CS

COC251

B628603

Automatic Wireless Control System

AUTOMATED ROOM COOLING SYSTEM

Mustafa .M Adnan

Supervisor: Dr. Q Meng

Acknowledgments

I'd like to thank Dr. Qinggang Meng for supervising and providing support when necessary.
And I'd like to also Thank the Loughborough Finance team for Funding the project.

Abstract

Automated processes and automatic control are on the rise. With so many different opportunities presenting itself to the process of automation; it opens up the door to many possibilities that could revolutionise the world. The aim of the project is to develop an automated room cooling and home system that needs minimal user interaction. Based on different parameters and settings, the system will behave differently. The system was developed and based around the concept of automating room cooling, whilst the requirements were everchanging, the main objective of an Automatic Control system was never changed. There are many technologies that enabled us to develop this system, From the Arduino board technology and architecture, to the Android mobile application that was developed. Many hardware components such as Temperature sensors, LCD displays, and motion sensors were used to implement the requirements specified. To ensure quality of the final product testing was also carried out to ensure the system was viable, efficient, reliable and error free.

Table of Contents

Introduction

This report follows the detailed implementation and development of an Automated Room Cooling Home System. The system is developed in Arduino environment, on an Arduino Microcontroller board. Through using microcontrollers, we are able to create a digital device that can sense and interact with other devices both physically and digitally through the use of many Digital and Analog I/O Pins, and serial communication (wikipedia.org, 2019).

The system is based around two components, the first is the system itself and the second is the wireless controller. The system, as mentioned, is developed in Arduino IDE (which uses a subset of C/C++ Functions), this component is responsible for sensing, gathering, and processing data, this is essentially the main control unit. The second component is the mobile android application. This component is responsible for Wirelessly controlling the system. The idea behind this is to offer the user more than one way to control the system, this is to increase both convenience and reliability.

Motivation and System Objectives

The motivation behind this project is simply to minimise the effort it takes to cool down your room. And implement an intelligent system that can automate the process with minimal interaction. Ways of cooling such as using a standard Fan or an AC, is that it lacks the ability to process data and take actions based on that data.

As discussed, the system is designed to automate the process of room cooling, this meant that there are many variables to consider when designing the system. This is because cooling in general tends to be more about preference rather than fact, this was discovered through a private survey that I carried out between friends and family to find out if people like similar or different cooling preferences. This meant that the system needed to have a way of dealing with such variations.

The system aims to offer a full comprehensive automated system that depending on user preferences, the system will engage a different method (mode) of cooling. The system 3 different types of cooling. The first is the Default automatic cooling, the second is Motion Based Cooling, and last but not least, we have Distance based cooling.

In Addition to these 3 Different types of cooling, we also have Airflow Mode variation of each of the Cooling methods. Airflow Mode changes the way that each of the methods behave to allow finer customization of user settings and preferences.

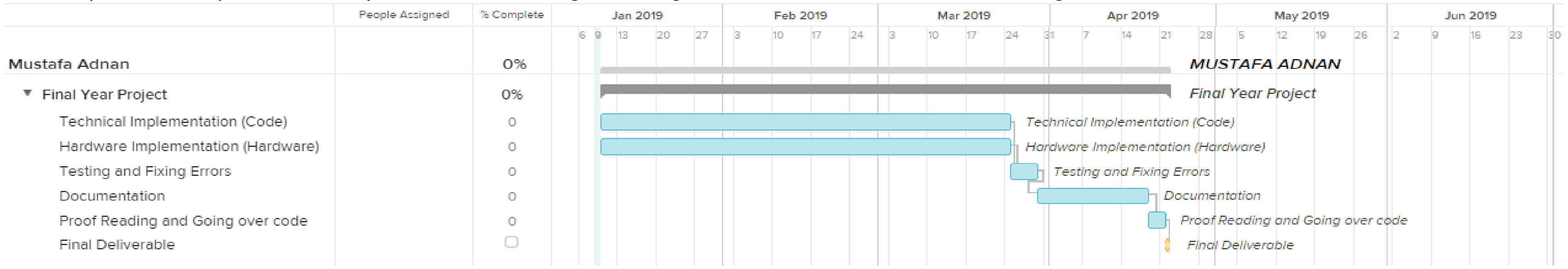
The system also aims to offer Analytics of the data gathered, this includes data such as Average temperature & humidity of each day of the week, and also show temperature & humidity levels of each day both at night time and day time.

Project Plan.

Project planning is a vital aspect to developing any type of project. Proper management and planning always leads to a more efficient and easier implementation and development phase. In This section I will discuss my project plan, define what was required at each stage and provide a time line of the project using a Gantt Char

Gantt Chart

This Gantt chart displays the project plan. There are multiple sections to the project plan, we have The technical implementation (software implementation), Hardware Implementation, Texting and fixing errors, Documentation, Proof Reading Documentation and code, Final deliverable.



	People Assigned	% Complete
Mustafa Adnan		0%
▼ Final Year Project		0%
Technical Implementation (Code)		0
Hardware Implementation (Hardware)		0
Testing and Fixing Errors		0
Documentation		0
Proof Reading and Going over code		0
Final Deliverable		☐

Technical Implementation

This is where we implement the code and software side of the system. This includes tasks such as coding the Arduino system or Mobile App.

Start Date: January 9th, 2019
End Date: March 24th, 2019

Hardware Implementation

In this phase we are aiming to implement the hardware side of the system. This includes things such as wiring Components and installing and modules/sensors that we are using. It's important to note that both the Software and hardware implementation phase overlap each other, this is because they are both intertwined. This means that both stages span over the same start and finish dates.

Start Date: January 9th, 2019
Finish Date: March 24th, 2019

Testing

Testing phase is where we aim to debug the system and find any underlaying or hidden errors that are causing the system not to operate efficiently. This is required to ensure quality of the system is at a satisfiable level.

Start Date: March 24th, 2019
End Date: March 31st, 2019

Documentation

This is the documentation phase, where we write up the final report on our development and findings. This report includes All findings and teachings learned and discovered across the Project and also detailed implementation of the software and hardware side of the system.

Start Date: March 31st, 2019
End Date: April 21st, 2019

Proof Reading and code checking

This is the last stage of the project, in this phase we proof read our documentation to check that it is up to a satisfiable quality and also run final checks on the code, this includes renaming variables, Commenting, and readjusting the layout of code to make more readable.

Start Date: April 21st, 2019
Finish Date: April 24th, 2019

Final Deliverable (Mile stone)

This is the deadline, in this we submit all of the deliverables, this includes documentation, Arduino Code, Mobile Application. The final deliverable the ending mile stone of the project.

Finish Date: April 28th

Literature Review

In this section I will cover the literature material that was involved in the project. This material spans over a multitude of subjects, from hardware components, to Tools, software and development models used to create the system and finish the project. All the findings from the literature aided me in decision making and aided in the process of development of the system.

Microcontrollers and Microprocessors

The first decision of course was to choose a microcontroller or a microprocessor. This was an important decision as they are both extremely different, and the implementation of the project would be completely different based on the choice I'm going to make, so this was an extremely critical decision.

To compare the two technologies, we can have a look at the specification and compare them together. As you can see In Figure A below, a microprocessor is much more powerful than a microcontroller, it has higher clock speed, has a 32-bit register as opposed to an 8-bit on microcontroller, etc. It's also important to note that Microcontrollers come embedded with their own RAM, ROM, and EEPROM, whereas microprocessors need to use external circuits (Circuitdigest.com, 2019).

From what we see on the table it is obviously clear that microprocessors are much stronger and much more capable, however, power doesn't necessarily mean that it is the better choice. A microprocessor is basically a small computer, it is used for things other than what I am trying to achieve, and since I have a predefined task a microcontroller would be the better choice (Choudhary, 2019).

With a microcontroller I wouldn't need to worry about Operating systems, different languages and multiple environments that must integrate with each other, using a programmable microcontroller such as Arduino with already an existing programming language, libraries and capabilities that are enough for my requirements.

	Arduino (Micro Controller)	Raspberry Pi (Micro Processor)
Processor	AVR ATmega328p	Broadcom ARM1176JZF-S
Clock Speed	16MHz	700MHz
Register Width	8-bit	32-bit
RAM	2k	512MB
GPIO	20	8
I/O Current Max	40 mA	5-10 mA
power	175 mW	700 mW
Operating system	None	Linux and Others

Arduino Boards

Choosing an Arduino board out of the many they have was difficult, this is because not only do I need to consider the features of the board and how they will align with my requirements, I must also remember the constraints I am dealing with. For example, I lack Engineering skills, so I mustn't choose an Arduino board which requires engineering works such as soldering.

Some of the Considerations are as follow:

- Ease of use
- Expertise Required
- Number of Pins
- Memory size
- Cost
- Limitations
- EEPROM Availability for saving data

The Arduino Mega has 70 pins, Built in TX/RX ports, 256KB of flash memory, 8KB of SRAM and 4KB of EEPROM. This board also allows us to store much bigger programs (Arduino.cc, 2019). This meant that the board meets more than enough requirements.

Arduino Programming Language (C++ and C)

Since we are using Arduino, the Arduino IDE provides a set of C and C++ functions that we can call. The Arduino language is a subset of C and C++. Our code is compiled by a C++ compiler, namely (avr-g++). The build process is formed of two parts, the pre-processing, and compilation. Pre-processing is responsible for making some changes to the sketch to prepare it for the compiler, things such as adding missing libraries, the pre-processing phase also generates prototypes for function definitions, whilst the compilation is for compiling as previously mentioned (GitHub, 2019).

Arduino IDE also offers a multitude of libraries, these libraries offer the functions that we can call, these functions are extremely important in the success of the project. Most components and modules that are used require these libraries, for example the DHT22 sensor needs the DHT library to function.

Development Model

Choosing the correct development model is extremely important. The Development model sets out how the software and hardware will be developed. There are many different types of models that we can use (Waterfall, Scrum, V-Model, etc). All of these models have different pros and cons and are suited to different types of projects.

If we were using the waterfall model for example, it wouldn't allow us to change requirements mid-development, however it is good model for small-midsized projects (Medium, 2019).

To choose the correct model, we must ensure that it fits my project. For example, Scrum would suit our project more than waterfall since it can adapt to changes in requirements easily (Medium, 2019).   And due to the nature of my project, requirements change can happen.

MIT App Inventor 2

MIT App inventor 2 is a free opensource android development platform. The platform is managed by MIT but previously owned by google. MIT App inventor offers a user-friendly graphical representation and methods of writing code (Wikipedia.org, 2019).The code that you write in MIT App inventor comes in the form of block, these blocks attach to each other forming a block of code. When it comes to creating a user interface, the platform offers many components such as buttons, text boxes, labels and many more that are common to android app development. It also offers not graphical components such as Bluetooth client and System clock that allows us to implement important functionalities, such as wireless control.

Other Tools used (Fritzing)

To complete the project, many tools were required, this is not only from development point of view but also management and illustrations in the final report. An important program that played a huge role in both development and management is Fritzing. Fritzing is a program that allows you to create schematics and has its built in Arduino IDE (Wikipedia.org, 2019). The program is essential for planning implementation of hardware and also visualizing the system before the development is engaged. This piece of software also played a huge role in illustrations in the report as it allowed us to create schematics of the hardware implementation, allowing for a clear and graphical representation to the reader.

Software Development Life Cycle

Agile Scrum Development model

As discussed in the literature review, choosing the correct development model is vital to the success of the project. Different models have different pros and cons, and each are designed for different types of projects.

Since my project needed to be implemented under a tight deadline and also considering the possibility of requirement changes and possible hardware changes; this led me to choose an agile development method that accommodates changing requirements. This development model is the Agile Scrum Model. SCRUM is generally used in a group environment where there are dedicated scrum masters, Product owners, stakeholders and staff. Since this is a 1-person project, I had to adopt and slightly change the structure of the model to fit my needs (Simplilearn.com, 2018).

Adopted Structure (Framework)

As discussed, I needed to slightly change the structure of the model to better fit my project; to fit the style of my project, I needed to remove the aspects of Scrum master, and Stakeholders and product owners. The following Framework is the process I used to develop the system:

1. Product Owner (Me) Creates a backlog. This backlog is a list of tasks that needs to be completed in the project.
2. The tasks in the backlog are then broken down into smaller more manageable subtasks.
3. A Sprint backlog is created, this covers what should be done in that specific sprint
4. Duration of the sprint is then decided, depending on the size and the difficulty of the sprint, I used between 1 to 2 weeks for each sprint.
5. After each sprint is done, I review what has been implemented, and document accordingly if need be.

