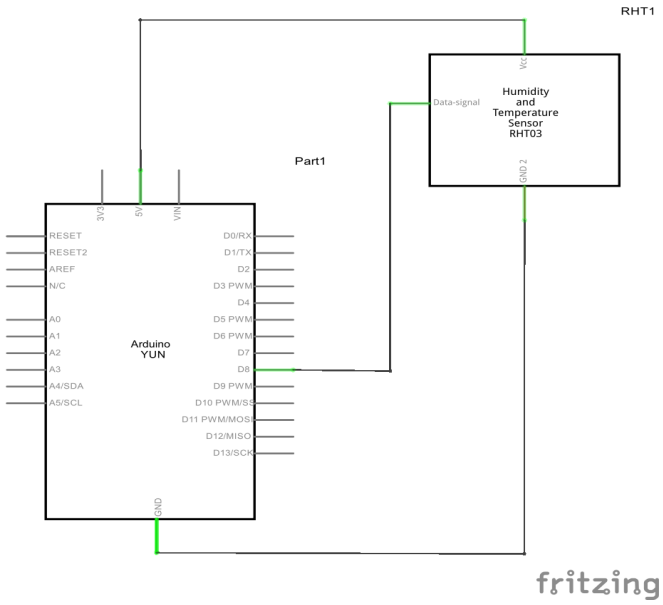


Fig 3.1.6b: Schematic of Humidity Sensor

3.1.7 PWM LED Light Control

3.1.7.1 Theory of PWM LED Light Control

The aim is to control the switching on/off of the LED light as well as control the intensity or brightness of LED light. This can be most efficiently done using PWM (Pulse Width Modulation) pulses.

Pulse Width Modulation or PWM is the scheme commonly used when controlling power using a microcontroller. Amongst the many applications, the most popular are controlling the brightness of LEDs and fading out LEDs instead of turning them ON and OFF, controlling the speed of a motor, etc.

The brightness of an LED is directly proportional to the power, which is sent to the LED, by using a variable resistor called potentiometer. But in digital circuits like microcontrollers, there are only two states, on and off or '0' and '1'. Hence to vary the power from a microcontroller, either we have to use a Digital to Analogue Convertor or PEM. PWM 'simulates' the concept of varying the levels of power by oscillating the output from the microcontroller.

If, for a brief time, we turn the LED 50% ON and 50% OFF (see figure below), the LED will appear half as bright as when it was always ON. On the other figure the LED is 25% ON and 75% OFF, the LED will appear further less bright. By varying (or 'modulating') the width of the ON pulse, the light output from the LED can be controlled. Hence it is called PWM or Pulse Width Modulation.

.

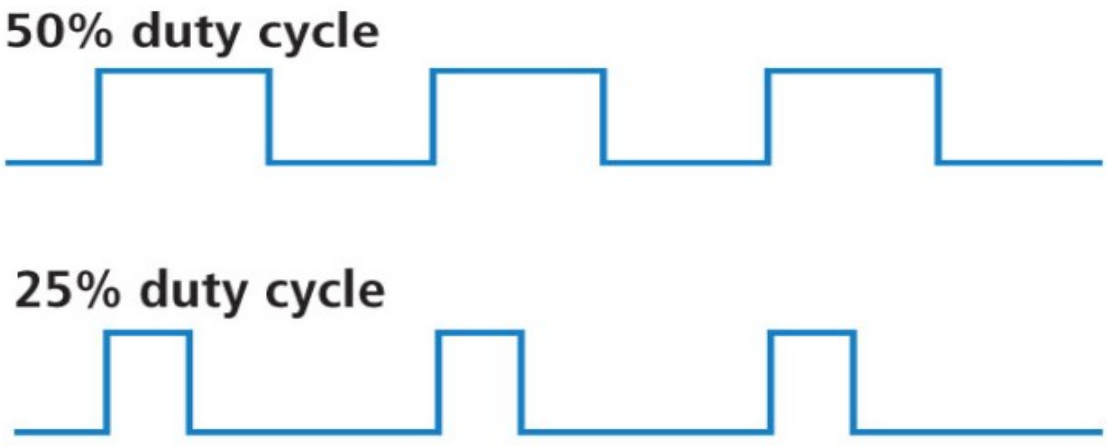
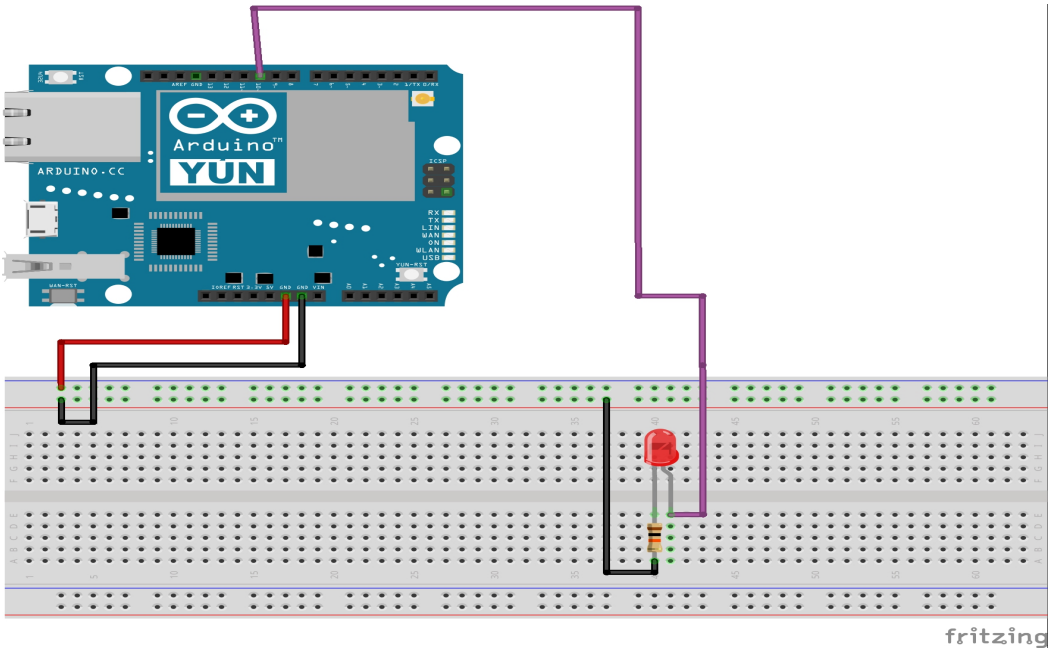


Fig 3.1.7a: Circuit Details of LED Light



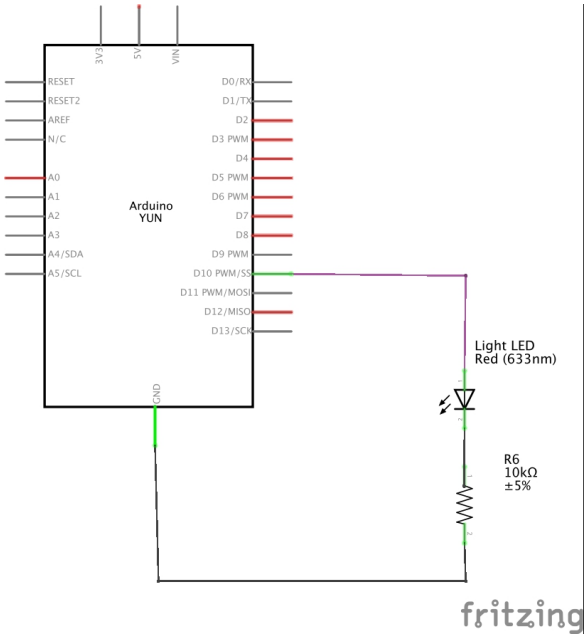


Fig 3.1.7b: Schematic of LED Light

3.1.8 Final Hardware Description Diagram

The final Breadboard integrating all the sensors and the Arduino board is as shown in the following figure (Figure 3.1.8a). The schematic details of the circuit are shown in the (Figure 3.1.8b).