Pros and Cons

With the SCRUM model, we can implement functionality in a sprint manor, where after each sprint of implementing new features and functionality, we then also test the product. This allows us to create an MVP (Minimal viable product). Each iteration will add features and functionality to the software until it is considered complete (cprime.com, 2019).

As you can see, this type of development model, fits perfectly to our needs for the projects. It allows us to iteratively revisit the requirements and coding phase, add features and functionality as we progress through the project. Since the model is designed to handle requirement changes, and since testing is carried out after each sprint, it allows us to detect and fix bugs and issues quickly, in the case we aren’t able to fix the problem we can quickly Roll back to one of the older versions (Medium, 2017).

However, as good as Scrum is, it also presents its self with some problems and limitations. The first problem is the everchanging requirements. Since we are always adding new features, and essentially constantly changing the design of the system, it creates a vulnerability in the documentation; the new added features and requirements might not always be documented. If the requirements and of the system are not completely understood, then we can deviate a great deal from our original objectives and lead the project into a downfall (Medium, 2017).

Requirements

This section here allows us to capture the requirements of the system. To properly implement functionality to the system, we need to predefine them before we start. Requirements is essentially what the system will be able to do, i.e. control fans, intruder alert, etc. To capture the requirements, I will use a table, in the table, there is the name of the requirement

Requirements	Description	Priority
Automatic Fan Control	The system should be able to control the fans automatically depending the target temperature & humidity and settings chosen	5
Manual Fan Control	The user Should be able to control the fan manually through the LCD Display	5
Wireless Fan Control	The user Should be able to control the fans wirelessly through an android Application	4
Set Target Temperature and humidity	The user should be able to specify their desired temperatures to cool down to.	5
Intruder Alert	The system should have intruder alert that warns the user when there is an intrusion detected.	3
Smoke Alert	The system should have a smoke alarm	2
Motion based Control	The system should offer a mode where the system cools down the room based on whether the person is in the room or not	3
Distance Based	The system should offer a mode where the cooling depends on how far you are away from the fans.	3
Airflow Mode	The system should offer a mode where it prioritizes increasing air flow in the room. This mode should change how other modes behave accordingly.	5
Analysis	The system should provide analysis of data gathered. The system should display the temperature & humidity of each day, both at night time and day time. In addition, the system should also work out the average temperature and humidity of each day.	4
Real Time Clock	The system should display a real time clock. This clock should keep the correct date and time even if the system switched off	2
LCD Touchscreen	The system should include an LCD Touchscreen display that allows the user to fully control the system.	5
Android Application	The system should include an App that allows the user to fully control the system wirelessly, this includes turning on and off settings, setting targets, and viewing analysis of the data gathered.	5
Saving User Data	The system should be able to save and retrieve user data. Target Temperatures, average temperatures, past temperatures, etc.	5

Limitations

It is extremely important to realise our limitations when undertaking any type of project. Understanding our limitations and constraints will allow us to plan accordingly, relative to them. This is to ensure that the project progresses smoothly without any hinderance. There are many possible limitations that underlay project. In this section I will talk about the limitations that I underwent to complete the project.

Cost

Cost is an extremely important limitation. In my project, due to the nature of the system that I am designing and implementing, cost is an extremely important aspect. This is because my system doesn’t just require software programming, but also requires hardware. This hardware is purchased through the open market since it’s a self-funded project.

Choosing the wrong components would essentially mean I have to purchase new ones, this is extra cost that may not be planned for. We have to ensure that anything that is being purchased is indeed the right purchase. We need to also evaluate how much money we are willing spend and how much money we have, so we can plan what to buy accordingly.

Time

Time is an extremely important constrain. We need to define our deadline to correctly plan our stages of development and our stages of documentation. Not realising our limitation on Time could possibly mean we fail to meet the deadline. We need to carefully allocate time to each stage of our development and also have contingency plans in case our project does fall back behind.

Expertise

Constraint on expertise is arguably the most important. We need to fully understand our capabilities and must understand when something is out of our technical or mechanical expertise. This is extremely important to consider as it could affect the flow of the project. Assume, we purchase a component that requires mechanical work, such as soldering or includes electrically manipulating circuits, then it would essentially cause the project to fall behind if I lack the expertise to carry out the procedures, this could either lead to two things, the first is plan for a different approach and try again (which would require more time, which is one of our other constraints) or lean the required skills necessary, which will also take time.

Risks

Risks is an extremely important aspect to consider when undergoing any development or project. If we are able to measure and weigh the risks beforehand, we can foresee possible issues and problems and formulate contingency plans and control measures, to properly manage the project and to ensure that I successfully meet my deadline if the risks do happen.

To illustrate the potential risks and their consequence I shall use a table. This table will contain a description of the possible risks, The probability of the risk happening, the level of consequence and plan of control. Probability ranging from 1-5, 5 being the highest, same thing applies for consequence. Plan of control refers to the methods or contingency planning that we use to manage the risk.

Risk	Probability and Consequence	Plan of control
Data Corruption or Deletion	Probability: 1 Consequence: 5	To ensure that we do not lose data or delete our data we need to back up our work regularly. This can be done through many automated services or can be done yourself.
Components/Modules Breaking	Probability: 2 Consequence: 4	Research implementation of the modules before attempting myself to ensure the connections made are exact.
Stages of development increasing in duration	Probability: 3 Consequence: 3	Anticipate stages of development overrunning and have plans in place to make up for lost time to allow the project to catch up to the deadline.
Personal Problems	Probability: 2 Consequence: 2	Plan for possible personal problems that may affect the progress of the project and implement a plan where we are able to reallocate time wisely.
Budget running out	Probability: 3 Consequence: 3	Research the total costs comprehensively and keep cost in mind when picking parts and components.

Hardware Implementation

Due to the nature of the project Choosing the correct hardware components is vital, this is because the functionalities of the system are enabled by the use of hardware components in conjunction with software programming, the code is designed to interface with these components to provide the functionalities in the system, and to achieve the overall objectives of the system. In this chapter I will talk about which components I used, why I used them, how I used them, and provide implementation schematics for each component.

List and Purpose of hardware components

As stated previously, to achieve the requirements (e.g. Automatic Control of the fans, Intruder detection, etc, etc) It was vital to research and assemble the perfect combination of components that would allow me to achieve the objectives of the system. These chosen parts need to interface together through the Arduino Microcontroller, and only together can they achieve the full functionality I intended. Each component was chosen for specific reasons, reasons such as accuracy, reliability, cost, efficiency and intended purpose. Choosing the wrong component on the other hand, would have adverse effects on the progress of the project, this is due to multiple reasons.

The first reason would be the effect on my deadline. This is simply because of delivery time, delivery time is an important factor to consider when working under a strict timeline, this is because this aspect is usually out of the hands of the developer. If, for example, I chose the wrong component after waiting a week for the delivery service, it would essentially mean I have wasted one week on waiting, then I would have to further allocate time to searching for a more suitable part and then wait again for the next delivery, and if the component is out of stock then even more extra time is required for the supplier to restock.

The second reason is the capability (or scope) of the component chosen. Does it fit the scope of the project? And does it fulfil the required objective? Understanding the capabilities AND limitations of the module (Component) is vital as it will allow us to foresee what we can do with it and if it can help us in any way to achieve the requirements of the system. Choosing the wrong component for the intended requirement could either lead to poor implementation of the requirement, or worse yet, it could potentially lead to not being able to implement the functionality.

Now let's talk about the modules that I have chosen, their intended purpose and the functionality that it achieves; the list of components is as follows:

- Arduino Mega 2560 Microcontroller
- HC-05 Bluetooth Module
- HC-SR04 Ultrasonic Sensor
- DHT22 Temperature and humidity sensor
- MQ2 Smoke and Gas Sensor
- DS3231 RTC (Real Time Clock)
- PIR Motion Sensor
- 2.8" Adafruit ILI9341 TFT LCD Display (320 x 240)
- MOSFET N-Channel Transistor
- 2k OHM Resistor & 1k OHM Resistor
- Buzzer
- 5V 2mA Power Supply
- 12v 2mA Power Supply

Arduino Mega 2560

The Arduino Mega 2560 is a microcontroller that is responsible for the collection and processing of data. The Arduino enables and gives us a platform to interface with the other sensors on. All of below stated components and sensors only work for their intended purpose because of the microcontroller. The Arduino IDE provides tools and libraries that allows us to implement code and functionality to the microcontroller. As stated before the microcontroller essentially allows us to access the data provided by the sensors and components that are attached to it, this is thanks to the Digital, Analog and Digital PWM I/O pins.

There are a multitude of Arduino boards that are designed for different applications. For example, we have the Arduino UNO which is designed for small programs and projects since it has small memory, also provides a limited number of I/O Pins (14 Pins, 6 of them having PWM and 6 Analog I/O Pins) whereas the Mega 2560 provides a much larger number of pins (54 I/O Pins, 15 of them Have PWM, and 16 Analog pins) (Gudino, 2017). Due to the nature of my system, having a large number of Digital pins is a necessity, this is because of the multiple sensors and modules I will be using, not only that but the TFT Display alone will use 18 Pins in 8-bit mode, In addition, each sensor and module will require between 1 to 3 Digital I/O Pins. The size of the software I implemented is also large, overall, its 2,000 lines of code, so having large memory that can suffice that is also a necessity.

HC-05 Bluetooth module

The HC-05 Module is a Bluetooth module for transmitting and receiving data (Can be either slave or master) through serial communication. The purpose of this module is to establish wireless communication between the Arduino Board and a mobile device. Having this communication enables us to create the App for a mobile device (or tablet) and implement Wireless Control functionality, where the user is able to change settings of the system, set target temperature & humidity and view analysis of the past week’s temperature and humidity statistics.

The HC-05 was the best available choice to implement this functionality, even though there are alternatives such as the HC-06, however the HC-06 is only configured to slave (Currey,2019). Due to the nature of the system we will be transmitting AND receiving data, so, Slave/Master Configurability is vital to the success of the intended objective of the module.

HC-SRO4 Ultrasonic Sensor

The HC-SR04 Ultra sonic sensor is the component that we will use to calculate the distance of an object from the device. This is done through ultrasonic sound, where the module emits an ultrasonic sound and calculates the time it takes back to reach the module again, through that we can calculate the distance. The purpose of this is to implement distance-based control of the fans, e.g. the further away you are sitting from the fans, the faster and more powerful they become. This is done to fulfil the objective of automation. The system contains many different settings that controls the cooling in different ways, distance based is one of them.

DHT22 Sensor

The DHT22 is a module that is responsible for sensing the temperature and humidity of the room, the module is a cheap and very accurate sensor, the DHT22 provide float accuracy readings meaning up to 2 decimal points. At the start of the project I used the DHT11 sensor but ran into a problem when there was an accident that fried on of my pins and that also lead the DHT sensor being damaged, so when purchasing a new one I decided to go the better version, the DHT22. The DHT22 version also provides a bigger range of up to 100^{o}c as opposed to the DHT11 where it only goes up to 50.

This module is very simple but also extremely important to the project. Most of the system's automation is based on the readings that we receive from the sensor. With the readings of the sensor, we can calculate the power of the fans. Also, the DHT22 Sensor, together with the RTC module allows us to capture the temperature and humidity readings during a set a time throughout the week to provide analysis such as average temperatures of the week. The recordings of the readings are saved on the EEPROM of the Arduino microcontroller, so even if the user turns the system off, the data is not lost.

MQ2 Sensor

This sensor is responsible for sensing the smoke in the air. This reading is important to implement the smoke alarm. The smoke alarm is an essential of any home control system, it was only natural that this is one of the minor functionalities. The sensor continuously monitors the environment checking for the air density, if the density crosses a certain threshold the buzzer will go off.

DS3231 RTC

The DS3231 RTC is a real time clock module, the purpose of this module is to enables us to keep a real time clock, even when the system is off the RTC module keeps going on a lithium battery. As simple as a real time clock sounds, it enables us to implement multiple functionalities, first is being able to keep display a real time clock to the user, second is allowing to capture data at certain times (e.g. Temperatures Every Day at 12pm and 10pm), and through that, it allows us to work out the average temperatures and humidity levels.

PIR Sensor

PIR Motion sensor is an infrared light sensor that senses movements from objects that emit infrared light. This allows us to implement the motion sensor functionality where the speed of the fans change depending if a person is detected in the room or not. There aren't any other alternative modules that we can use to implement this functionality, the only thing that is close to this is using a HC-SR04 Sensor but that could create a multitude of problems since it detects the first object in its path and not moving infrared objects. Not only does this sensor play a part in the functionality of the fan automation but it also enabled me to create an intruder alarm for the system.

Adafruit TFT LCD Display

The Adafruit TFT LCD display is a method of visual representation of the data and a method of control of the system. The Adafruit ILI93421 TFT LCD Display is a touchscreen GUI that I implemented. The GUI is a comprehensive tool to control the system and also view the data and current readings. This component allows the user to essentially control and view the system.

There are many different display and alternatives available, however, I found that using an LCD Touchscreen display in conjunction with an Android Application would be the best and most efficient way to control the system, this is because we need both central and wireless control, this is to ensure the home system can be as accessible as possible. An example of where this is useful is for example, if the user gets into bed and forgets the system on; the user can use their phone to turn the fans off or change the settings.

N-Channel MOSFET Transistors

N-Channel MOSFET Transistors are used in this project to enable us to control the power of the fans. These transistors allow us to control the power of the fans through using PWM Digital I/O pins and applying different duty cycles to the pin, essentially, we are turning the fan on and off really fast at a specified duty cycles to lower the RPM of the fans.

This component, as small and as simple as it is, it allows the whole operation of automation of the fans. Without these transistors we would only be able to control the fans in an off and on manner, we wouldn't be able to specify certain power levels e.g. 40% Power (Duty Cycle).

Resistors

Resistors are important in any project, they allow us to control the power levels when dealing with intricate modules that operate at different power levels than the rest. For example, in my project I have used the resistors to create a voltage divider, this is because the RX pin for the HC-05 Module operates at 3.3v Logic level, since both the HC-05 and Arduino mega operate at different logic levels, we need the voltage divider to pull down the voltage otherwise the module will get too hot and burn out. We also need to pull down

Resistors are commonly used in many areas of electronics, they are a necessity for correct voltage regulation throughout the project to ensure the components are operating under the safe and correct voltage levels.

Buzzer

The buzzer is used in conjunction with the PIR motion sensor and the MQ2 Gas sensor. The buzzer together with the GUI is used as a smoke and intruder alarm. If the motion is detected with intruder alert mode on then the buzzer will sound an alarm, the same thing applies for smoke detection, if there is smoke detected then the buzzer will sound an alarm.

The purpose of this component is to implement some basic safety features in the home system, as any home system needs a form of security features.

Power Supplies

Power supply is an important factor to consider. We need to ensure that we are feeding the right power into the right components, otherwise we could face catastrophic consequences. If we provide voltage power that is too high then the components will burn out, if we don't supply enough AMPS then it will also cause problems.

Because the system is responsible for controlling fans, this meant that we will be needing two types of power supplies, one 5V 2mA power supply for the Arduino and the 5v components such as the DHT22, HC-05 and HC-SR04 sensors, and 1 12v Power supply for the fans.

The PWM fans operate at 12v, thus meaning our 5v power supply running into the Arduino will not be able to fully drive the fans. This meant that we will need a separate power supply to drive the fans to their full power.

In-depth Analysis and Schematics of Components

Now that we have looked at the reasons and purpose of the components, to further understand the necessity and to further understand their operations, we now look at the in-depth analysis of how these components operate to provide us with these readings and functionalities. The schematics of the hardware implementation provide a coherent representation of the implementation/installation of the module and allows us to see accurate wiring of the module.

HC-05 Bluetooth Module

As stated earlier, the HC-05 is a Bluetooth module that uses serial communication to allow us to communicate to other devices. The HC-05 is an extremely important component, but to further understand how we have implemented it we must first understand serial communication technology, and we must also understand logic levels to conceptualize why I have used voltage dividers to bring down the voltage level of the HC-05 RX Pin.

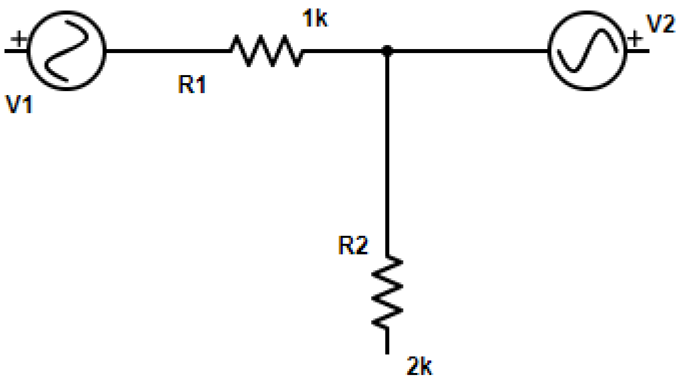
Serial communication is a process of sending data 1 bit at a time, sequentially, over a channel. This is how we communicate between the Arduino and the mobile device through the HC-05 module. On the Arduino Mega board, we have the RX and TX Pins, these two pins are the two channels of communication. The RX Channel is responsible for receiving data whereas the TX Channel is responsible for transmitting data. The Arduino Mega Board also uses these two channels (Pins number 1TX and 0RX for connecting with the computer, this means that when we upload code to the Arduino these two pins are used for communication (Huang, 2019).

Logic Levels are physical variables that represent the information in any digital circuit. Data is transmitted as a bitstream of binary (1s and 0s .e.g. “0110101”). For example, when we transmit the data to the Arduino, the data is transmitted by changing the voltage level across the TX wire, the allowed logic level for the TX Wire connecting the HC-05 and the Arduino board is 3.3v, thus meaning we need a network of resistors (1k Ohm resistor and a 2k ohm resistor) to bring it down (Huang, 2019).

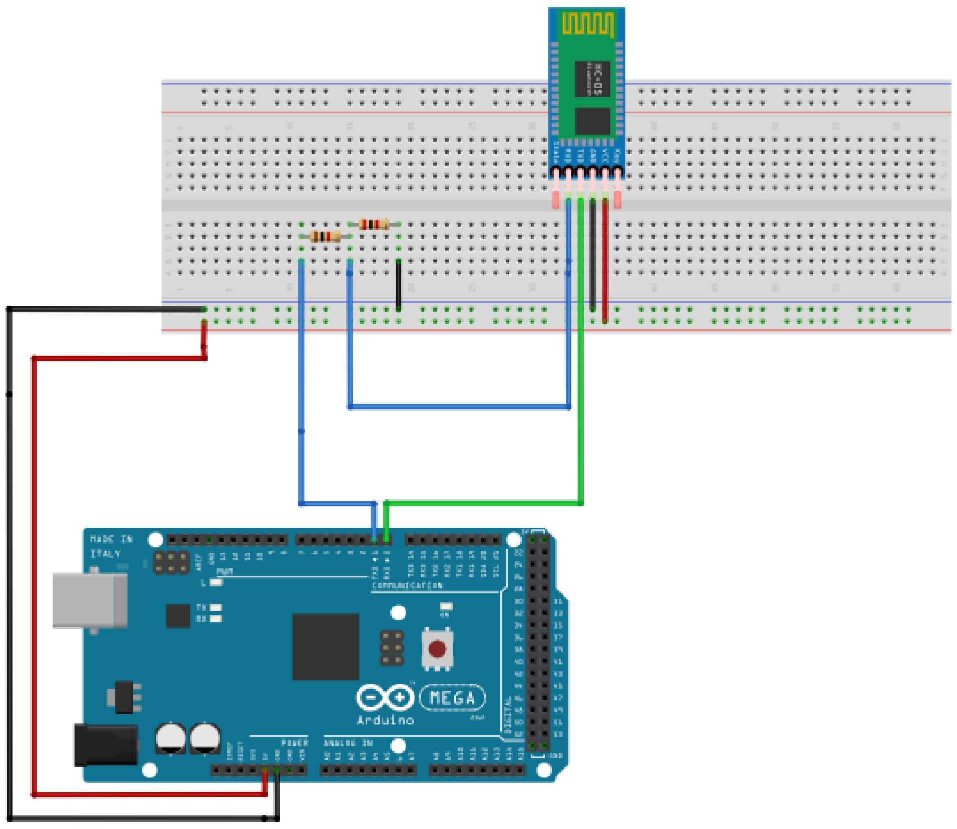
This can be done through the use of resistors to create a voltage divider. A Voltage divider is simply a network of resistors that is used to scale the voltage, the scaling depends on the value of resistors that we use. If we use the HC-05 Voltage divider that I used as an example, it would look and formulate as follows:

Where V1 = $\text{Voltage}_{\text{in}}$, V2 = $\text{Voltage}_{\text{Out}}$, R1 = 1k Ohm Resistor, R2 = 2k Ohm Resistor

$\text{Voltage}_{\text{out}} = \text{Voltage}_{\text{in}} * \frac{R2}{R1+R2}$



Now that we understand the technology behind the module and the reasoning for the resistors we can now look at the wiring of the module. As stated previously we need to use a voltage divider for the TX wire that connects the HC-05 RX pin to the Arduino TX pin. So, the blue Wire which Connects the HC-05 RX to Arduino TX passes through our voltage divider, The voltage divider must also be connected to the common ground. The Green wire is the HC-05 TX Pin that connects to the Arduino RX pin, this pin doesn't need to run through the voltage divider. Last but not least The VCC and GRND pins on the HC-05 connect to the 5v Power and GRND pins respectively.



HC-SR04 Ultrasonic Sensor

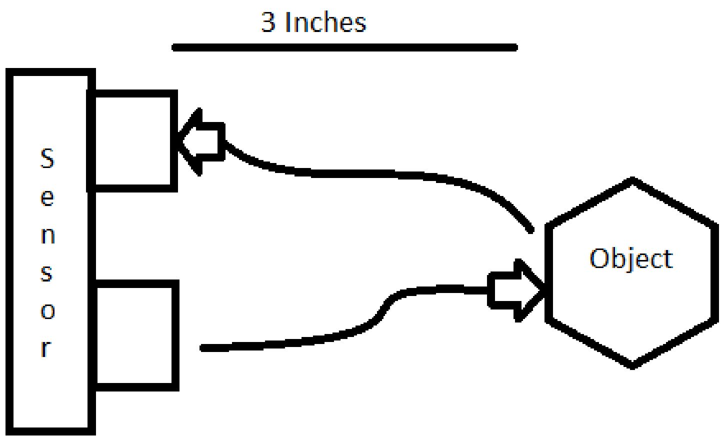
As discussed earlier, I use the HC-SR04 Sensor to measure the distance between the user and the fans. This is done using sound waves. But how is this done?

The module emits an ultrasound at 40k Hz, this sound wave travels until the it hits the first object, this sound wave will reflect back to the module, by doing so it will allow us to deduct the time it has taken to travel, and since we know the speed that it has been emitted at we can calculate the distance. This is possible thanks to the Echo and Trig pins on the module.

To Emit the sound wave we need to set the Trig pin to a high state for 10 microseconds, this produces an 8-cycle sonic burst, the Echo pin on the other hand is responsible for listening for that sound and calculating the time it takes to reach back, this reading will be outputted by the echo Pin and will allow us to work out the distance (Sparkfun, 2019).

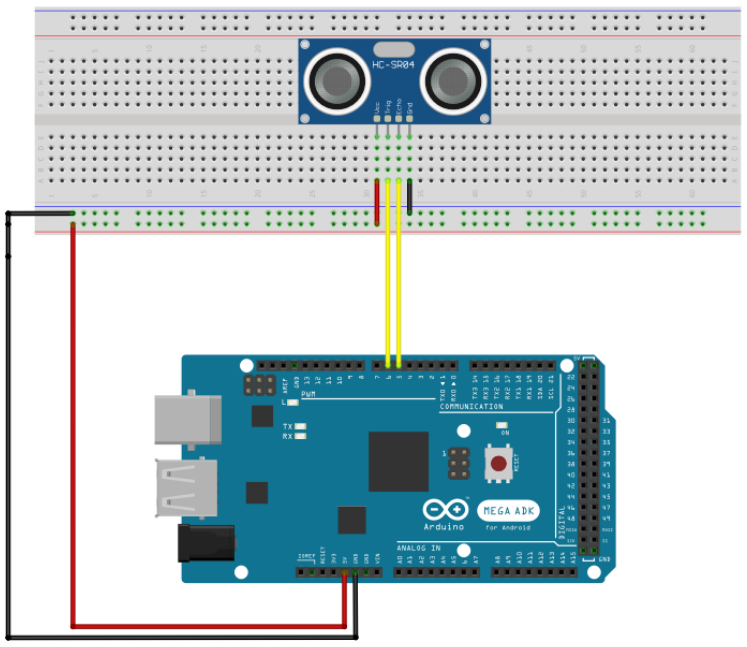
Let's assume this as an example, where have an object that is 3 Inches far away from the sensor; calculation of the distance would formulate as follows:

Where Speed of Sound = V = 0.0133 inch/µs,
Time = Distance / Speed = 3 / 0.0133,
Distance = Time * 0.0133 / 2



The reason that we divide by 2 is because this calculation returns the Round-Trip time, from the sensor to the object and then back to the sensor, however we only need half of that calculation since we are only trying to measure the distance TO the object, not distance From sensor to object to sensor again.