Fig 3.1.8a: Hardware Details of the System

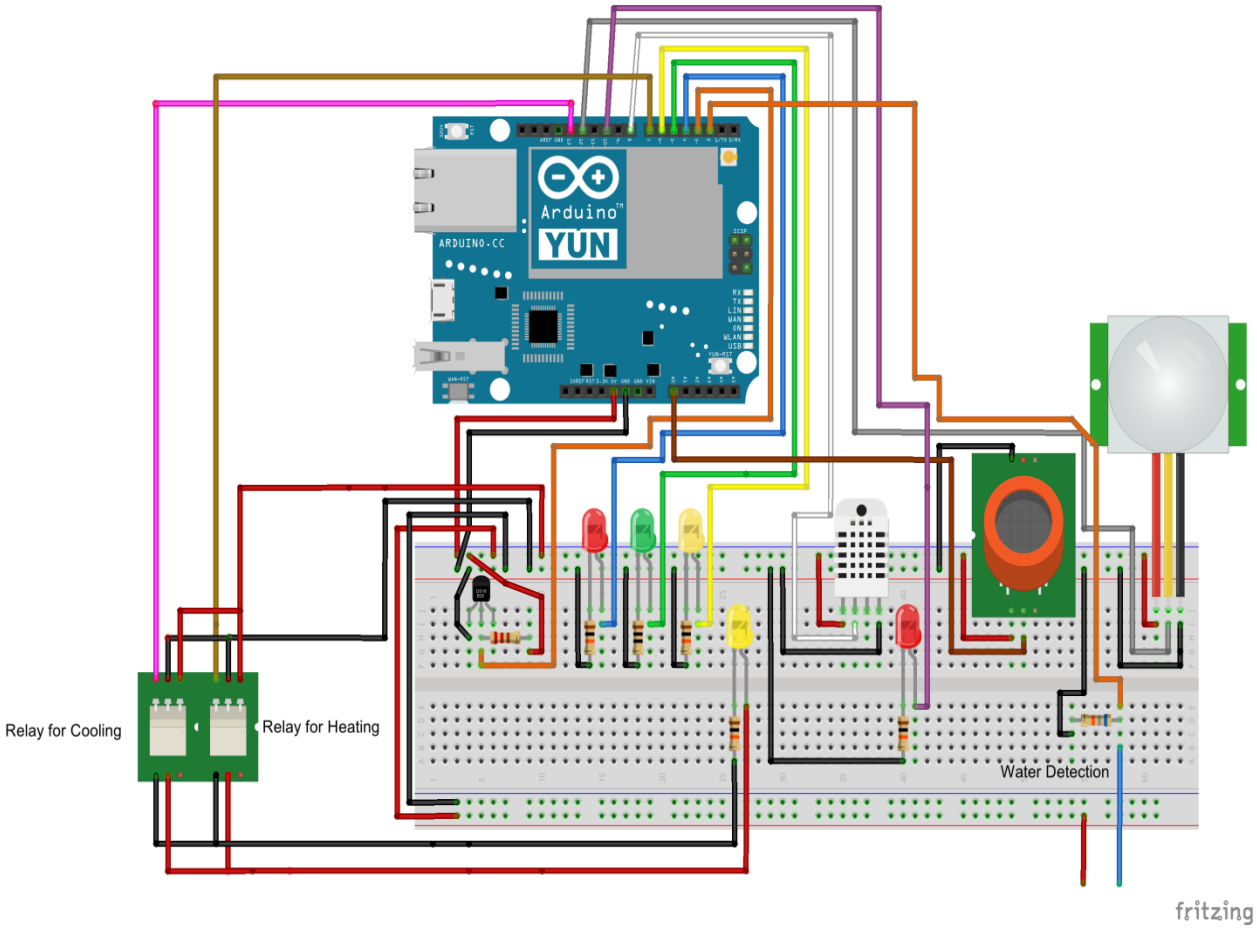
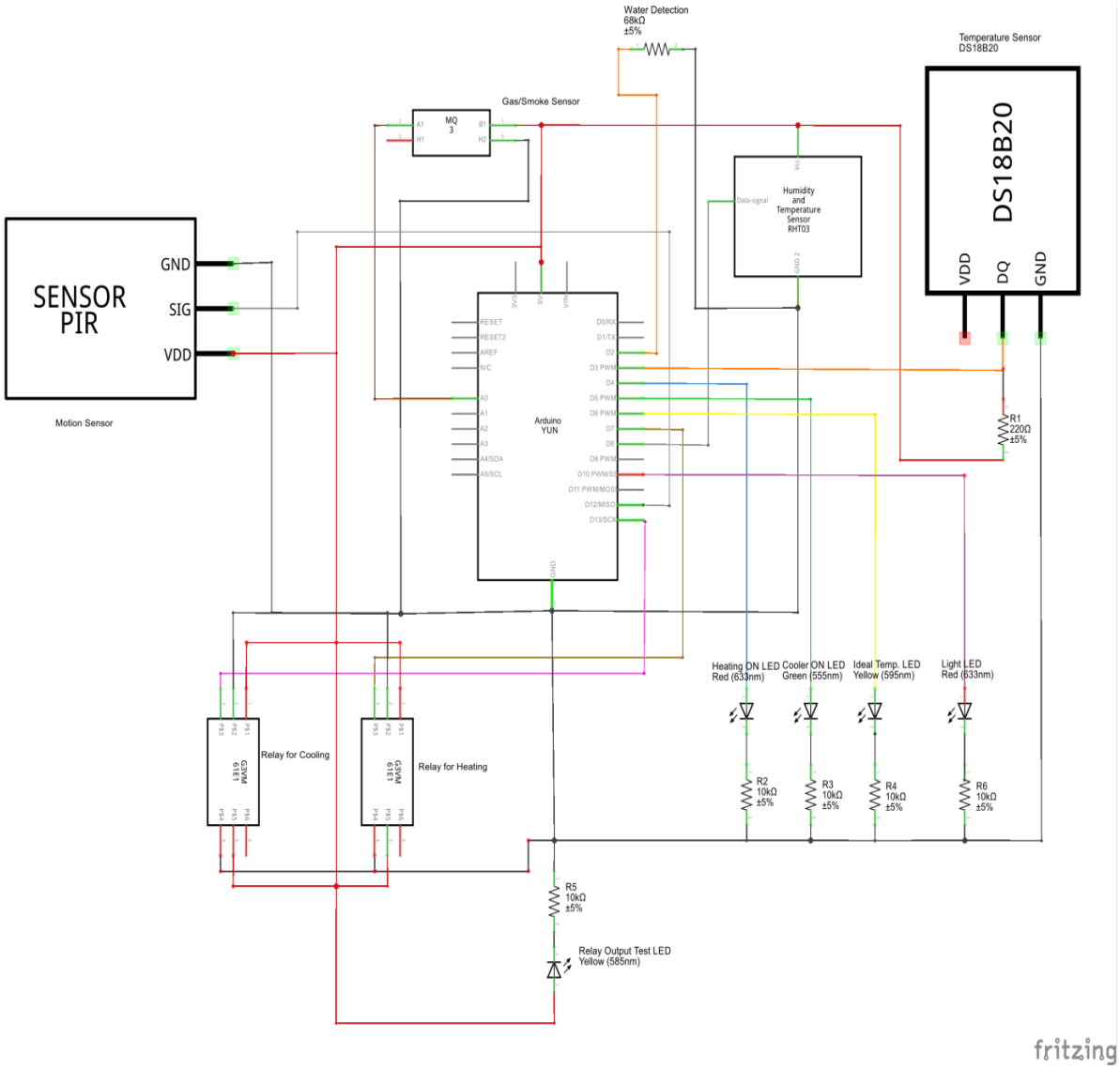
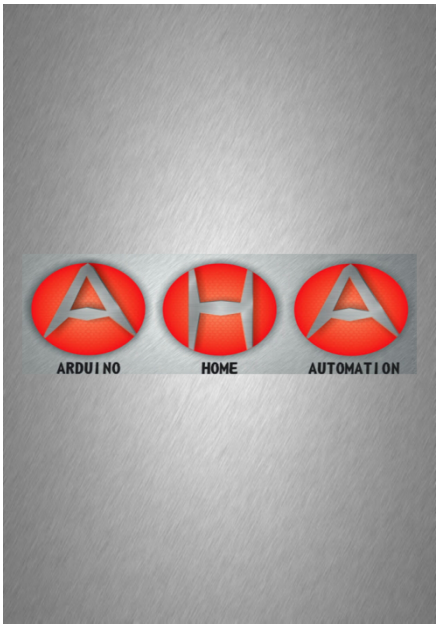


Fig 3.1.8b: Schematic Diagram of the System



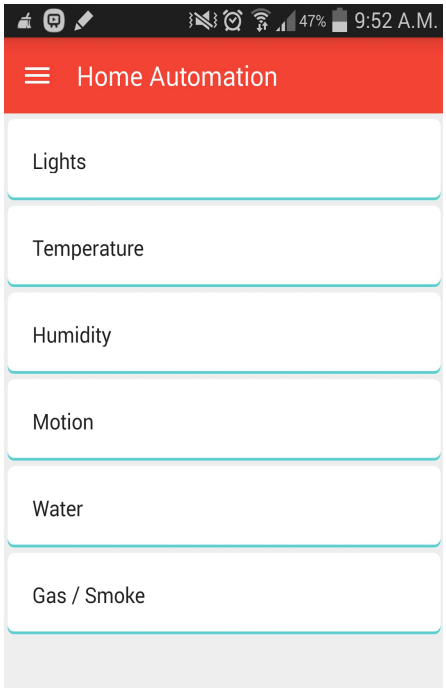
3.2 Software System Design

3.2.1 Android User Interface Design



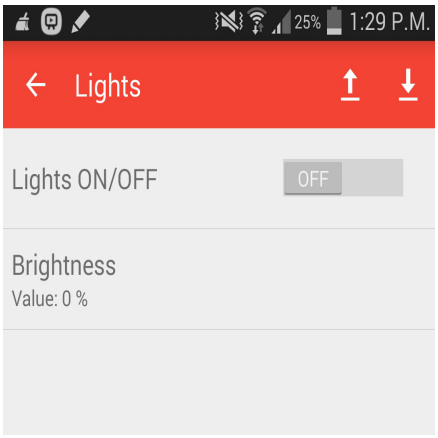
Splash screen

This is the splash screen, which appears as the Arduino Home Automation App is selected from the list of applications available in the android phone. This splash screen is held for pre defined time and then after the elapse of that time home screen appears on the screen.



Home Screen

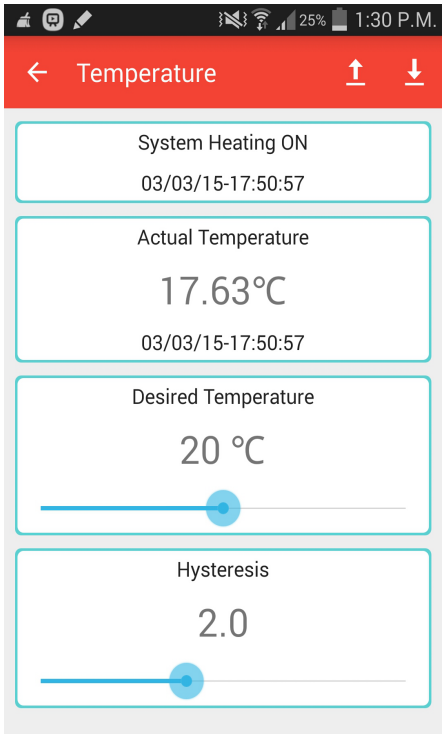
In the Home Screen list of sensors is shown. The user can make a selection from the list of sensors for which he needs to interact to move to the corresponding next screen.



Light

In case of light, configurations are with respect to switching on/off the light.

The brightness is displayed as a percentage, and can be adjusted by the user.



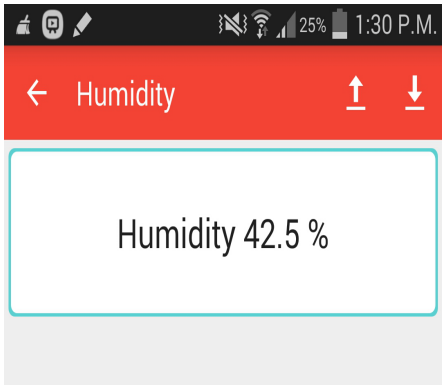
Temperature

The temperature screen shows the status of the system whether the heating relay is on, whether the cooling relay is on or whether the system is in ideal state, along with that it will show the time and date to when the status of the system was last updated on the interface.

Readings from temperature sensor is displayed under Actual Temperature along with date and time for that reading.

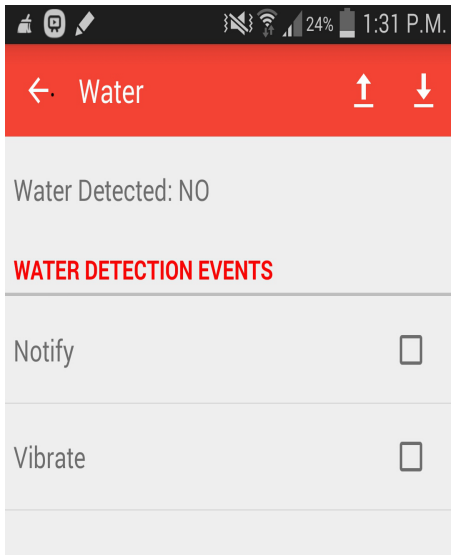
User can set his Desired temperature from the horizontal slider below.

Hysteresis value can also be specified from the horizontal slider below.



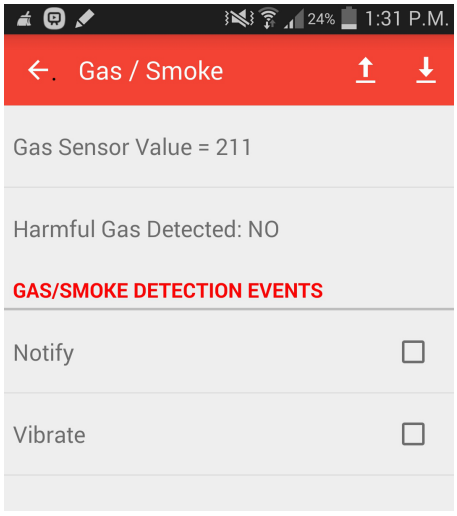
Humidity

The value of humidity measured as percentage from the remote humidity sensor is displayed in this screen.



Water

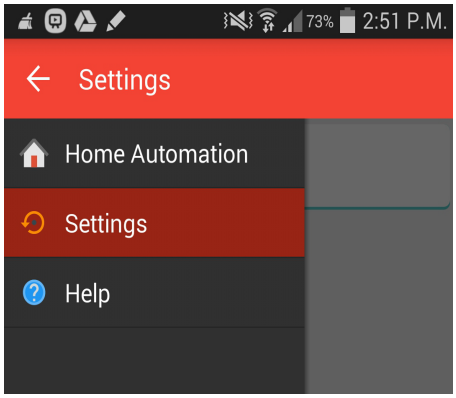
The indication of presence of water is displayed. The configuration of how to alert the user- via notification and/or vibration can be selected.



Gas/Smoke

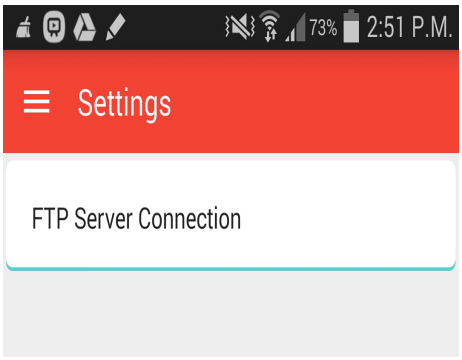
Value of gas level measured is displayed. Also, the detection of harmful gas is indicated.