Now that we have understood how the module works we can look at the installation of the module by referring the schematic. The HC-SR04 is a relatively simple module to install, the module consists of 4 pins, GRND, VCC, ECHO, and TRIG. Echo and trig pins like we discussed are responsible for sensing the sound, one emits an 8-cycle sonic burst and one checks for how long it takes to reach back, Both of those pins are connected to any Digital I/O Pin (Yellow Wires in the schematic). The GRND and VCC pins connected to the ground and 5v power respectively. The module doesn't require any resistors or a different power supply since it operates at 5v, thus allowing the module to have non-complex installation and implementation.



Adafruit TFT Display

The Touchscreen LCD Display was the biggest component that I had to implement. There are many different LCD Displays and many of them are connected differently, for example we can either get them on breakout boards or we can get them as shields.

The TFT Display that I installed uses 16 Digital I/O in total, 6 of them being Analog. The Display has two modes, SPI and 8-bit mode. The difference between these two modes is that you essentially use more Pins for faster speed in 8-Bit mode, whereas SPI Mode is 2-4 times slower but uses less pins. Let's take a look at the pins and what they are responsible for.

Ground and Voltage In Pins: First we have the GND and 3-5Vin Pins which are connected to ground and 5v Pin respectively. This is of course needed to supply power to the Display. In the schematic provided below it is the Black and Red wires.

Chip select (C/S): This pin is connected to Analog pin A3, chip select is a technology used to select a specific chip if there are multiple one connected, this is to allow for control of data flow, for example, if we want to ignore the inputs and outputs of a specific chip we cannot include it in the set of chips that we have selected. In the schematic Shown below, this is one of the green wires, the rest of the green wires are C/D, WR, and RD.

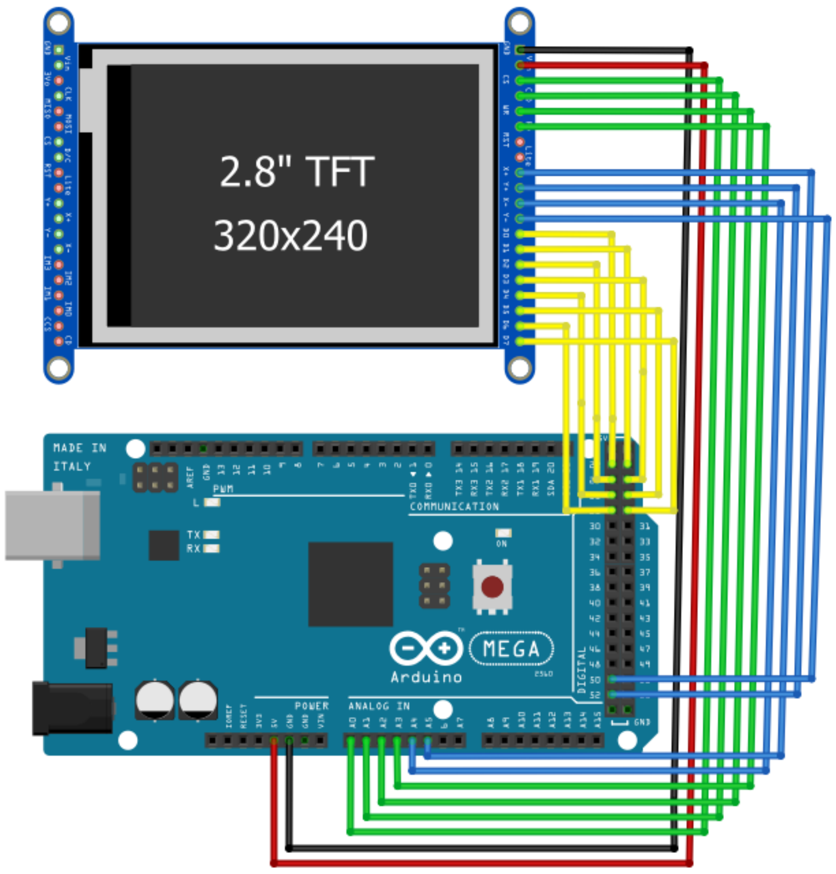
Command Select (C/D): This is the command selector; This wire is connected to the Analog A2 pin.

Write Strobe (WR): This is the write strobe pin, and this pin is connected to Analog A1 Pin. It is also one of the green wires in the schematic provided below.

D0 **–** D7: These are the parallel data lines. These data lines transmit data to the TFT in parallel, each pin sends one Bit. D0 sends the least significant bit whereas D7 sends the most significant Bit.

The wiring of these pins has to be in a specific order, this order is specified by the datasheet provided by Adafruit. The wiring is displayed as the yellow wires in the schematic below. The connections are as Follows:

D0 – Digital I/O 22
D1 – Digital I/O 23
D2 – Digital I/O 24
D3 – Digital I/O 25
D4 – Digital I/O 26
D5 – Digital I/O 27
D6 – Digital I/O 28
D7 – Digital I/O 29

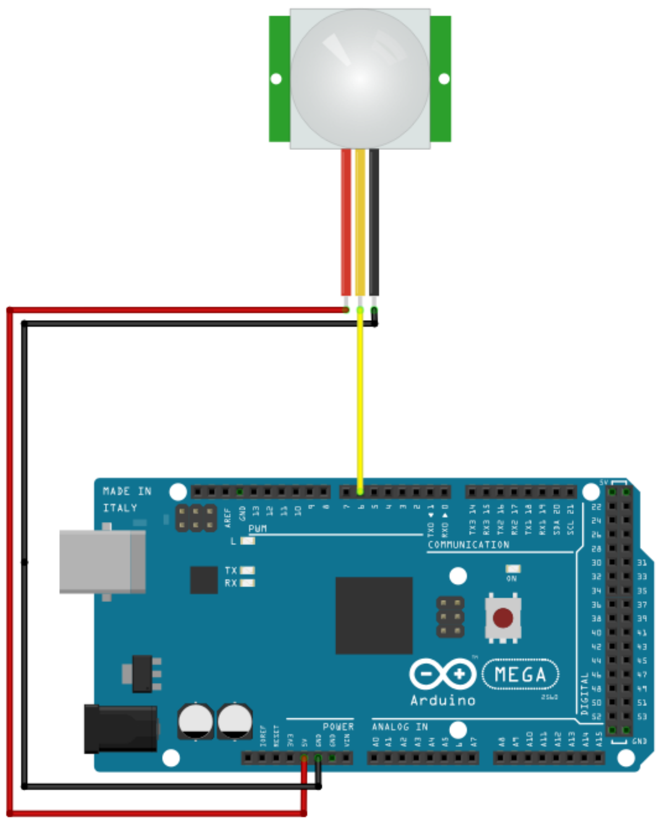


PIR motion Sensor

The PIR motion sensor is an extremely important component, and to fully utilize its capabilities we need to understand how it works. The PIR motion sensor consists of two slots, each slot detects IR (infrared radiation). There are multiple states to the sensor, and in each state the sensor behaves differently.

When the sensor is idle, both of the slots are detecting the ambient IR, they both detect the same amount. However, when a person passes through the detection lens, the first slot will detect the warm body, when this happens, the first slot will be detecting more IR than the second one, thus causing a positive differential change, a negative differential change is generated when a person or a body generating IR leaves the sensing area. This signal is read through the Digital pin, and through that we are able to detect the two different states (“Motion Detected” and “Motion Not Detected”). These states enable us to implement the functionalities I have discussed earlier.

The wiring for the module is simple, the PIR motion sensor is a 3 pin Module. It has a voltage in pin, which can take voltage power from 3v to as high as 12v. The other two Pins are the ground and Signal output pin, the signal output pin is what we read to detect the two different states.



DS3231 RTC

The DS3231 RTC module is extremely important in our project, as I have covered earlier it allows us to implement more than one functionality. The DS3231 module is the most accurate Real time clock module you can attain, this is because of the measurers that it goes through to ensure that the time kept is as accurate as possible.

Generally, a 32khz timing crystal is used in standard RTC modules to keep the time when main power to the module is off, however when the temperature increases, it changes the oscillation frequency. Although it is a trivial amount, over a period of time It will become noticeable.

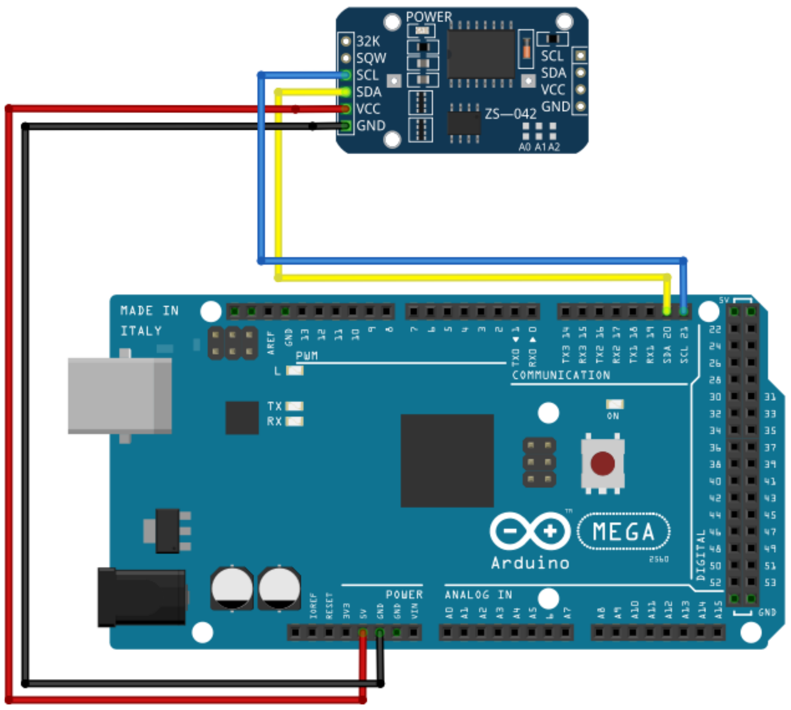
As mentioned above, the DS3231 Version of the module takes extra measures to ensure accurate time keeping, the first is that the crystal is located inside the chip. Now as mentioned earlier changes in temperatures changes the oscillation frequency, however in the DS3231 Module it has a temperature sensor of its own, the module then compensates for the changes in frequency by adding and removing clock ticks, this is covered in the datasheet of the component (maximintegrated.com, 2019).

The wiring of the module is relatively simple, we need 4 wires in total, for our VCC, GRND, SCL and SDA pins.

SDA (Yellow wire in the schematic) connects to the I^{2}C SDA pin on the Arduino. This pin is the data line for the I^{2}C serial interface.

SCL (Blue wire in the schematic) is connected to the I^{2}C SCL clock pin on the Arduino. This line is responsible for synchronizing the data movement.

VCC (Red Wire) and GND (Black Wire) are the 5v and ground pin respectively.

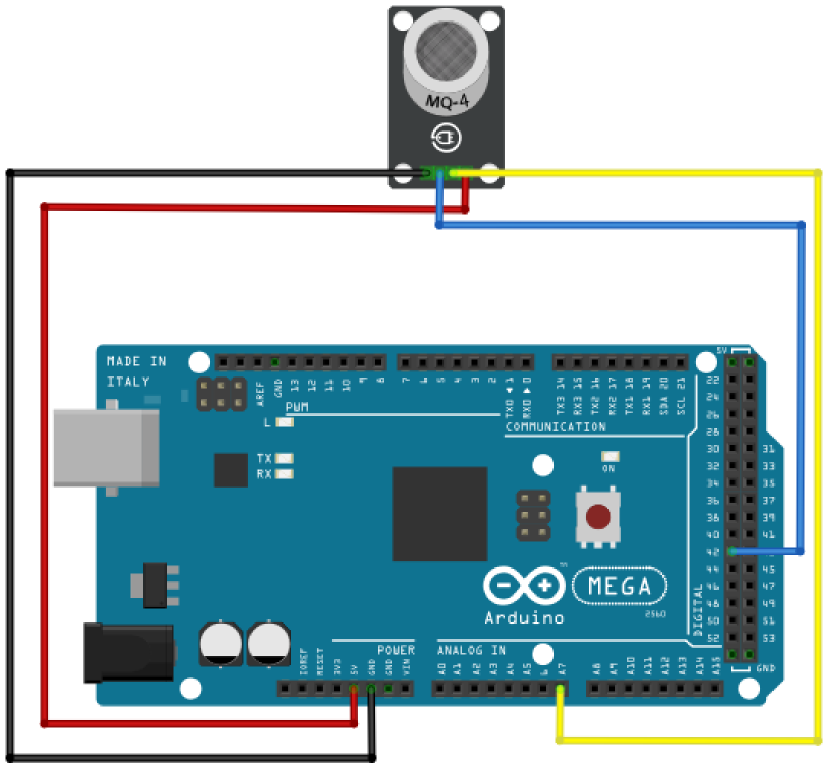


MQ2 Smoke sensor

The MQ2 Sensor was a rather straightforward implementation, however, the module also needed to be calibrated to the smoke we are sensing. The sensor uses 4 pins in total, 1 Digital I/O and 1 Analog I/O Pins. If smoke is detected, the voltage changes relative to the concentration of smoke. E.g. the higher the density of smoke is, the higher the output voltage will be, this is provided through the Analog output of the module. The digital output pin provides binary data based on the presence of combustible gases present in the air.

To calibrate the sensor, you need to adjust the sensitivity at which to detect it. You do this by putting the sensor next to the gas and turning the sensitivity dial on the back of the module until the red LED on the module is glowing red.

The Wiring the quite simple for the module, 4 Pins VCC is 5v power wire (Red Wire), Ground connects to ground on the Arduino (Black Wire), The last two connections are the Analog and Digital I/O Connections. D0 (Blue Wire) can connect to any Digital I/O and A0 (Yellow Wire) can connect to any Analog I/O Pin.



MOSFET N-Channel Transistor and PWM Fan

As stated before the MOSFET transistor is a vital component to the project, this component allows us to control the power of the fans, which is the overall objective of the system, which is to automate room cooling. Due to the fact that I am using a fan that uses a different power supply to my Arduino makes the implementation a little more complex. The Fans use 12v DC power, the important fact to notice here is that these fans must be PWM type fans, this will allow us to use the PWM pin on the Arduino in combination with the transistor to write different duty cycles to the fan.

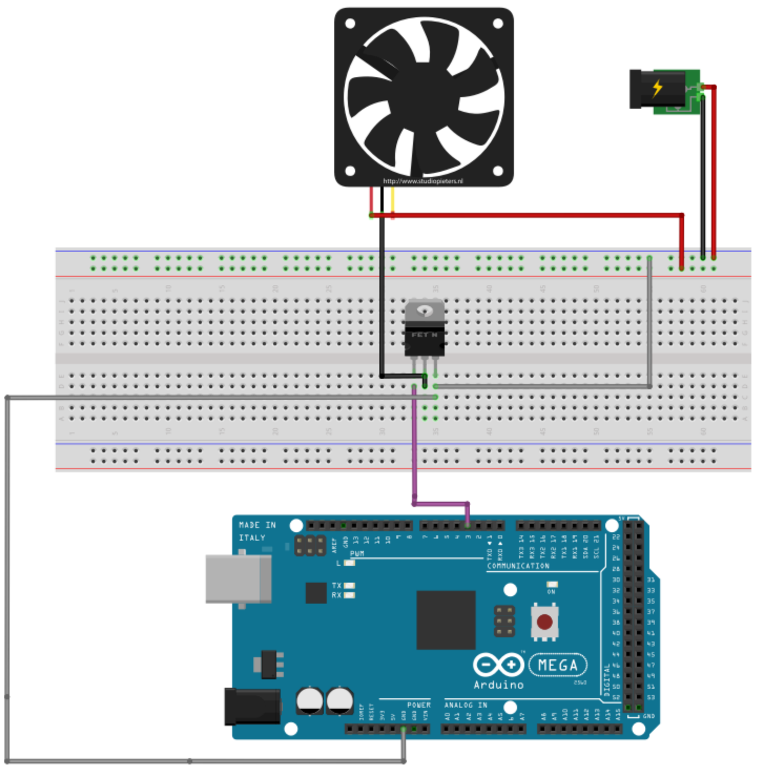
Before we look at how I implemented the transistors in my project we must first understand what a MOSFET N-Channel transistor is.

A transistor is like a pipe, with a source and a drain, in addition, we also have a gate. The source is where the power flows in from, and the drain is where it ends up. The important aspect in these transistors is the gate. The gate can block the source and stop the current from flowing, now, we can open the gate by connecting to the positive, this will let the current flow, alternatively we can now connect to the ground to the close the gate again and stop the current from flowing.

To further illustrate how I control the speed we must now look at the wiring and implementation of fans and the transistor. As previously discussed the Fans need their own Power supply to drive them since the Arduino cannot provide 12v to the PWM fans its self. The wiring is as follows:

PWM Fan Positive connected to 12v Power supply Positive
PWM Fan Ground Connected to MOSFET Drain Pin (Middle pin)
MOSFET Gate Pin (Left Pin) Connected to PWM Pin on Arduino
MOSFET Source Pin (Right Pin) Connected to Arduino ground and power supply ground (this is important to establish a common ground, so we are able to control the flow the current)

Now that we have established a common ground and have a PWM pin connection to the gate of the MOSFET, we can now start writing to the pin our preferred duty cycles to control the fan speed. These duty cycles must be a percentage of 255. For example, 20% duty cycle would be 51.
This is how the system automatically controls the fans, depending on certain conditions, factors and settings the system will write different duty cycles to the gate allowing the system to slow and speed up the fans.



Software Implementation

Now that we have covered the hardware side of the implementation, we must now look at the software side, where we start configuring these modules to allow us to implement functionalities and create the base system for all of these components to operate on. The software phase is also where we start to gather all of this data that the modules can produce and make sense and use of it. This phase had a lot of risks of factors to consider before approaching it, having a reliable and efficient approach to the implementation means more organized code and project in the long run. Considering that, before I started writing the actual code for the system I wrote pseudo code for the requirements and experimented with different ways to doing things, this allowed me to get a better understanding of the components and hardware that I was working with. Writing the pseudo code for the requirements assisted the development greatly, this is because I was able to get an idea of how I was going to implement my functionality and work around possible problems and come up with solutions.

Now that we have installed our components we need to define our pins and implement functionality in the system. We can analyse the code and see how I have made use of the components to serve the objectives of the system.

We must firstly briefly understand how the code runs in Arduino. The code is composed of two parts, void setup() and void loop(). The Void setup() method only runs once when the device is switched on, we use this method to define pins and initiate components such as the RTC and DHT. Void loop() is the method that constantly runs in a loop, anything in the loop will be executed over and over again, this is an important aspect of Arduino to understand as we need manipulate the execution of the loop through the usage of Boolean states to set different screens.

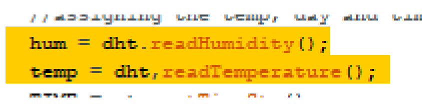
Fan Control

Fan Control is one of the most important functionalities, this is because the whole system is oriented around control the fans to increase and decrease the cooling of the room. As discussed earlier in the hardware side of the implementation, we use MOSFET N-Channel transistors to alter the speed of the fans. However, to achieve automation of this processes there are different means and methods of doing so. Fan Control in the system consists of different modes, the first mode is the default standard automatic Control. The Default automatic Control is based off two variables, the first is the current temperature, and the second is the target temperature & humidity. The target temperature is a value set by the user to identify a goal for the system. Depending on the goal of the system and the specified parameters (User settings) the system will alter the cooling differently. This mode is used as a basis for all the other types of cooling that the system offers.

Default Automatic Control

As discussed above, this is the bases and the default cooling mode of the system. As stated before this mode uses the Target temperature and the current temperature to figure out the cooling needed.

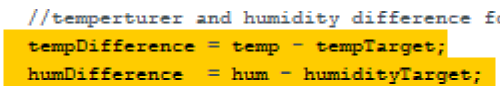
To achieve automation, we must base our decisions on the difference between the Current and target temperature & humidity. The system continuously checks for the temperature and humidity, this is done through the usage of the DHT22 Module. A float variable “temp” and “hum” are created for current temperature and humidity, Their value is assigned in the loop, since the loop method continuously loops, the value is recorded each loop and updated. To retrieve the data from the DHT sensor we use the DHT library and use the readHumidty() and readTemperature() Functions to assign the temp and hum variables their values.



```
//assigning the temp, day and
hum = dht.readHumidity();
temp = dht,readTemperature();
```

The target Temperature & Humidity is set through the use of the user interface, or wirelessly through the Android App. The exact process of how it is set will be covered later in the Interface section.

Now that we have our target values and current values, we can figure out the difference, and through the difference we are able to set different fan speed. This needs to be done in the loop so it can constantly work out the difference by subtracting the current temperature and humidity from the target values.



```
//temperturer and humidity difference forr automatic control
tempDifference = temp - tempTarget;
humDifference  = hum - humidityTarget;
```

Now that we have the difference in values we can change the speed of the fans. This is done through the use of analogWrite() function, which enables us to write the PWM duty cycle to the desired pin. This fan control is contained in a method called automaticFanControl(). This method contains the parameters at which the speed is changed, the parameter as discussed is the difference between the temperature & humidity and the targets.

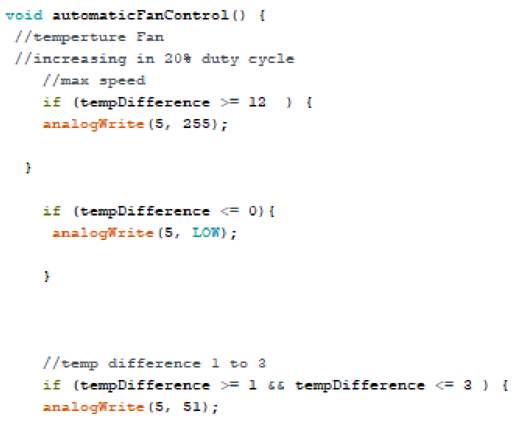
Each fan has separate control, one fan changes depending on the temperature difference and the other changes based on the difference in humidity. To assign the fans different power levels based on specific parameters we need to use if statements.

An if statement checks a specified parameter, if that parameter evaluates to true, then the contain code will execute, this is vital to the automation of the system, since we can specify how we want the system to behave based on different actions that the user takes or values that the system calculates.

The maximum difference is 12ºC for the temperature and 30% for the humidity. If the difference is greater than each of these values, then the then each that is responsible for either the temperature or humidity will be turned to maximum speed respectively. E.g. if temperature is greater than 12 then the temperature fan duty cycle will be set to 255 (100%). Or if the temperature difference is less than or equal to 0 then the fans are switched off since cooling isn't necessary.

We must also control the differences in between 0-12ºC for temperature fan and 0-30% for humidity fan. To do this we set power levels based on a boundary of difference.

For example, here we can see that if the temperature difference is between 1 and 3 the fan speed is set to "51" through the use of analogWrite() function, this is the 20% duty cycle (If we recall from a previous section, the duty cycle is worked out as a percentage of 255, therefor 20% of 255 is 51), as the temperature difference increase so does our duty cycles. Our fans increase in speed by 20% for each boundary that the difference value crosses. 1-3 = 20%, 3-5 = 40% 5-7 = 60%, etc.