The configuration of how to alert the user- via notification and/or vibration can be selected.



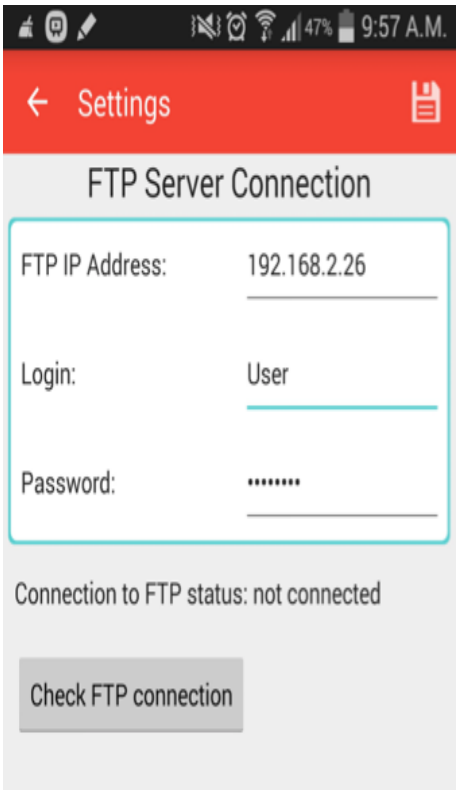
Settings

- To move to the home screen, press the "Home Automation" button.
- To change settings (eg. FTP server configurations), press the "Settings" button.
- For help on the usage of application, Press the "Help" button.



FTP Server Settings

Selecting FTP server connection takes to the next screen for configuring FTP server related information.

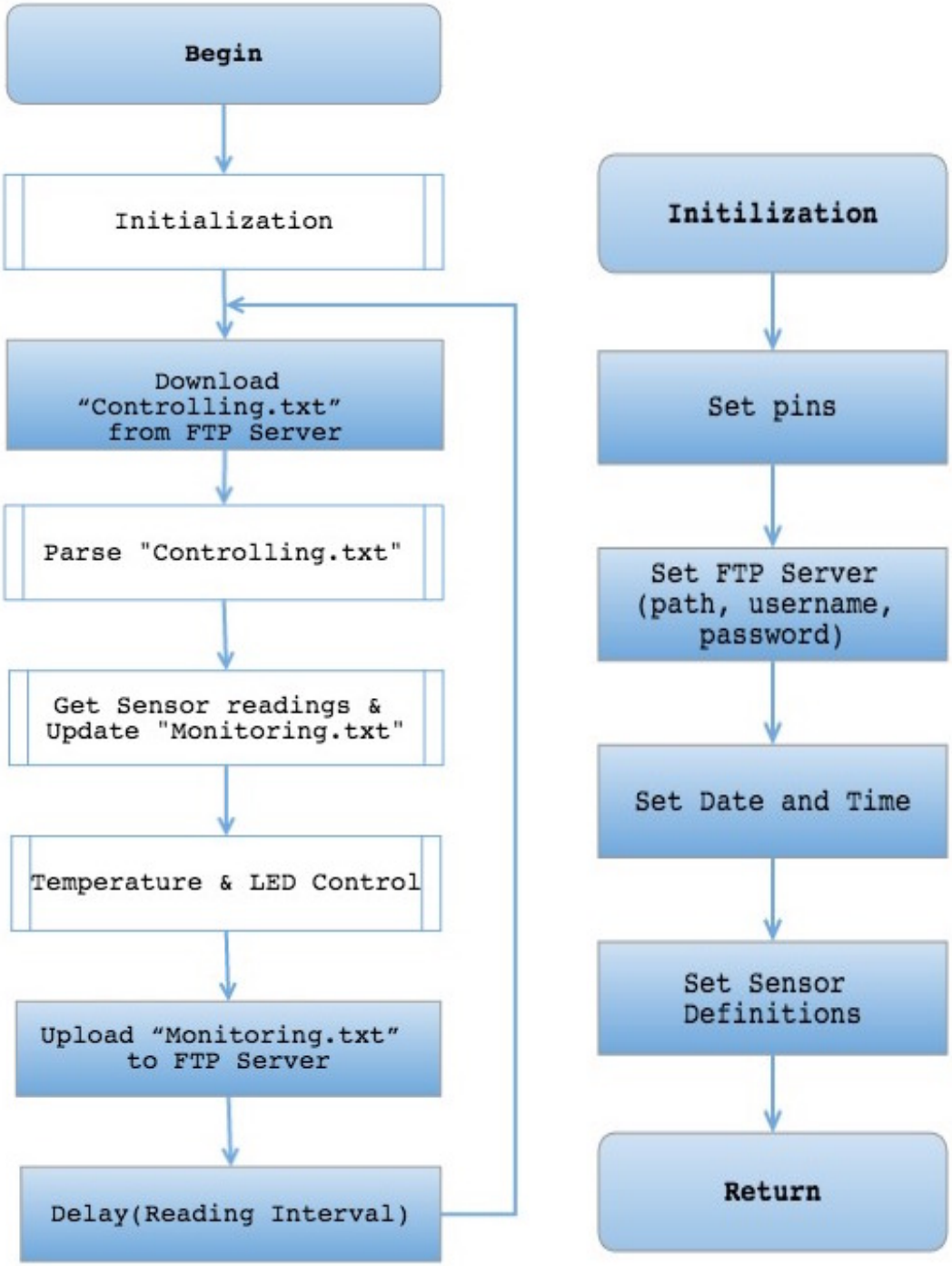


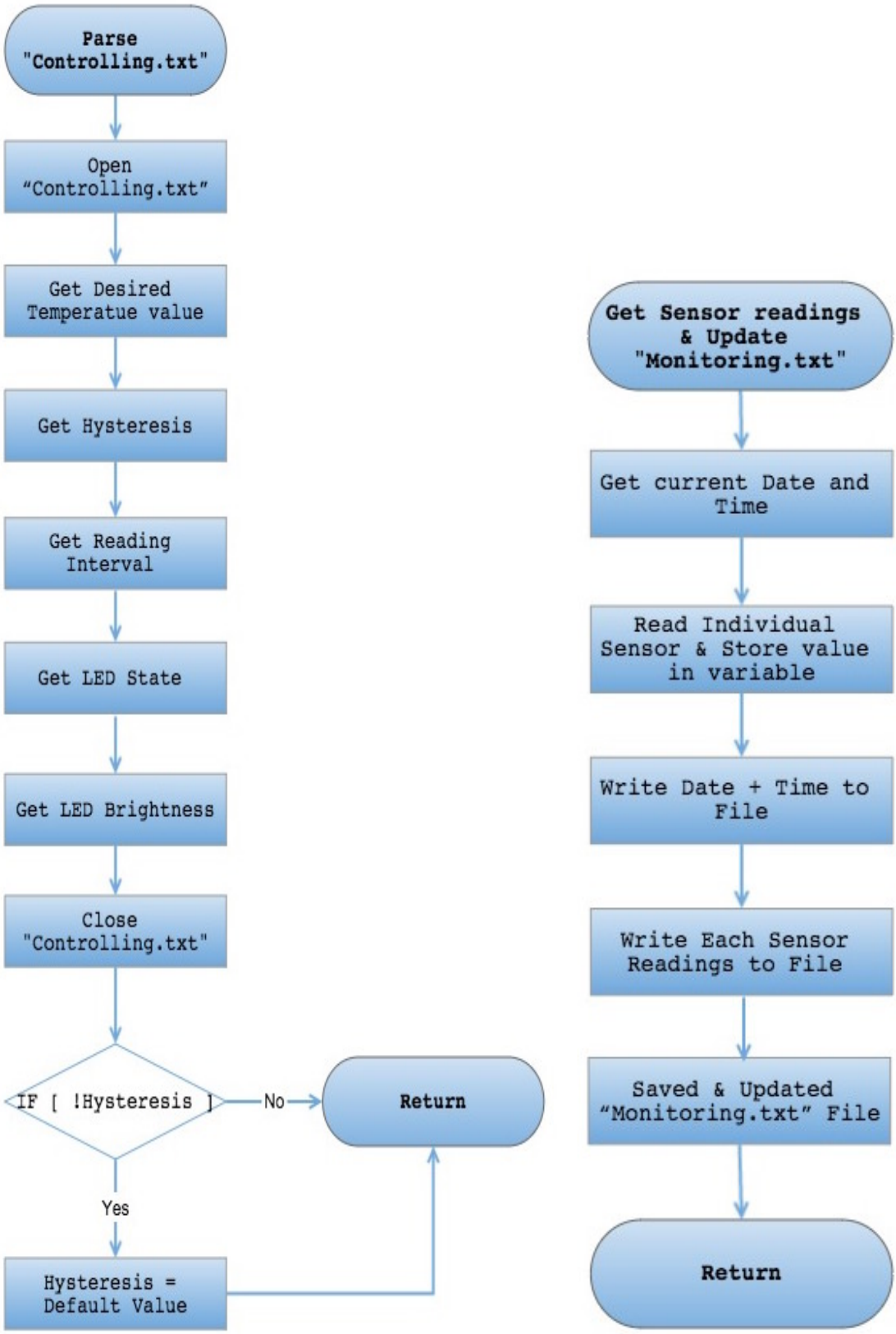
FTP Server Connection Settings

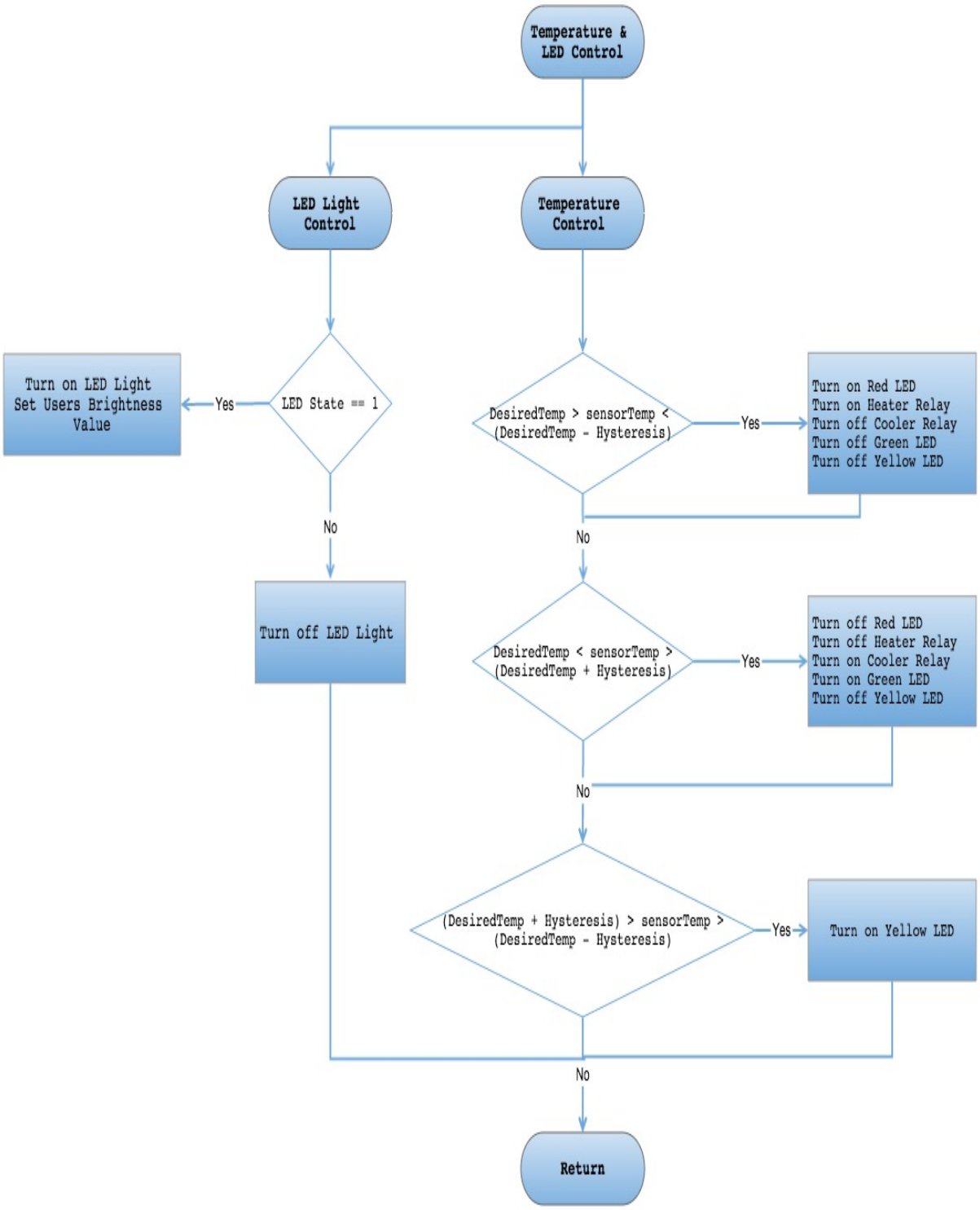
Details of the FTP server (IP address, login and password) are configured here by the user and when configured, pressing the save button at the top right will check and verify the FTP IP Address, Login and Password to make sure it is the correct information that is entered by the user. If the information is correct, it will connect to that FTP server, otherwise it will fail to connect due to incorrect information entered.