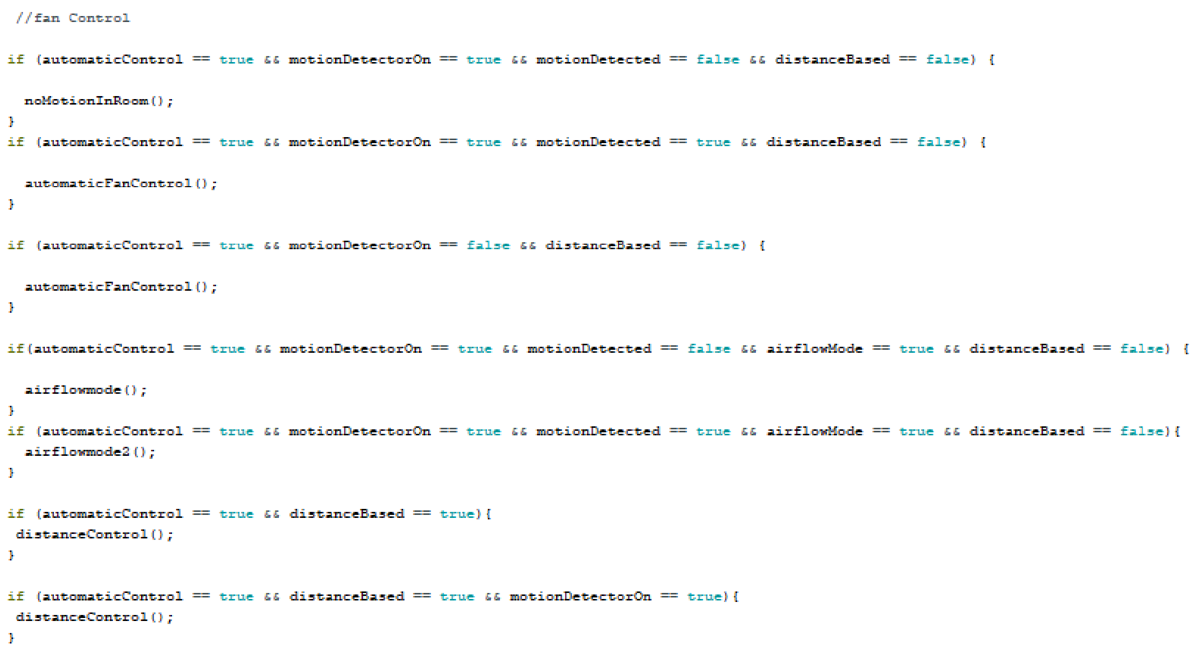
```
void automaticFanControl() {
 //temperture Fan
 //increasing in 20% duty cycle
    //max speed
    if (tempDifference >= 12  ) {
    analogWrite(5, 255);

  }

    if (tempDifference <= 0){
     analogWrite(5, LOW);

    }



    //temp difference 1 to 3
    if (tempDifference >= 1 && tempDifference <= 3 ) {
    analogWrite(5, 51);
```

The same principles are applied to the humidity fan, the changes of power are based on the boundaries that are specified.

This mode runs in default, and it is the bases for most of the control that goes on in the system. Now let’s look at how and when the system executes this.

To control when different modes are executed we need to make use of Boolean variables to create different states for the system, these states are not only used for the control of the variables but also for controlling the user interface. The Boolean values that are responsible for choosing which mode to operate are the settings variables. These settings Booleans are changed by the user through the use of the user interface display or through using the App.

This mode specifically is used when the user has either selected the standard default automatic cooling OR if the user has selected the Motion Based mode and there is movement in the room is detected (more details on the motion-based mode below). This is done through the use of if statements in the void loop() to continuously check the Boolean values, depending on which values are true and false the system will choose different type of cooling.



```
 //fan Control

if (automaticControl == true && motionDetectorOn == true && motionDetected == false && distanceBased == false) {

  noMotionInRoom();
}
if (automaticControl == true && motionDetectorOn == true && motionDetected == true && distanceBased == false) {

  automaticFanControl();
}

if (automaticControl == true && motionDetectorOn == false && distanceBased == false) {

  automaticFanControl();
}

if(automaticControl == true && motionDetectorOn == true && motionDetected == false && airflowMode == true && distanceBased == false) {

  airflowmode();
}
if (automaticControl == true && motionDetectorOn == true && motionDetected == true && airflowMode == true && distanceBased == false){
  airflowmode2();
}

if (automaticControl == true && distanceBased == true){
 distanceControl();
}

if (automaticControl == true && distanceBased == true && motionDetectorOn == true){
 distanceControl();
}
```

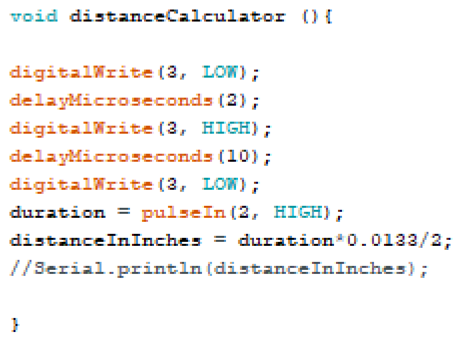
Distance Based

Distance based cooling is a method of cooling that uses the HC-SR04 Sensor. This mode is designed to allow the fans to change speed depending on how far an object is spotted from the sensor. As discussed earlier the sensor emits a sound and calculates the distanced based on how long it takes to comeback. So, in order to implement this functionality, we must first use the sensor to implement a distance calculator.

The distance calculator is a simple method created to calculate the distance between the object and the sensor. This method is called in the void loop(), this ensures that the sensor is constantly measuring the distance.

As discussed earlier, we need to emit an 8-cycle sonic burst through the Trig pin, this done by setting the state of the pin to HIGH for 10 microseconds, but to do so, we must first set the pin to LOW state to clear the pin and get it ready to use.

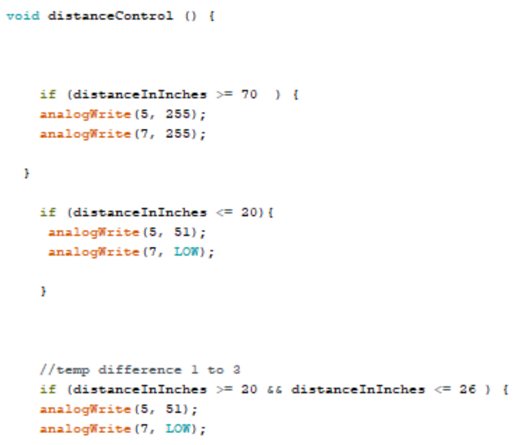
To do this, we use digitalWrite Function. We set the Sensor Trig pin (Pin 3) to high for 10 microseconds, and then set it to low. After that we use the pulseIn() function on the Echo pin (Pin 2) this function measurers the time that it takes for the sound to travel back, this is done by using pulseIn(2, HIGH), this is because when the sound wave reflects back to the sensor, the Echo pin will be in HIGH state, the pulseIn function allows you to measure the time it takes to bounce back when the pin state is HIGH allowing us to time the duration and assign it to a variable.



```
void distanceCalculator (){

digitalWrite(3, LOW);
delayMicroseconds(2);
digitalWrite(3, HIGH);
delayMicroseconds(10);
digitalWrite(3, LOW);
duration = pulseIn(2, HIGH);
distanceInInches = duration*0.0133/2;
//Serial.println(distanceInInches);

}
```

The distance is then calculated in inches by multiplying duration by 0.0133 and dividing it by 2 since we only need to know the distance for half the trip if the sound wave.

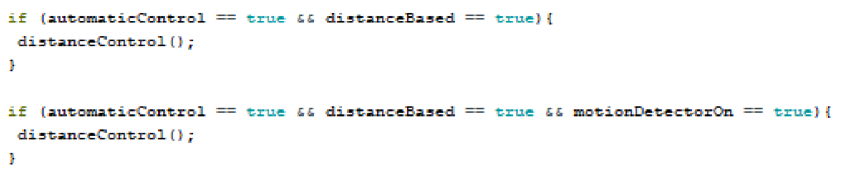
To implement the actual control of the fans based on the distance we use the same principles that we used for the Default automatic control. We create a method called distanceControl(), based on the user settings, if the user has chosen automatic fan control and distance-based control settings to be on then this method will be called.



```
void distanceControl () {


   if (distanceInInches >= 70  ) {
   analogWrite(5, 255);
   analogWrite(7, 255);

  }

   if (distanceInInches <= 20){
    analogWrite(5, 51);
    analogWrite(7, LOW);

   }


   //temp difference 1 to 3
   if (distanceInInches >= 20 && distanceInInches <= 26 ) {
   analogWrite(5, 51);
   analogWrite(7, LOW);
```

The distance Control method as stated controls the speed of the fans, this is done through the use of if statements to check the distance of the object from the sensor. Since we are working with humans and large distances it's better to measure the distance in inches. Depending the distance away in inches the object is from the sensor, this is again done through using if statements and boundaries of 6 inches.

For example, if the distance is between 30 & 36 then Fan 1 is on at 20% duty cycle and Fan 2 is off. Each boundary of 6 will increase the fan by speed by 20% duty cycle, if the fan at max speed then the same will be applied to fan 2, starting from 20% when the first fan is at max speed.

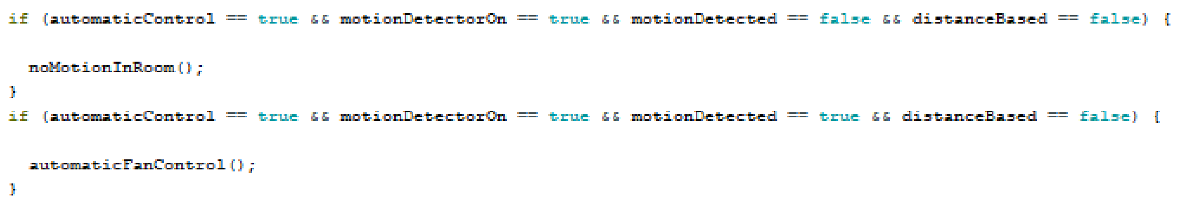


```
if (automaticControl == true && distanceBased == true){
 distanceControl();
}

if (automaticControl == true && distanceBased == true && motionDetectorOn == true){
 distanceControl();
}
```

Motion Based

Motion based control is an extension of the default automatic cooling process. The difference with motion based is that we are able to set the system to only cool the room if there is motion detected i.e. the user is present in the room. If there is no motion in the room and the temperature is greater than the target temperature then both fans are set to 40%, if the temperature is lower than the target then the fans are switched off.

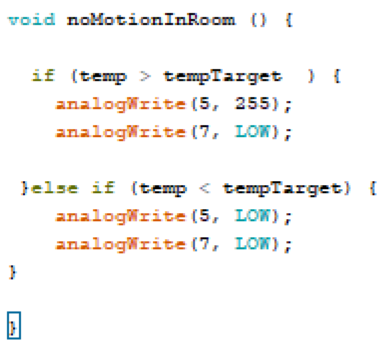
This is enabled by the use of the motion Detector that I implemented (this is covered below in the sensors section). In Void loop() we check for the state of the detector and for automatic control Boolean (The user chooses either manual or automatic cooling in the settings) by using an if statement. If automatic control on and there is motion detected, then we resume the default automatic cooling which is the automaticFanControl(() method. However, if there is no motion detected and automatic control is on then noMotionInRoom() method is called. It's important to note that in addition to the "automatic control" Boolean and "motion detected" Boolean, we also need "motion detector" Boolean to be true, this is the Motion based setting in the setting menu, the user must choose to enable the motion-based mode for it to function.



```
if (automaticControl == true && motionDetectorOn == true && motionDetected == false && distanceBased == false) {

  noMotionInRoom();
}
if (automaticControl == true && motionDetectorOn == true && motionDetected == true && distanceBased == false) {

  automaticFanControl();
}
```

The noMotionInRoom() method is responsible for choosing what to do when there's no motion detected in the room. If there is no motion in the room and the temperature is greater than the target, then only 1 of the fans is set on. If there is no motion in the room and the temperature is lower than the target, then both of then fans are switched off.

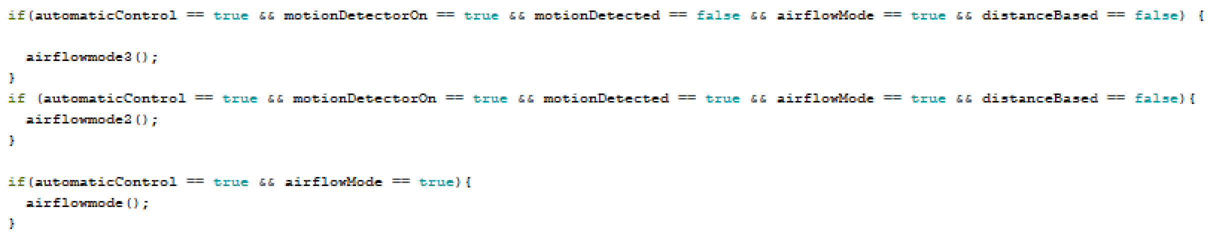
Airflow mode also operates in conjunction with this mode, the default automatic fan cooling mode and the distance-based method.



```
void noMotionInRoom () {

  if (temp > tempTarget  ) {
    analogWrite(5, 255);
    analogWrite(7, LOW);

 }else if (temp < tempTarget) {
    analogWrite(5, LOW);
    analogWrite(7, LOW);
}

}
```

Airflow Mode

Airflow mode changes the behaviour of the other cooling modes that are offered by the system. Airflow mode allows users to increase the cooling capacity of each of the modes. For default automatic cooling if the temperature is lower than the target then the fans are off, however, with airflow mode enabled even if the temperature and humidity drop below the targets, both of the fans will remain at 20% duty cycle. The airflow methods are called in the loop when the Booleans are satisfied in the if statements. Depending on which settings are chosen and if airflow mode is enabled either 1 of 3 airflow methods will be called.



```
if(automaticControl == true && motionDetectorOn == true && motionDetected == false && airflowMode == true && distanceBased == false) {

  airflowmode3();
}
if (automaticControl == true && motionDetectorOn == true && motionDetected == true && airflowMode == true && distanceBased == false){
  airflowmode2();
}

if(automaticControl == true && airflowMode == true){
  airflowmode();
}
```

Airflowmode 1

is the method that is called when we only have chosen the default automatic method in airflow mode. This method follows the same procedures as the default automatic cooling method, however, if the temperature drops below the target the fans will stay at 20% power.