By pressing the Check FTP connection button, it will try to make the connection to FTP server that user has specified, if connection is successful it will update the FTP status to connected otherwise it will update to not connected.

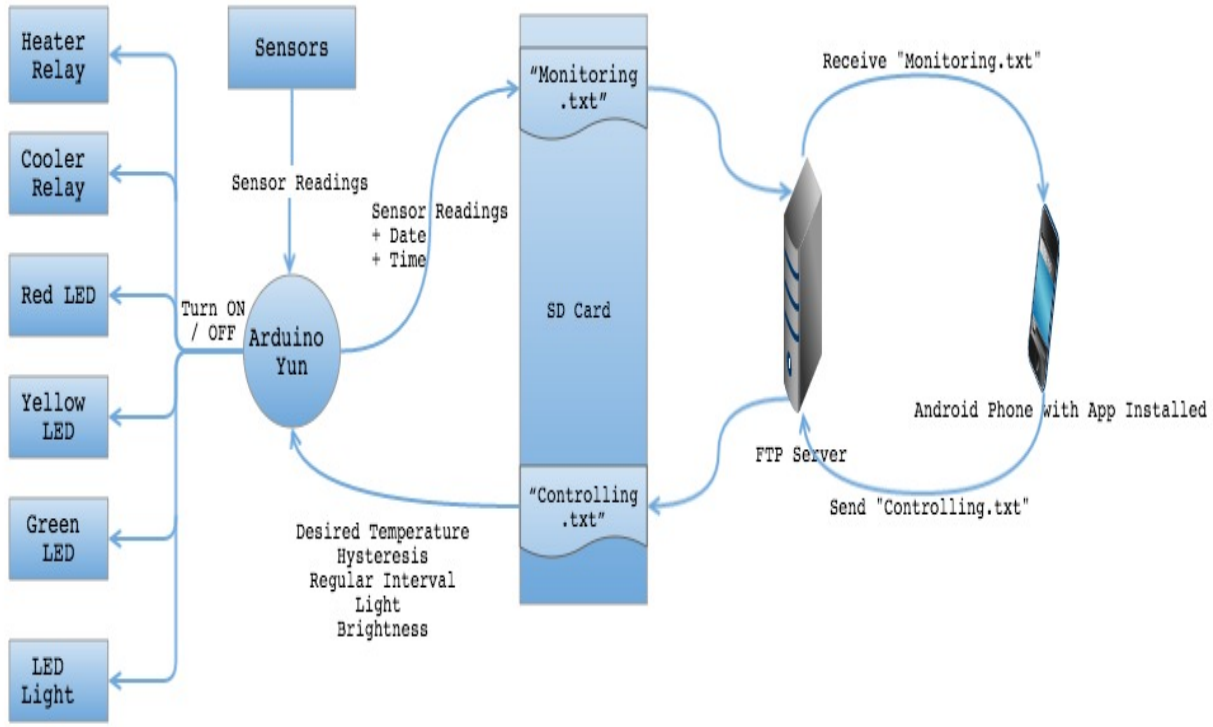
3.2.2 Arduino-Yun Software Design



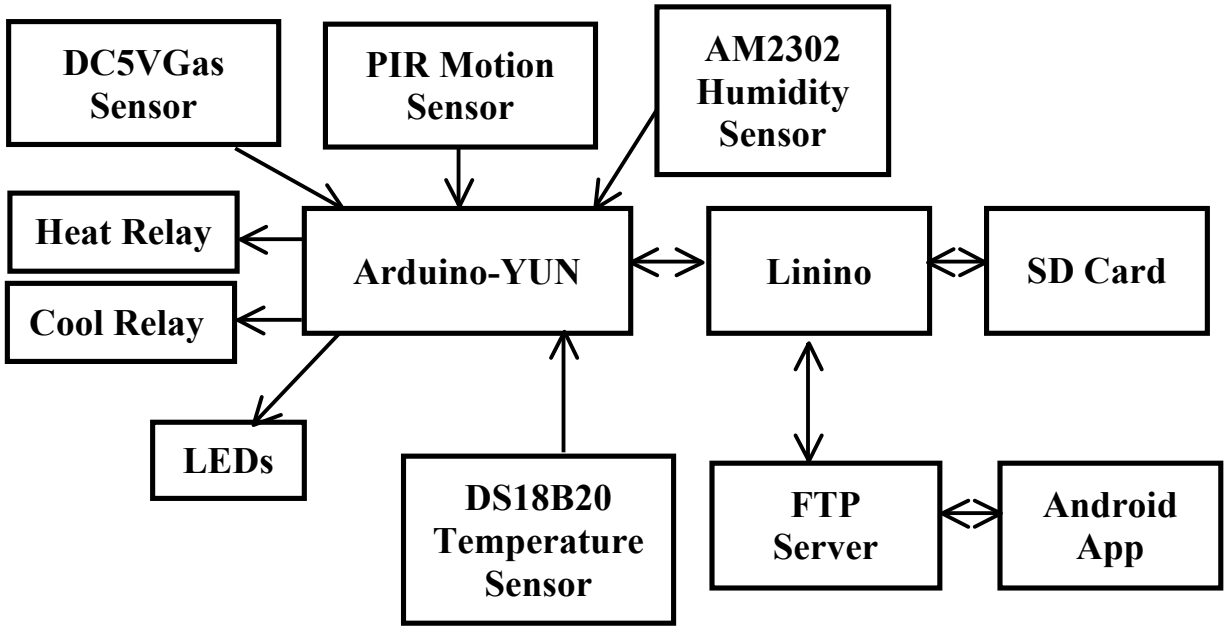

Parse "Controlling.txt"
Open "Controlling.txt"
Get Desired Temperatue value
Get Hysteresis
Get Reading Interval
Get LED State
Get LED Brightness
Close "Controlling.txt"
IF [!Hysteresis]
No
Return
Yes
Hysteresis = Default Value
Get Sensor readings & Update "Monitoring.txt"
Get current Date and Time
Read Individual Sensor & Store value in variable
Write Date + Time to File
Write Each Sensor Readings to File
Saved & Updated "Monitoring.txt" File
Return


Temperature & LED Control
LED Light Control
Temperature Control
LED State == 1
Yes
Turn on LED Light
Set Users Brightness Value
No
Turn off LED Light
DesiredTemp > sensorTemp < (DesiredTemp - Hysteresis)
Yes
Turn on Red LED
Turn on Heater Relay
Turn off Cooler Relay
Turn off Green LED
Turn off Yellow LED
No
DesiredTemp < sensorTemp > (DesiredTemp + Hysteresis)
Yes
Turn off Red LED
Turn off Heater Relay
Turn on Cooler Relay
Turn on Green LED
Turn off Yellow LED
No
(DesiredTemp + Hysteresis) > sensorTemp > (DesiredTemp - Hysteresis)
Yes
Turn on Yellow LED
No
Return

3.3 System Data Flow Diagram



3.4 System Design Diagram



3.5 Design Choices

3.5.1 Choice of Arduino Yun

The following table gives a brief comparison between Arduino-Yun and Raspberry Pi :

	Arduino Yun	Raspberry Pi
1	Created in Italy. A simple hardware prototyping tool for design students and Hobbyists.	Created in UK. Designed to be a cheap, hack able computer for improving tinkering skills
2	Microcontroller based physical computing platform.	Microprocessor based single-board computer (SBC)
3	Need to create a sketch in the Arduino language. This is not hard - it looks very C-like. There are a ton of libraries and classes for Arduino - so it is possible to make it do just about anything	Runs on Linux. You can program for it in C+, Java, python or some other language you may already be comfortable with
4	Supports one USB port and connect wirelessly to the Network. It's powerful enough to function as a hardware controller.	Supports two USB ports and connect wirelessly to the Internet. It's powerful enough to function as a PC
5	Simplicity makes it a much better bet for pure hardware projects	Superior to Arduino, but that's only when it comes to software applications
6	'Real-time' and 'analog' capability. This flexibility allows it to work with just about any kind of sensor or chips.	Not there. Not as flexible; for example, reading analog sensors requires extra hardware assistance
7	Thousands of tutorials on hooking an Arduino into just about every kind of part	Pi benefits from decades of Linux software
8	Arduino IDE is significantly easier to use than Linux. You can get an LED light to blink in just few lines of code. Since Arduino-Yun isn't designed to run an OS or a lot of software, you can just plug it in and get started	A program to blink an LED with Raspberry Pie, you'd need to install an operating system and some code libraries—and that's just to start.

9	Can leave an Arduino-Yun plugged in as it conducts a single process for a long time, and just unplug it when you're not using it.	Can multitask processes—it can run multiple programs in the background while activated
10	Simpler, harder to 'break' or 'damage' and has much more learning resources at this time for beginners	You have to learn some Linux as well as programming—such as Python
11	Works with any computer and can run off of a battery. You can also turn it on and off safely at any time	Pi setup can be damaged by unplugging it without a proper shutdown
12	Makes hardware projects very simple	Hardware projects are quite complex to develop comparing to Arduino.

The reason Arduino Yun was the best choice for this project was because it required interfacing with many hardware components such as sensors, relays and LED's. Arduino Yun had in built Wi-Fi-shield that enabled data communication between FTP server and on board Linino Linux processor, without the need for external hardware Wi-Fi-shield.
Raspberry Pi would be better suited for Internet-connectivity, as a mini computer that can play videos, music or send emails with ease and for projects that are related to more software then hardware.

3.5.2 Programming language for Arduino Yun

The reason for selecting C programming on Arduino-Yun motherboard is explained in the following paragraphs:
C/C++ provides a lot of advantages compared to other languages when working with Arduino-Yun. They are as follows:

1. It is small and easy to learn.

2. It is processor-independent because compilers exist for almost all processors in the world. This Independence provides something very useful to programmers: they can focus on algorithms and

the application levels of their job instead of thinking about the hardware level at each row of code.

3. It is a very "low-level" high-level language.

This is its main strength. C is a relatively "low level" language. It means that C deals with the same sort of objects that most computers do. These may be combined and moved about with the arithmetic and logical operators implemented by real machines.