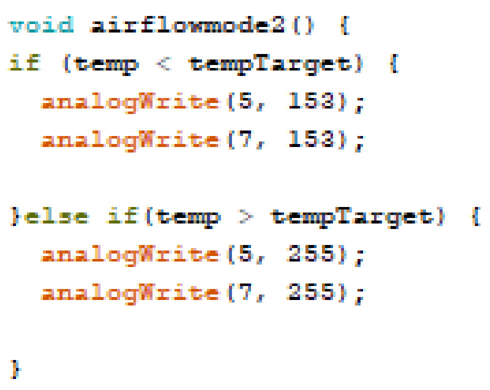
Airflowmode 2

This is the method that is called when we are using motion-based control and airflow mode is on. This mode is responsible for when there is movement detected in the room. If there is movement detected in the room, then both fans are operated at 100%. If the temperature is lower than the target however, then both fans are operated at 60%.
If there is no one detected in the room, then we cool the room down at a reduced rate.

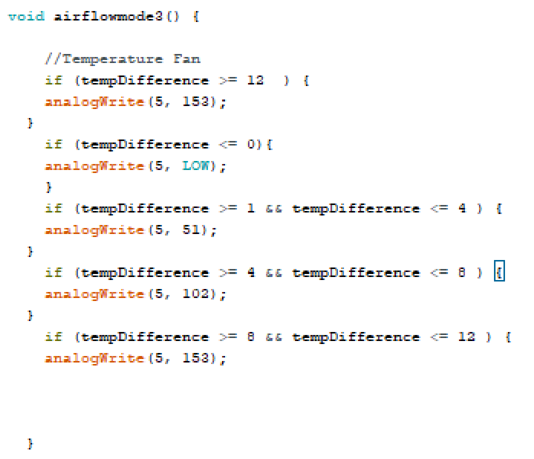
Airflowmode 3

This airflow mode, just like airflowmode2() is responsible for motion-based control. Originally with just motion Based Control, if the person is out of the room i.e. motion not detected, we either run one fan at 20% if the temperature is greater than the targets or turn them off ifs lower than target. With Airflowmode enabled, and Motion-based , the room can be still cooled down at a reduced rate. This is done just like the other methods of controlling the fan, through the use of if statements and analogWrite function to assign the duty cycle based on different parameters.

A



```
void airflowmode2() {
if (temp < tempTarget) {
  analogWrite(5, 153);
  analogWrite(7, 153);

}else if(temp > tempTarget) {
  analogWrite(5, 255);
  analogWrite(7, 255);


}
```



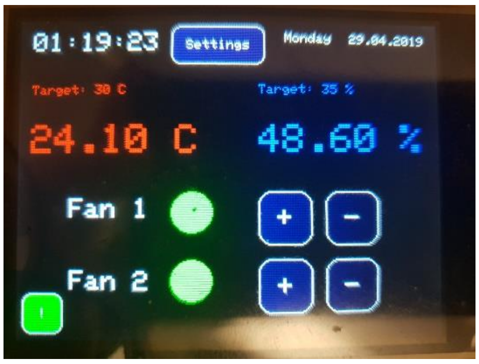
```
void airflowmode3() {

   //Temperature Fan
   if (tempDifference >= 12  ) {
   analogWrite(5, 153);
 }
   if (tempDifference <= 0){
   analogWrite(5, LOW);
   }
   if (tempDifference >= 1 && tempDifference <= 4 ) {
   analogWrite(5, 51);
 }
   if (tempDifference >= 4 && tempDifference <= 8 ) {
   analogWrite(5, 102);
 }
   if (tempDifference >= 8 && tempDifference <= 12 ) {
   analogWrite(5, 153);



 }
```

Airflowmode 4

This airflow mode is activated when we have distance based and airflow modes on. This allows the distance-based fan speed to reach maximum speed at a faster rate, by reducing the range needed for max power. This mode also keeps the fans on even if its lower than the minimum distance value.

Manual Control

As well as automatic control, manual control of the fans was also a necessity, we can't just have an automated system. There are two ways in which we are able to manually control the fan speed. The First is done Through the LCD Touchscreen display. The Home screen of the system contains buttons, 4 of which are Speed up and down buttons for Fan 1 and Fan 2. The second way is wirelessly through the App. Here we will cover the first method of manually controlling the fan.



To manually control the fan speed, automatic fan control setting needs to be turned off. If automatic Control setting is switched off, then in the void loop() we will write the variable "powerP" as the duty cycle for fan 1 and "PowerP2" for fan 2. These two variables are integers that increase when a button is pressed. This will increase the speed of the fans by a certain amount each button press.

```
if (manualControl && wirelessControl == false) {
  analogWrite(5, powerP);
  analogWrite(7, powerP2);
} else if(manualControl &&  wirelessControl ){
```

As previously discussed, we are able to alter settings through the touchscreen by checking if a region has been touched. Depending on which region is touched we increase the powerP and powerP2 values by 5 or decrease by 5 respectively.

```
//power up button for fan 1
if(homeScreenOn && manualControl && powerP < 255 && p.x>200 && p.x<230 && p.y>90 && p.y<122 ){
     powerP = powerP + 5;


}
//power up button for fan 2
if(homeScreenOn && manualControl && powerP2 < 255 && p.x>250 && p.x<290 && p.y>88 && p.y<122 ){
     powerP2 = powerP2 + 5;

}
//power down buttons for fan 1
if(homeScreenOn && manualControl && powerP > 10 && p.x>208 && p.x<236 && p.y>41 && p.y<81 ){
     powerP = powerP - 5;

}
//power down button for fan 2
if(homeScreenOn && manualControl && powerP2 > 10 && p.x>258 && p.x<298 && p.y>41 && p.y<81 ){
     powerP2 = powerP2 - 5;

}
```

User interface

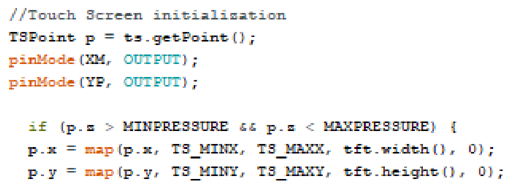
The user interface is extremely important as it allows us to control the system. The UI needs to be user friendly and allow ease of use. This is one of the main reasons that I chose to implement the system over an Android application; to offer different means of controlling the system.

The user interface is made up of two parts. First part is the LCD Display attached to the system. This is a capacitive touchscreen LCD display. The second part of the user interface is the mobile application. In both types of user interfaces, we are able to fully control the system

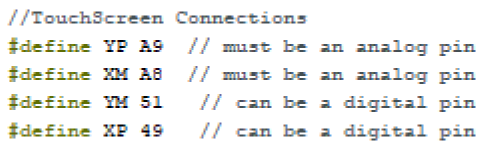
THE LCD display is programmed into the Arduino. TFT library offers multiple functions that allow us to create a comprehensive GUI. Some of these functions are tft.print, tft.drawRect, tft.Fill, tft.Draw and many other functions. The adafruit GFX library contains the core graphics of the display, this library is important for things such as buttons.

Since the system has more than one screen, I had to think about way to implement multiple different screens. To do this we can create different methods, and depending on which state we are In, we call the method of the screen. The states switch when the user presses the buttons leading to the other screens, this will change certain Boolean values that will allow the program to execute in a different way, this is required since Arduino code runs in a loop, so we need to control which section is selected to be run. Below I shall discuss how I implemented different aspects of the user interface, first let's start with touchscreen functionality.

Touchscreen **–** Our LCD Display offers touchscreen functionality; this functionality is important as it is one of the methods of control for the system. To implement touchscreen functionality, we must use the Y-, Y+, X-, and X+ Pinouts on the breakout board. This will allow us to define the pins and collect the incoming data from the pins. This is needed since we need the coordinates of the touch in order to specify which button is being pressed.



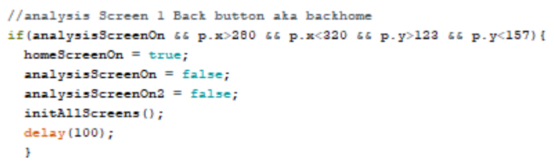
```
//Touch Screen initialisation
TSPoint p = ts.getPoint();
pinMode(XM, OUTPUT);
pinMode(YP, OUTPUT);

  if (p.z > MINPRESSURE && p.z < MAXPRESSURE) {
  p.x = map(p.x, TS_MINX, TS_MAXX, tft.width(), 0);
  p.y = map(p.y, TS_MINY, TS_MAXY, tft.height(), 0);
```



```
//TouchScreen Connections
#define YP A9   // must be an analog pin
#define XM A8   // must be an analog pin
#define YM 51    // can be a digital pin
#define XP 49    // can be a digital pin
```

Then we use the ts.getPoint() and map() function to retrieve the coordinates that are being pressed. This is extremely important, because through this we are able to recognise which area is being pressed, and through that we are able to create graphical buttons that respond to touch and change settings.

To implement functions to the buttons we use an if statement in the void loop() to check if the specified region that the buttons is under is pressed or not. Depending on which region is pressed and which screen is on a specific action is taken, for example if home screen is on (home screen state is set to true) and a touch is detected between x coordinates of 250 and 320 and y coordinate region of 123 and 157 then this will set the home screen state to true and analysis screen states to false, to allow us to return to the home screen



```
//analysis Screen 1 Back button aka backhome
if(analysisScreenOn && p.x>280 && p.x<320 && p.y>123 && p.y<157){
  homeScreenOn = true;
  analysisScreenOn = false;
  analysisScreenOn2 = false;
  initAllScreens();
  delay(100);
  }
```

Multiple Screens – To implement multiple screens, as stated previously, we need to make use of Boolean variables and create different states for the Arduino. To do this, we create the screens as normal methods in the program, we also declare two variables, one for declaring the state of the screen (whether it’s on or off) and one to initiate the screen (essentially refresh the LCD display so it can be updated when we are switching between screens). In the void loop() if the variable homeScreenOn is detected to be true, then the HomeScreen() method is called, inside the method the screen is only initiated if the variable to initiate the screen (initiateHomeScreen) is set to false, then inside the if statement we set the variable to true. What this essentially does is allow us to only initiate the screen once, rather than repeatedly running it, this would cause a flashing effect of the LCD screen.

```
void HomeScreen(){
  if(!initiateHomeScreen){
    tft.fillScreen(BLACK);
    tft.setRotation(3);
    initiateHomeScreen = true;
```

```
if(homeScreenOn){
HomeScreen();
```

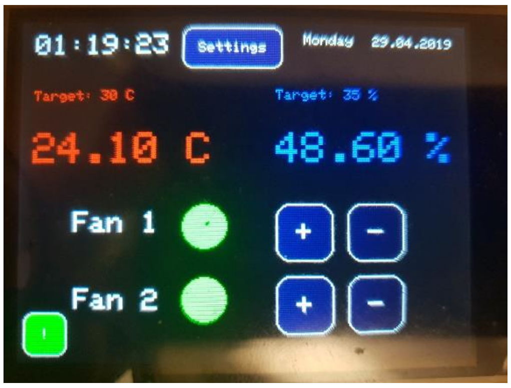
To initiate the screen, we use the initiateAllScreens() method, what this allows us to do is set all of the initiate variables to false, this means that whichever screen we change into, if we call this method (for example in the back home button), it would allow us to initiate any of the screens when we set the state of that specific screen to true.

```
  initiateHomeScreen = false;
  initiateSettingsScreen = false;
  initiateAlert = false;
  initiateAnalysisScreen = false;
  initiateAnalysisScreen2 = false;
  initiateSettingsScreen2 = false;
```

Home Screen

The home screen is what the user sees when the device is switched on, this is the main screen of the system. The screen contains 6 Buttons.