Today, this is the only language that allows interacting with the underlying hardware engine so easily and this is the reason why the Arduino tool chain is based on C.
4. Almost all Arduino-Yun libraries are made using C/C++ programming in order to be easily reusable, which is one of the most important qualities in programming.

5. It is possible to port general C/C++ code directly to Arduino-Yun without difficulty.

6. If one has previous programming experience and aware of concepts such as OOP, it is better to go for C/C++. The Arduino-Yun C language is good for beginners.

Note: There isn't really an Arduino language as such. It's really just C/C++ with some domain-specific libraries. These add on various features, such as functions you can call to control the hardware. If you didn't have those functions, you'd need to work directly with special registers to control everything.

3.5.3 Choice for communication with other devices

FTP protocol used for communication

The **File Transfer Protocol** (**FTP**) is a standard network protocol used to transfer computer files from one host to another host over a Network, such as the Internet.

FTP is built on client-server architecture. It uses different connections for control and data between the client and the server. FTP users may authenticate themselves using a clear-text method (a username and password) but can be the server permits it). For secure transmission that protects the username and password, and encrypts the content, FTP is often secured with SSL/TLS (FTPS).

The first FTP client applications were command-line applications developed before operating systems had GUIs. Many FTP clients and other utilities have been developed for desktops, servers, mobile devices, and hardware. FTP has also been embedded into applications e.g., Web page editors.

The two modes of working of FTP is *active* or *passive* mode. The mode determines how the data connection is established. In both cases, the client creates a TCP control connection from a random unprivileged port N to the FTP server command port 21. In active modes, the client starts listening for incoming data connections on port N+1 from the server (the client sends the FTP command PORT N+1 to inform the server on which port it is listening). *Passive mode* may be used in such cases where the client is behind a firewall (and unable to accept incoming TCP connections).

The server responds over the control connection with three-digit status codes in ASCII with an optional text message. For example "200" (or "200 OK") means that the last command was successful. The numbers represent the code for the response and the optional text represents a human-readable

Data transfer can be done in any of three modes:

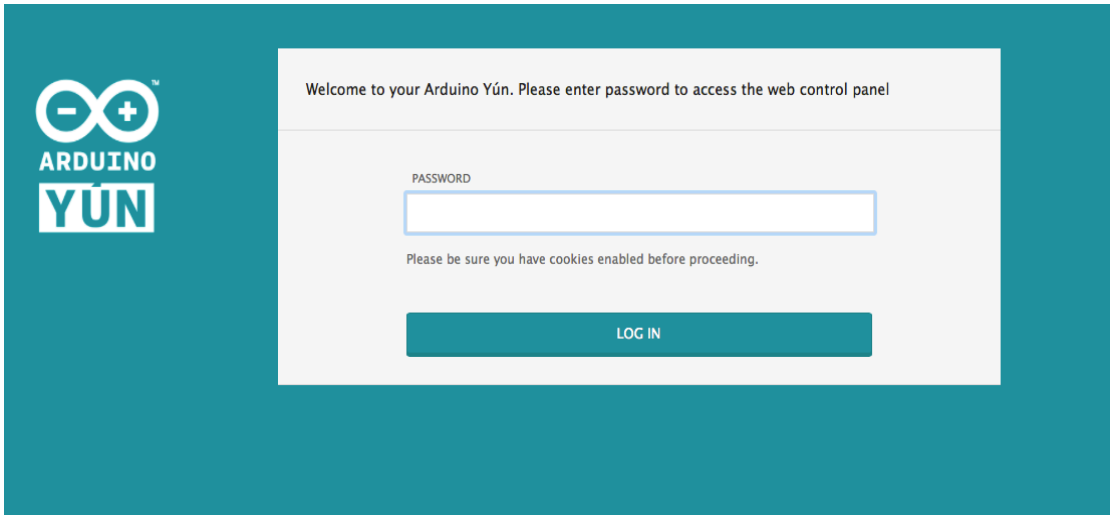
- Stream mode: Data is sent as a continuous stream, relieving FTP from doing any processing. Rather, all processing is left up to TCP. No End-of-file indicator is needed, unless the data is divided into records.
- Block mode: FTP breaks the data into several blocks (block header, byte count, and data field) and then passes it on to TCP
- Compressed mode: Data is compressed using a single algorithm (usually run-length encoding).

Login

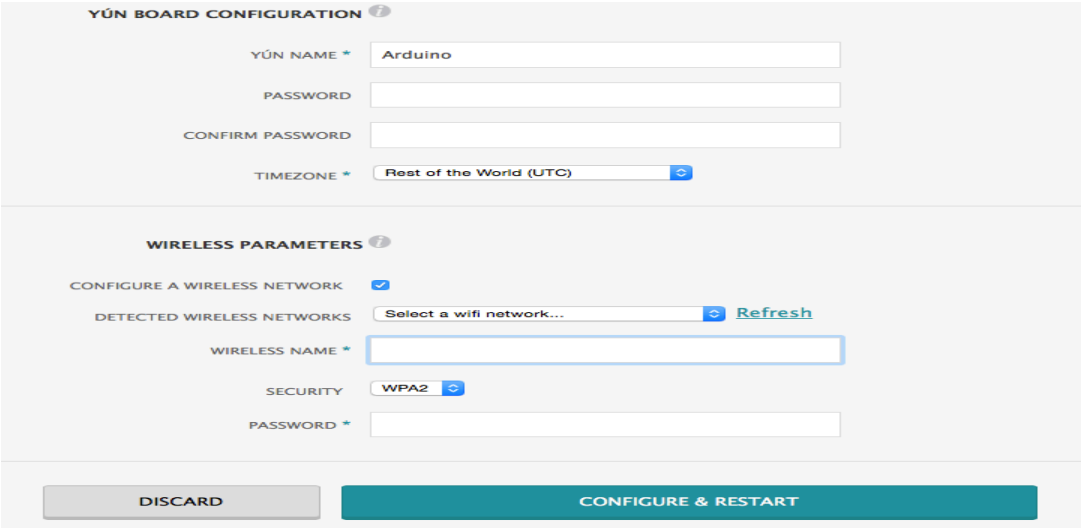
For granting access to users, FTP login utilizes a username and password scheme. The username is sent to the server using the USER command, whereas the password is sent with the PASS command. If the information provided by the client is acceptable to the server, the server responds with a greeting message, following which the session will commence.

Chapter 4 - Configuration and Implementation

The procedure of starting the system along with the necessary configurations is explained below:



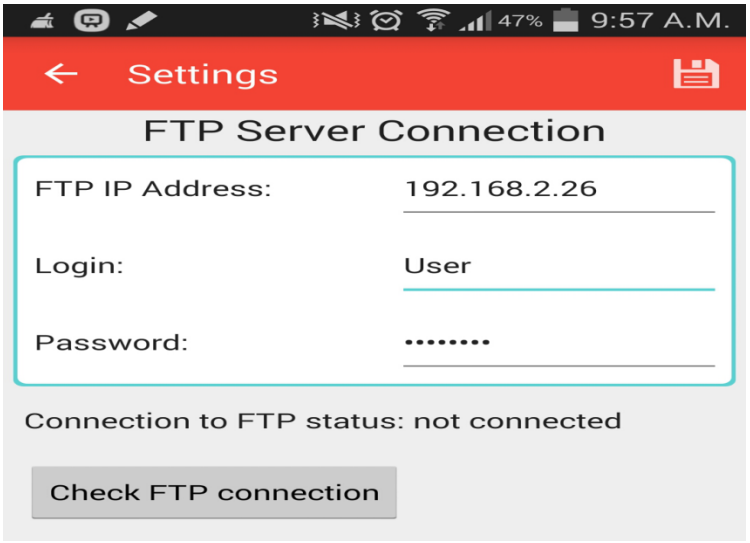
1) Connect Arduino Yun to your computer and search for Wi-Fi Networks. Select the Arduino Yun's Wi-Fi Network.
2) Once connected, go to the link http://arduino.local and it will direct you to the above screen. Enter the password into the corresponding field. Initially the password is "arduino" and press Log In button to gain access into web panel settings; user may change the password to the desired value once he logs into the web panel.



3) Once logged in, you will be directed into the Board's configuration page where it is prompted to select your wireless network and enter its security password.
   Provide these details and follow it up by pressing Configure & Restart.
4) Wait for a moment while Arduino Yun connects to the wireless network. The board will then be restarted with the applied configurations.
5) Arduino Yun now has an IP address on the Network, which can be viewed by logging into the web panel. Using the Linux command "ssh" (and this IP address), log into the Linino Linux processor.

```
//Server path, login and password Definitions
String FTP_SERVER_PATH="192.168.2.26/FTP/";
String login="UserName";
String pass="******";
```

6) In the project's sketch, locate the code (shown above) and change The IP address to your FTP server's IP address. Enter your username in the login string and password in the password string. It is required that a Folder labeled "FTP" be created in the FTP server's location, with read and write permissions enabled.
7) Now upload the sketch to the Arduino. The sensor results can be instantly viewed in the Serial Monitor.

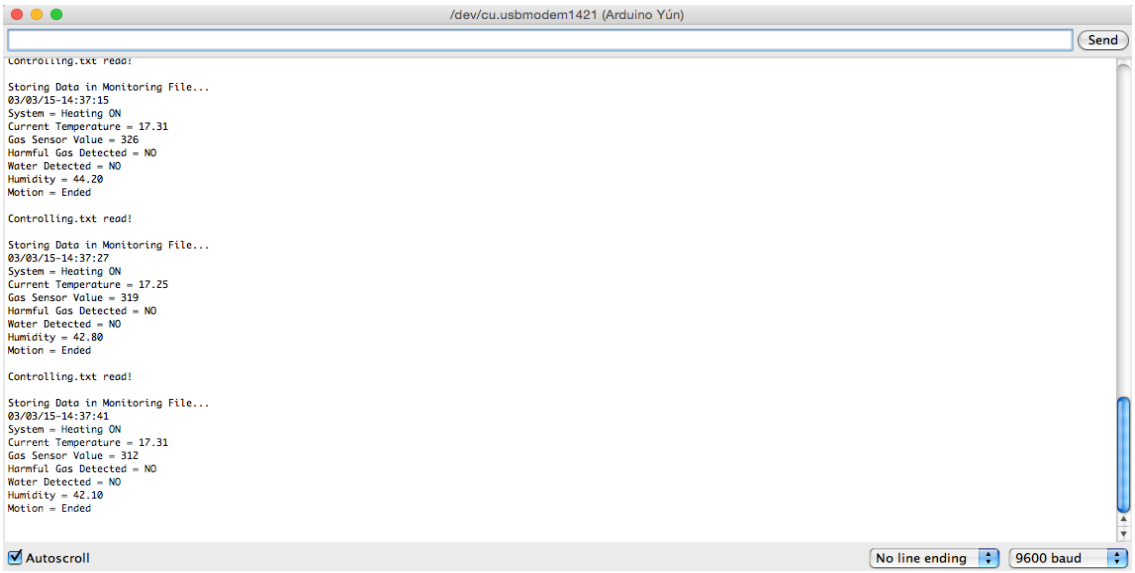


8) On the Android device, the user has to configure the IP address of the FTP server, Login and password. Click on the Save button at the top right hand side of the screen to save the

settings applied. The user can check the FTP connection by clicking on the button "Check FTP connection", whereupon he is notified if the app is connected to the FTP server or not.

NOTE: The code for the Arduino Sketch for implementation is provided in the Appendix Section.

Chapter 5 - Testing and Results



Serial Monitor Output of Arduino

The output from the Serial Monitor shows that the "Controlling.txt" file was successfully downloaded by the Linino Linux processor and stored in the SD Card. That file was parsed by the sketch uploaded to the Arduino and the values were compared to the temperature sensor readings to turn On/Off the heating and cooling relay, as well as to turn On/Off the LED light and adjust it's brightness according the values in the file.

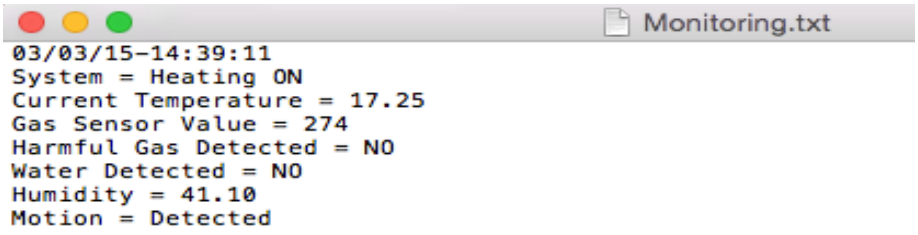
The output also shows that the sensor data along with date and time is being written to "Monitoring.txt" file and appended to "MonitoringHistory.txt" file in the SD Card at regular intervals. The Linux processor then uploads the "Monitoring.txt" and "MonitoringHistory.txt" files to the FTP server in the FTP directory of the server.

FTP Server: Controlling.txt File uploaded to FTP server by Android

The snapshot of the file above demonstrates the Controlling.txt file being successfully uploaded

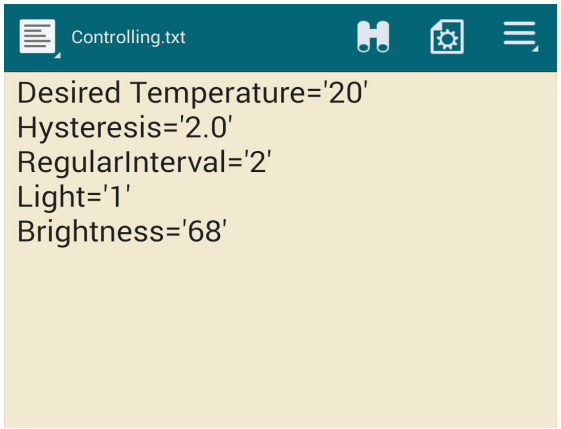
to the FTP server in the FTP directory. During testing, the time to upload the file from the Android device to the FTP server was approximately 2 – 4 seconds.

Carrying out tests on a better network speed, the time to upload the file was much less then normal.



FTP Server: Monitoring.txt File uploaded to FTP server by Arduino

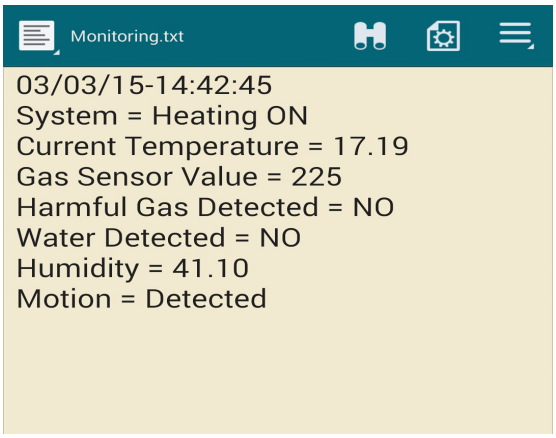
The picture of the Monitoring.txt file proves the file being uploaded from the SD Card to the FTP server. The Linino Linux processor handles this process of uploading the file to the FTP server. The time taken to upload the file during testing stage was roughly 3 – 6 seconds.



Controlling.txt File stored in Arduino directory on Android Device

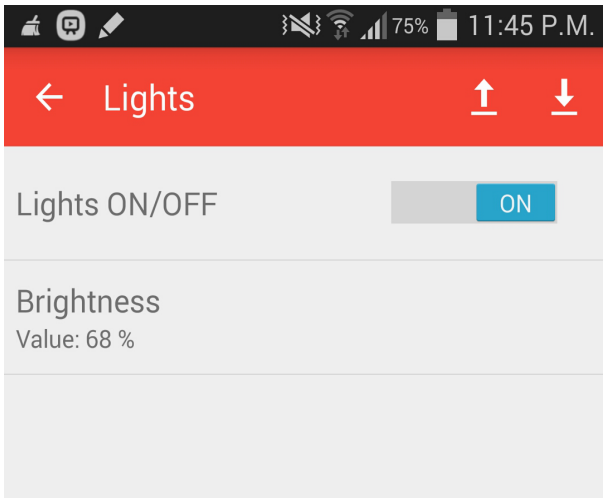
Whenever the user sets the values for desired temperature, hysteresis, light and brightness from the User Interface, it then gets directly reflected to this file by the press of the upload button at

the top right hand corner of the app. The format of the file remains the same with the exact same content, only the numerical values between the two apostrophes gets modified. This file will then get uploaded to the FTP server automatically without the user being notified. Once the user has pressed the upload button, the app will carry out two functions, one that is the storing the values in this file as shown above, the second which is the uploading of that file to the FTP server.



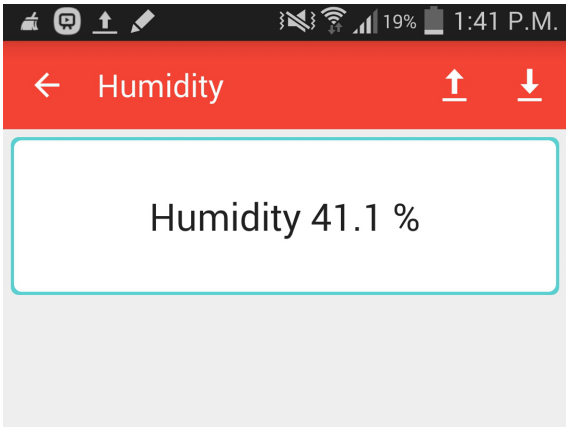
Monitoring.txt File downloaded to the Arduino directory on Android device from FTP Server

By the press of the download button at the top right hand corner of the app, it will download the Monitoring.txt file from the FTP server and store it in the Arduino directory on the Android device. The snapshot above shows the content of the file when the download button was pressed from the app. During the testing stage, the results in the downloaded Monitoring.txt file were not identical to the output results from the Arduino. The results were matching with the third last or the second last output results from the Arduino. The reason behind this is that there is a certain delay of around 4 – 6 seconds for the latest output results from Arduino to reach the Android app.



Testing of the Lights Functionality

Few tests were undertaken where light was turned off and on, and the brightness of the light was changed to few different values. On the hardware side, the results were satisfactory. The LED Light, which was connected to PWM pin 10 of Arduino was responding after 4 – 5 seconds when the state of the LED was changed from the app or when the value of the brightness was changed.

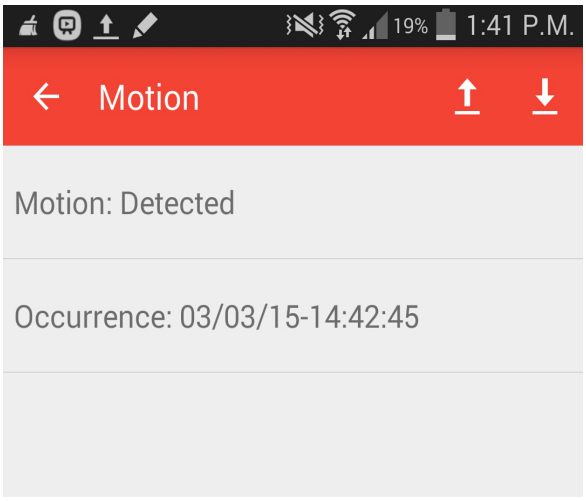


Testing the Humidity

Testing on the Humidity was performed, to check whether the app parsed the correct information from the Monitoring.txt file. The output from the humidity sensor is floating value as shown above in the snapshot, if it is expecting a floating point value in the file and that value got

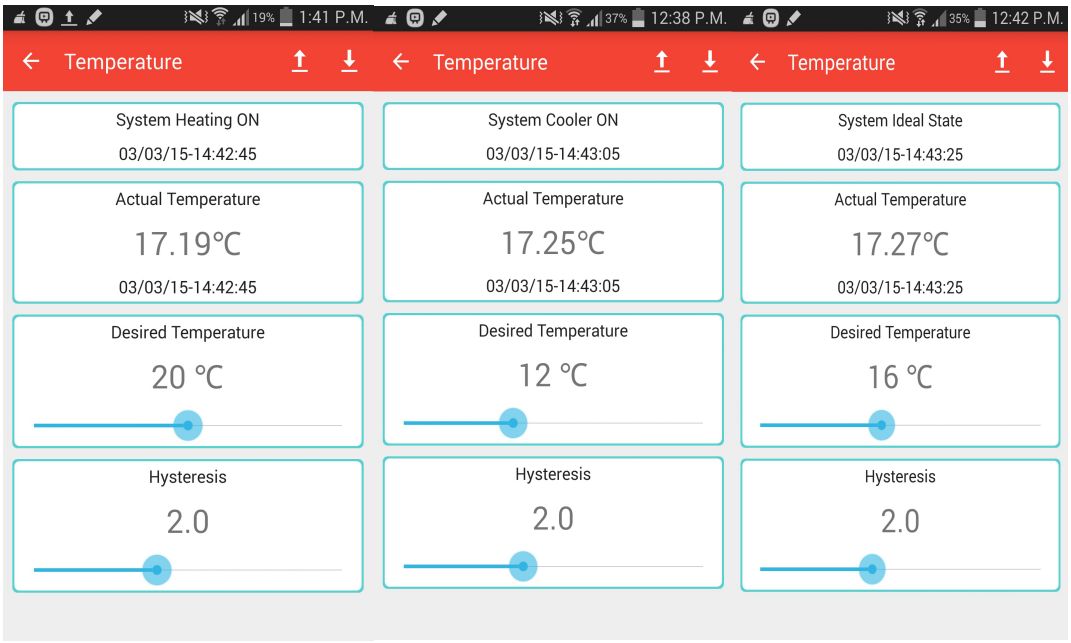
modified to a string or an integer then possibly the application would crash because it would not be able to parse the value due to different data type in the file.

There were few tests performed where Humidity value was modified from a float number to a string value in the Monitoring.txt file in the FYP server. The results from that testing was sudden crash of the application when the download button from the top right hand corner of the app was pressed. When the value of it got updated to the floating-point number then the app was successful in pressing the correct data type, which it was expecting, and therefor the application did not crash. The result for the gases shown above in the snapshot was taken when the Monitoring file had the floating-point number as it's data type.



Motion Detection Testing

The snapshot above shows the results, when testing for motion detection was carried out. The results show that Motion was and the time and date of the detection when it occurred. The result for motion can be either "Detected" or "Ended", ended result occurs when there is no motion detected. The application parses only string type from the Monitoring.txt file, if the data type was changed to integer or floating point number then the result could be the crash of the application.



(a) **(b)** **(c)**

Testing for Temperature Control System

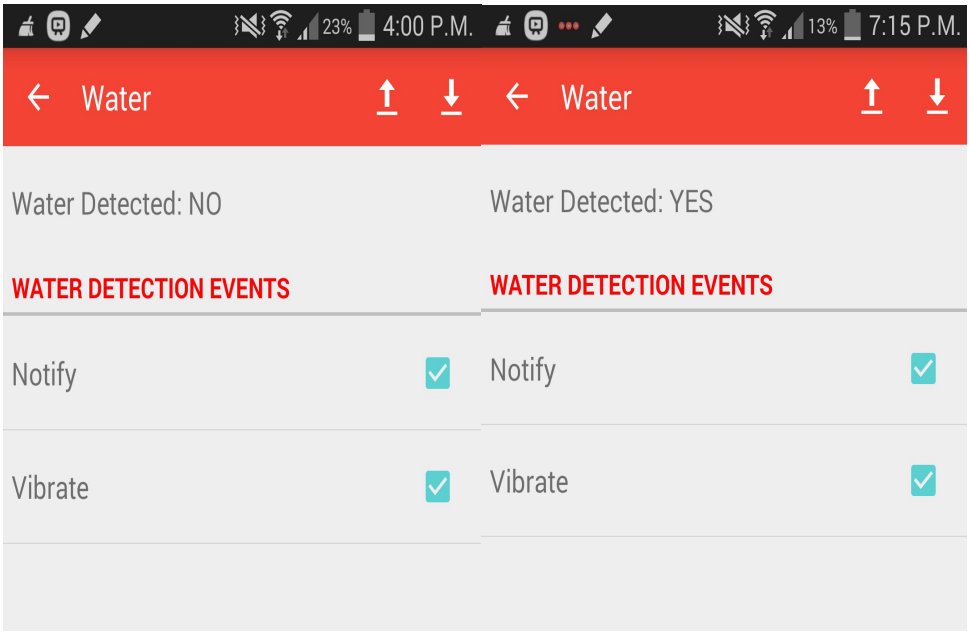
There were three testing scenarios carried out, the first one as the testing of the Heating Relay that meant that heating system was turned on. The second test was demonstrating the Cooler Relay was turned on, while temperature reached desired temperature plus the added hysteresis value. The third test was show the ideal state of the system, which meant that both the Riley's stopped its functioning as the system was in position.

The first snapshot (a) demonstrates the functioning of the Heating system. It shows when the actual temperature is less then the desired temperature minus the hysteresis value then an electrical heating system is to carry out its work. On the hardware side the red LED light, which is connected to pin number four demonstrates the turning on of the Heating Relay. It is assumed that if the Heating Reley were to be connected to a system, then it would perform as expected.

Snapshot (b) illustrates the operating of the Cooler system. When the user wishes to select desired temperature to be less then the actual temperature, then automatically the system understands to turn on the Cooling Relay. Then time when actual temperature reaches the desired temperature plus hystereisis value then the relay is turned off. The green LED light, which is connected to pin number five is used for demonstrating that the Cooling Relay is turn on. It is

again assumed that if the Cooling Relay were to be connected to a real electrical cooling system, then it would perform as anticipated.

The last snapshot (c) shows the system in an ideal state, which means that the actual temperature is less then the desired temperature plus the hysteresis value and it's greater then the desired temperature minus the hysteresis value. It demonstrates that the actual temperature is close to desired temperature; Heating and cooling system during that time are in their off state. The yellow LED light, which is connected to pin number six is used for the purpose of proving that the system is in ideal state and relays are in their off state.

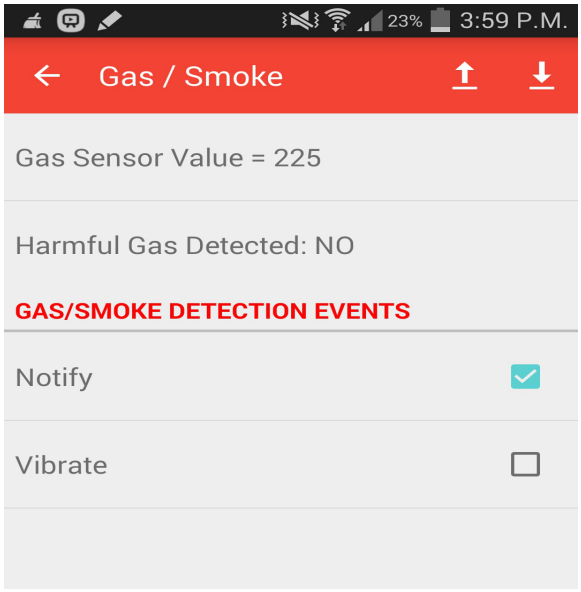


(a) **(b)**

Testing for Water Detection Functionality

Testing for water detection was executed and whenever the two jumper wires for detecting the water was dipped into tap water, current was flown through the tap water and the result was the detection of water. The current through water can only flow if there are ions existing in it, which will be free to move in water, the tap water has this property and this makes it a good conductor of electricity.

The time when water was detected, it was then registered as YES in the monitoring file otherwise it stored as NO. As it can be seen from the snapshot (a) above, the water was not detected during that test. The snapshot (b) was the taken to detect and display water detected on the screen, also in that test the Notify and Vibrate functionalities were tested and as expected they were performing as desired. The Notify and Vibrate box was ticked before the test (b), and when the results were shown in that screen (b) along with that it gave notification in the notification bar at the top of the screen and the mobile device also vibrated.



Testing for Gas/Smoke Functionality

The snapshot above was the test undertaken for the Gas/Smoke functionality. The results show the Gas sensor value parsed from the Monitoring.txt file and below it, indicates whether harmful gas was detected or not. The notify and vibrate functionality was working as expected when the harmful gas detected value parsed from the Monitoring.txt file was “YES”.

Discussion of Results

Each of the sensors was tested thoroughly. The test results were based on Android screen views, results indicated from LEDs, viewing logs in Controller.txt, Monitoring.txt files and serial monitor logs. The above section with elaborated test results for each sensor shows that performance is as per design.

The system was tested for stability by switching on for a prolonged period of time and the results were consistent when tested for repeatability. Also, there was a match between the information on files on the Android app and Arduino logs. Thus, we are also able to track history accurately.

Hysteresis performance was verified by changing the desired temperature continuously (cooling and heating) and observing the effects of this on switching ON/OFF the heater / cooler relays. This also shows that the concept of "Hysteresis" has been implemented successfully.

The project is planned as prototype and home automation "proof of concept" on a smaller scale. FTP is used as protocol for communications with various devices. For more real time results, an enhanced design with SNMP protocol and events triggered through interrupts is suggested.

As planned to work on the limitation that FTP will lead to a delay in transferring the files, the regressive load testing is not achievable. To achieve consistency in recording the time of failures on the Android screen and occurrence of failure, it is required to create errors sequentially. If values change at a faster rate, there might be a lack of synchronization between values on display and actual values (older values will be recorded).

Conclusion

The objective to control devices at home using an Android phone and an Arduino microcontroller was achieved in this project. It is designed to sense parameters like temperature, presence of gases, water, motion and humidity, and automate to switch on/off devices like heaters, coolers and LED lights.

The performance results, with android screenshots, configuration and monitoring.txt files along with LED indications on the breadboard all prove that the sensors are controlled as designed. Thus, Arduino has proven to be an effective microcontroller for monitoring different sensors and controlling various devices at home.

I would like to conclude that there could be enhancements made to this project. Further Recommendations discusses the recommendations for future improvements.

The project has helped me gain an understanding of the concepts involved with interfacing software and hardware with a microcontroller and communicate with an Android phone using FTP protocol. This project has helped me to update my Android Java and Arduino C skills. Also, I have learnt about embedded programming and required Linux commands. There has also been an addition of knowledge in other areas such as choice of sensors, microcontroller and programming languages to be used.

Further Recommendations

The project was also conducted with web camera based streaming and voice recording on voice detection. Due to the inherent limitations on memory, I had to eliminate these two functionalities from this project. Hence, it is recommended to add these applications with an enhanced version of Arduino.

A study of other products available in the market has been undertaken to review additional features offered by them. Based on the table below, it is proposed that the current project can be enhanced with extra functionalities mentioned in the table.

No.	Similar project	Remarks & Additional features supported
1.	Arduino-Yun Smart Home Automation App (Amphan)	a. Web server based GUI (in addition to app - Arduino-YunGUI) b. Relays to control on/off of home appliances
2.	Souliss App	Automate group actions and commands. Additional feature: Light colors/brightness of led strips can be controlled by music/led controls
3.	Smartphone solutions	Able to control doors, garage doors, solar systems, domestic hot water system
4.	Eight control Arduino-Yun	AES encrypted port forwarding possible
5.	Home control pro	In English and German languages
6.	AHA(Automatic home automation)	Allows time based schedules to automatically turn on/off devices at defined times Define location based schedules to turn on/off devices when you are away from home
7.	IZIG home automation	WiFi ZigBee wireless technology RGB moody lighting, AC - geyser drivers, curtain controllers

References

The following is a list of resources that I have benefited from during my learning and developing stage of the project:

[1] Julien Bayle, 2013, "C Programming for Arduino", Packt Publishing

[2] Jack Purdum, 2012, "Beginning C for Arduino: Learn C Programming for the Arduino (Technology in Action)", 1st Edition: Apress

[3] Marco Schwartz, 2014, "Arduino Networking", Packt Publishing

[4] Marco Schwartz, 2014, "Internet of Things with the Arduino Yún", Packt Publishing

[5] Michael Margolis, "Arduino Cookbook", 2nd ed: O'Reilly Media

[6] "AMN Design Manual.pdf"

Other Sources Used

http://arduino.cc/en/Reference/YunBridgeLibrary

http://www.hacktronics.com/Tutorials/arduino-1-wire-tutorial.html

http://arduino.cc/en/Main/ArduinoBoardYun?from=Products.ArduinoYUN

http://www.vernier.com/products/sensors/motion-detectors/md-btd/

http://www2.vernier.com/booklets/md.pdf

Appendix

Arduino-Yun Code

```
/*

Name: Pritpal Sahota

ID: 11142375

Description: Arduino-Yun sketch for FTP Remote Control / Monitoring of Home

In this sketch the following sensors are monitored: Temperature, Humidity, Motion, Gas and
Water detector

The monitoring information of these sensors along with Date + Time are stored in monitoring.txt
and the file is uploaded to FTP server.

The following devices are controlled: Relays (controlled by Arduino by comparing Actual vs.
Desired Temperature) through user configured controlling.txt file

 LED light (controlled by the user through controlling.txt file)

*/

#include <Process.h>

#include <FileIO.h>

#include <OneWire.h>

#include <DallasTemperature.h>

#include "DHT.h"


/*Digital Pins*/

//LED's -> Digital Pins 4,5,6

int RedLed = 4;

int GreenLed = 5;

int YellowLed = 6;
```

```
//Relay -> Digital Pin 7  --> Heating System
int IN1=7;
//Relay -> Digital Pin 13 --> Cooling System
int IN2=13;



//Humidity Sensor Definitions
/*Defining the Pin for Humidity Sensor*/
#define DHTPIN 8     // what pin we're connected to

//#define DHTTYPE DHT22 Sensor
#define DHTTYPE DHT22

// Initialize DHT sensor for Arduino Yun
DHT dht(DHTPIN, DHTTYPE);

//PIR Motion Sensor Definitions
int inputPirPin = 12; // Input pin (for PIR sensor)
int pirVal = 0; // variable for reading the PIR Sensor pin status

//Gas sensor Definitions
const int analogInPin = A0;  // Analog input pin that the potentiometer (Gas Sensor) is attached to
int sensorValueGas = 0;      // value read from the sensor
```

```
//Water Detection Definitions
const int inputWaterPin = 2;  // sets the digital pin as input for Water Detection
int sensorValueWater = 0;  //value read from the water detection circuit

// Light (LED) Definition
int led = 10;  //Defining pin value connected to LED light

/*DS18B20 Temperature Sensor Definitions*/
// Data wire is plugged into pin 3 on the Arduino
#define ONE_WIRE_BUS 3
// Setup a oneWire instance to communicate with any OneWire devices
OneWire oneWire(ONE_WIRE_BUS);
// Pass our oneWire reference to Dallas Temperature.
DallasTemperature temp(&oneWire);
// Assign the addresses of your 1-Wire temp sensors.
DeviceAddress Thermometer = { 0x28, 0x0E, 0xF1, 0xCF, 0x05, 0x00, 0x00, 0xF0 };

//Server path, login and password Definitions
String FTP_SERVER_PATH="192.168.1.22/FTP/";
String login="Gugz";
String pass="gugz";

void setup() {
  /*Configure the Digital Pins as Inputs and Outputs*/
  pinMode(RedLed, OUTPUT);
```

```
pinMode(GreenLed, OUTPUT);
pinMode(YellowLed, OUTPUT);
pinMode(IN1, OUTPUT);
pinMode(IN2, OUTPUT);
pinMode(led, INPUT);
pinMode(inputWaterPin, INPUT);
pinMode(inputPirPin, INPUT);

// Initialize the Bridge and the Serial
Bridge.begin();
Serial.begin(9600);

//Initializes the SD card and FileIO class
FileSystem.begin();

//Begin operation of Humidity Sensor
dht.begin();

// Start up the Temperature sensor library
temp.begin();
// set the resolution to 10 bit
temp.setResolution(Thermometer, 10);

//Setting the correct date and time from the Linux processor via Bridge
```

```
  Process time;  //Created a process time that will Use Bridge library's Process class to run Linux process
  time.runShellCommand("ntpd -qn -p 0.ie.pool.ntp.org");
  time.close();

  while (!Serial); // wait for Serial port to connect.
  Serial.println("Arduino Yun Project Setting Up...\n");
}

void loop () {

  //Download Controlling.txt from the FTP Server and Upload MonitorFile.txt to the FTP Server
  Process p;
  p.runShellCommand("curl ftp://"+FTP_SERVER_PATH+"Controlling.txt --user "+login+":"+pass+" -o /mnt/sda1/Controlling.txt");
  p.runShellCommand("curl -T /mnt/sda1/MonitorFile.txt ftp://"+FTP_SERVER_PATH+"Monitoring.txt --user "+login+":"+pass);
  p.runShellCommand("curl -T /mmt/sda1/MonitorFileHistory.txt ftp://"+FTP_SERVER_PATH+"MonitoringHistory.txt --user "+login+":"+pass);

  // Read the commands in the Commands.txt file located in the SD Card
  File Commands = FileSystem.open("/mnt/sda1/Controlling.txt", FILE_READ);

  // Positions of the values in the Commands.txt file in these string variables
  String Temp, Hys, RI,LI,Brightness;
  int init=0;
```

```
  // If the file is available, read from it and parse integers from it and store in corresponding
variables
  int first=0;
  if (Commands) {


    while (Commands.available()>0) {
      char c = Commands.read();
      if (c=='\'')
        init++;
      if (init==1) {
        if (first==1) {
          Temp+=c;
        } else
          first=1;
      }
      if (init==3) {
        if (first==0) {
          Hys+=c;
        } else
          first=0;
      }
      if (init==5) {
        if (first==1) {
          RI+=c;
        } else
```

```
      first=1;
    }
    if (init==7) {
      if (first==0) {
        LI+=c;
      } else
        first=0;
    }
    if (init==9) {
      if (first==1) {
        Brightness+=c;
      } else
        first=1;
    }
  }

  Serial.println("Controlling.txt read!");
  Serial.println("");  //blank line after the previous print output
  Commands.close();
}

// If the file isn't open, pop up an error:
else {
  Serial.println("Error opening Controlling.txt");
}
```

```
//Define a type conversion buffer and default value for hysteresis
char floatbuf[10];
float Default=0.3;

//Convert String Commands to float and int
Temp.toCharArray(floatbuf, sizeof(floatbuf));
float DesireTemp = atof(floatbuf);
Hys.toCharArray(floatbuf, sizeof(floatbuf));
float Hysteresis = atof(floatbuf);
int ReadingInterval = RI.toInt();
int LED_Status = LI.toInt();
int BrightnessValuePercent = Brightness.toInt();


//If the user doesn't introduce the hysteresis value, the hysteresis takes a default value
if (!Hysteresis)
  Hysteresis=Default;

//If value parsed from the controlling.txt file is "1" then turn the LED light ON
//Adjust the brightness of LED light from specified value
if (LED_Status == 1)  {

  digitalWrite(led,HIGH);
```

```
  //convert the percentage of brighness value to value between 0 - 255
  int BrightnessValue = (255/100)*(BrightnessValuePercent);

  // Set the brightness of LED speceficed by user
  analogWrite(led, BrightnessValue);
 }
 else {
  //Other wise if the value is not "1" then LED light is turned OFF
  digitalWrite(led,LOW);
 }
```

// Read Temperature sensor, Gas Sensor, Humidity Sensor, Motion Sensor, Water Detection and save the data in the MonitorFile in the SD Card

```
 // Strings are created for assembling the data to log MonitorFile
 String TimeStamp;  //Stores Time Stamp values in this format: mm/dd/yy-hh:mm:ss
 String System_Info;
 String TempString;
 String GasString;
 String WaterString;
 String HumidityString;
 String MotionString;

 TimeStamp += getTimeStamp();  //Writes value returned from getTimeStamp() function to TimeStamp
```

```
System_Info += "System = ";
TempString += "Current Temperature = ";
GasString += Gas();  //Writes value returned from Gas() function to GasString
WaterString += Water();
HumidityString += Humidity();
MotionString += "Motion = ";
MotionString += Motion();



//Read the Temperature sensor through 1-Wire Protocol
temp.requestTemperatures();
float sensorTemp = temp.getTempC(Thermometer);
if (sensorTemp == -126.00) {
  Serial.print("Error getting temperature");
}
TempString += sensorTemp; //add Temperature value to TempString String


//Implement the Temperature Control
// Heating Part of the Control
if (sensorTemp < (DesireTemp - Hysteresis)) {
  // When Sensor Temp. is less then (Desired Temp. - Hysteresis) OR less then Desired Temp. --
> Heating Relay turned ON (Heating ON) (Red LED ON)
  digitalWrite(RedLed, HIGH);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
```

```
  digitalWrite(GreenLed, LOW);
  digitalWrite(YellowLed, LOW);
  System_Info += "Heating ON";
 }

 // Cooling Part of the Control
 if (sensorTemp > (DesireTemp + Hysteresis)) {
  // When Sensor Temp. reached above (Desired Temp. + Hysteresis) OR reached above Desired Temp. --> Cooling Relay turned ON (Cooler ON) (Green LED ON)
  digitalWrite(GreenLed, HIGH);
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(RedLed, LOW);
  digitalWrite(YellowLed, LOW);
  System_Info += "Cooler ON";
 }

 //Ideal State
 if (sensorTemp > (DesireTemp - Hysteresis) && sensorTemp < (DesireTemp + Hysteresis)) {
  //If Sensor Temp. is between these values --> System is in Ideal state (Sensor Temp. ~ Desired Temp.) (Yellow LED ON)
  digitalWrite(YellowLed, HIGH);
  digitalWrite(RedLed, LOW);
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(GreenLed, LOW);
```

```
  System_Info += "Ideal State";
 }


 // Open the file. Note that only one file can be open at a time,
 // The FileSystem card is mounted at the following "/mnt/FileSystem"
 File MonitorFileHistory = FileSystem.open("/mnt/sda1/MonitorFileHistory.txt",
FILE_APPEND);  //Monitoring File where updated data is appended to the File
 File MonitorFile = FileSystem.open("/mnt/sda1/MonitorFile.txt", FILE_WRITE);  //Monitoring
File where it only writes the updated data


  // If the file is available, write to it:
 if (MonitorFile) {
  MonitorFile.println(TimeStamp);
  MonitorFile.println(System_Info);
  MonitorFile.println(TempString);
  MonitorFile.println(GasString);
  MonitorFile.println(WaterString);
  MonitorFile.println(HumidityString);
  MonitorFile.println(MotionString);
  MonitorFile.close();
 }
 // If the file isn't open, pop up an error
 else {
  Serial.println("Error opening MonitorFile.txt");
 }
```

```
// If the file is available, write to it:
if (MonitorFileHistory) {
  MonitorFileHistory.println(TimeStamp);
  MonitorFileHistory.println(System_Info);
  MonitorFileHistory.println(TempString);
  MonitorFileHistory.println(GasString);
  MonitorFileHistory.println(WaterString);
  MonitorFileHistory.println(HumidityString);
  MonitorFileHistory.println(MotionString);
  MonitorFileHistory.println(" ");
  MonitorFileHistory.close();
}
// If the file isn't open, pop up an error
else {
  Serial.println("Error opening MonitorFileHistory.txt");
}

   // Print to the serial port too:
  Serial.println("Storing Data in Monitoring File...");
  Serial.println(TimeStamp);
  Serial.println(System_Info);
  Serial.println(TempString);
  Serial.println(GasString);
```

```
    Serial.println(WaterString);
    Serial.println(HumidityString);
    Serial.println(MotionString);
    Serial.println("");  //blank line after the previous print output

  delay(ReadingInterval*10); //Regular Interval Time
}

// This function returns a string with the time stamp
String getTimeStamp() {
  String result;
  Process time;
  // Date is a command line utility to get the date and the time
  // in different formats depending on the additional parameter
  time.begin("date");
  time.addParameter("+%D-%T");  // parameters: D for the complete date mm/dd/yy and T for the time hh:mm:ss
  time.run();  // run the command

  // Read the output of the command
  while (time.available() > 0) {
    char c = time.read();
    if (c != '\n')
     result += c;
  }
```

```
 return result;
}

String Gas(){

 String GasValue;  //Defining String to store the value of int variable sensorValueGas to String
 String HarmFulGas;
 String Gas;  //String that get's returend from the function at the end

 // Read the analog value from Gas sensor Pin
  sensorValueGas = analogRead(analogInPin);
  // Determine status of analoge value from the sensor
  if (sensorValueGas >= 750)
  {
    HarmFulGas += "YES";   //stores this string indicating Harmful Gas is detected
  }
  else
  {
    HarmFulGas += "NO";  //stores this string indicating Harmful Gas is not detected
  }

  //Convert the int value of sensorValueGas to String and store all the data gathered in one string
  GasValue += String(sensorValueGas);
  Gas += "Gas Sensor Value = "+GasValue+'\n'+"Harmful Gas Detected = "+HarmFulGas;
```

```
  // Wait 10 milliseconds before the next loop
  // for the analog-to-digital converter to settle after the last reading
  delay(10);

  //return the Gas string from this function
  return Gas;

}

String Water(){

  String WaterDetect;  //String that get's returend from the function, which tells whether Water was detected or not

  // Read value from inputWaterPin on arduino
  sensorValueWater = digitalRead(inputWaterPin);

  //If the water was detected
  if (sensorValueWater == 1)
  {
    WaterDetect += "Water Detected = YES";  //write to WaterDetect string
  }
  // If the water was not detected
  else
```

```
  {
   WaterDetect += "Water Detected = NO";  //write to WaterDetect string
  }

  delay(10);  // Delay before next digitalRead

  //return WaterDetect string from this function
  return WaterDetect;

}

String Humidity(){

  String HumidityValue;  //Defining String to store the int value from the humidity sensor as
String
  String Humidity;  //String that get's returend from the function at the end

  HumidityValue += String(dht.readHumidity());  //Store int value from humidty sensor to String

   // Check if any reads failed and exit early (to try again).
  if (isnan(dht.readHumidity())) {
    Serial.println("Failed to read from DHT sensor!");
    Humidity += "Humidity = ";
  }
```

```
  Humidity += "Humidity = "+HumidityValue;
  delay(10);  // Delay before next digitalRead

  //return Humidity string from this function
  return Humidity;
}

String Motion(){

  String Motion;

  pirVal = digitalRead(inputPirPin); // read input value from motion sensor and store value in this
variable

  if (pirVal == HIGH) { // check if the input is HIGH
    Motion += "Detected";
  }
  else {
    Motion += "Ended";
  }

  delay(10);  // Delay before next digitalRead
  //return Motion string from this function
  return Motion;
   }
```