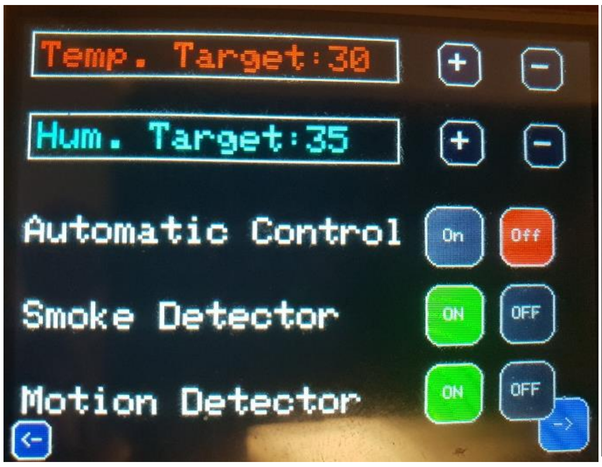
- Settings Button – Allows the user to navigate into the settings page, where the user is able to change system settings and set target temperatures and humidity.
- Power Up and Down Buttons – There 4 Buttons in total for controlling the fan speed manually. Power Up for fan 1 and for fan 2, and, power down for fan 1 and fan 2. Each time the button is pressed, the duty cycle is increased by 5.
- Analysis Button – This button allows the user to navigate to the analysis screen, where they are able to see the average temperatures of the week and also see the temperature and humidity level for each day for Day and Night.

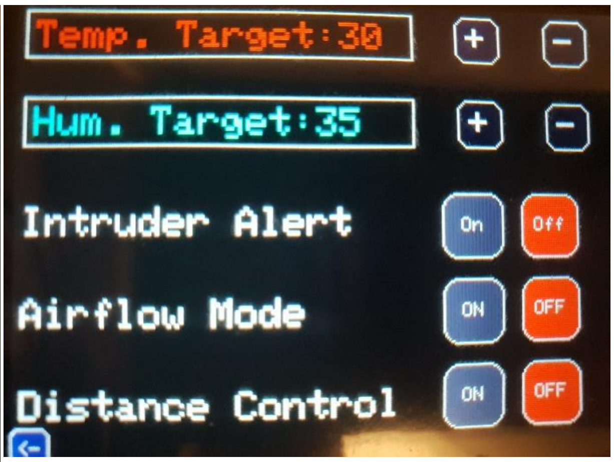


Settings Screen

The settings Screen is formed from 2 parts, this is because there are too many settings to fit on one screen, so the settings screen is actually two screens.

- Automatic Control Buttons – Allows you to choose between automatic control of the fans or manual control (OFF)
- Smoke Detector buttons – Allows you to turn the smoke detector on or off, for example if the user prefers the alarm off during the day but on during at night.
- Motion Detector buttons – This allows the user to turn on or off the motion-based control of the fans.
- Temperature and humidity Buttons – These 4 buttons allow you to either increase or decrease the targets that you have set.
- Intruder Alert Buttons – These allow you to either turn on or off the intruder alert.
- Air flow mode Buttons – Allow the user to set airflow mode on or off
- Distance Control – Allow the user to choose if they want distance-based cooling mode on or off.
- Back and forward Buttons – These buttons allow you to either go to the next screen (settings screen 2) or take you back to the home screen.

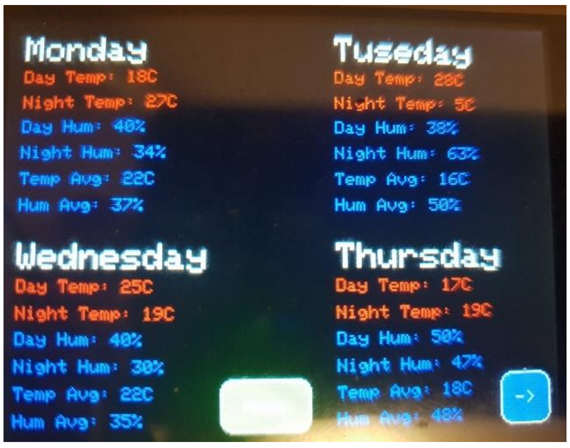

Temp. Target:30
Hum. Target:35
Automatic Control
Smoke Detector
Motion Detector


Temp. Target:30
Hum. Target:35
Intruder Alert
Airflow Mode
Distance Control

Analysis Screen

The Analysis screen, just like the settings screen is formed of two parts, this is because we have too much data to fit in on one screen. The analysis screen is where the user can view the average temperature and humidity of each day, and also look at the temperature and humidity levels of each day both at night time and day time.

- Home Button – This allows the user to navigate back to the home screen
- Next button – This allows the user to navigate to the second part of the analysis screen
- Back button – allows the user to go back to the first part of the analysis screen

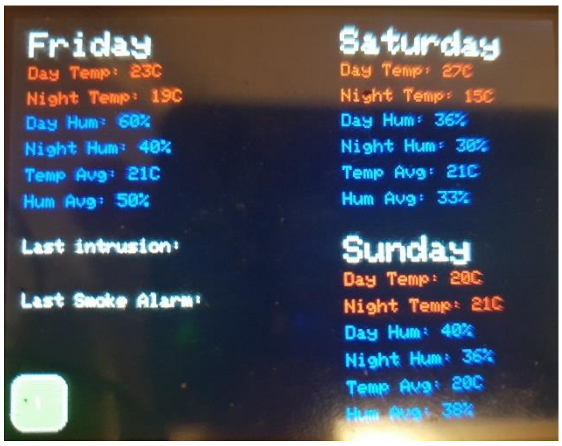


Monday
Day Temp: 18C
Night Temp: 27C
Day Hum: 40%
Night Hum: 34%
Temp Avg: 22C
Hum Avg: 37%

Tuseday
Day Temp: 28C
Night Temp: 5C
Day Hum: 38%
Night Hum: 63%
Temp Avg: 16C
Hum Avg: 50%

Wednesday
Day Temp: 25C
Night Temp: 19C
Day Hum: 40%
Night Hum: 30%
Temp Avg: 22C
Hum Avg: 35%

Thursday
Day Temp: 17C
Night Temp: 19C
Day Hum: 50%
Night Hum: 47%
Temp Avg: 18C
Hum Avg: 48%



Friday
Day Temp: 23C
Night Temp: 19C
Day Hum: 60%
Night Hum: 40%
Temp Avg: 21C
Hum Avg: 50%

Last intrusion:

Last Smoke Alarm:

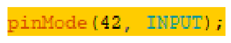
Saturday
Day Temp: 27C
Night Temp: 15C
Day Hum: 36%
Night Hum: 30%
Temp Avg: 21C
Hum Avg: 33%

Sunday
Day Temp: 20C
Night Temp: 21C
Day Hum: 40%
Night Hum: 36%
Temp Avg: 20C
Hum Avg: 38%

Sensors

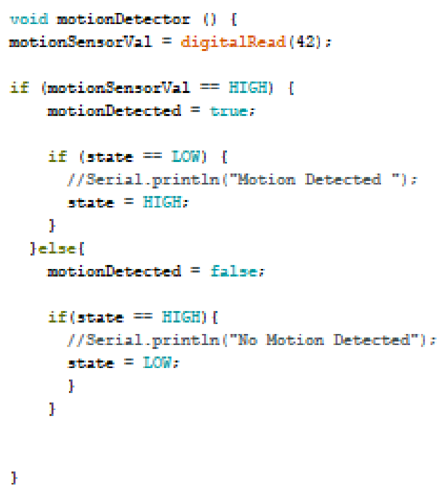
Motion Detector

The PIR motion sensor as we discussed earlier was a 3-pin module, with only 1 of them being used to receive data. Since we will be receiving data from the module we need to define the pin as in an Input Pin. This is done through the use of pinMode() function to initialize the pin. This is done in the setup() as shown in the Figure below



```
pinMode(42, INPUT);
```

Since we know how the PIR sensor works we just need to capture that information and store it somehow. To do this, I created a Boolean variable called "motionDetected", this Boolean value changes depending on the sensor readings that we receive from the PIR motion sensor. If we look at the snippet of code here we can see how it is done.



```
void motionDetector () {
motionSensorVal = digitalRead(42);

if (motionSensorVal == HIGH) {
    motionDetected = true;

    if (state == LOW) {
      //Serial.println("Motion Detected ");
      state = HIGH;
    }
  }else{
    motionDetected = false;

    if(state == HIGH){
      //Serial.println("No Motion Detected");
      state = LOW;
      }
    }



}
```

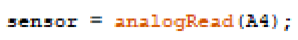
This motion Detector is a method that is continuously called in the void loop() every time that it is called it reads the digital pin 42 which is where our PIR sensor is connected to, this value is then assigned to a variable. This variable is checked, and if the sensor reading we get is equal to HIGH then we set our Boolean value of motionDetected to TRUE, else it will be false. Having this simple Boolean value of motion being detected or not will allow us to automate the system in many ways.

The nested if statement is used to set the state of the sensor, this state is stored on a different variable for testing and debugging purposes, the purpose of it being nested is that it allows the state variable to only be able to be assigned to HIGH if its LOW and to LOW only if its HIGH, this is because if there is movement detected you don't need to keep assigning HIGH to HIGH, it only needs to change between the two, this saves computation power and allows the system to be efficient, because after all, microcontrollers need to be efficiently used because of their limited power and memory.

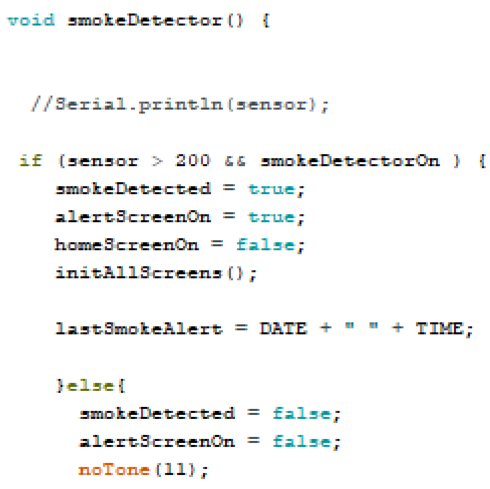
Smoke Sensor

The smoke sensor was also a straight forward implementation, since we already calibrated the sensor, most the job has already been done, all we need to do now is read the value of the sensor, to do this we must utilize the analogRead() Function.

The analogRead() function will return the sensor reading to us, we save this as a variable simply named sensor. If this sensor value crosses a certain threshold then (200) then "smokeDetected" variable will be set to true, this will allow us to sound the alarm. The system will also log the time and date of the occurrence. This is done by creating a String variable and assigning the current date and time to the variable when the threshold crosses the limit.



```
sensor = analogRead(A4);
```

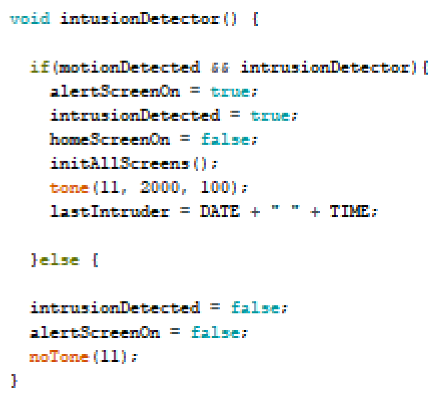


```
void smokeDetector() {


  //Serial.println(sensor);

 if (sensor > 200 && smokeDetectorOn ) {
    smokeDetected = true;
    alertScreenOn = true;
    homeScreenOn = false;
    initAllScreens();

    lastSmokeAlert = DATE + " " + TIME;

    }else{
      smokeDetected = false;
      alertScreenOn = false;
      noTone(11);
```

If the threshold drops below 200 then smokeDetected Boolean is again set to false and we use the noTone() function to cancel the buzzer tone on pin 11 which is the pin connecting the buzzer.

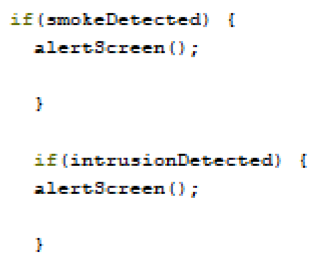
Smoke and intruder Alarm

Smoke and Intruder alarms are based off the PIR motion Detector sensor (For intrusion) and the MQ2 sensor (For smoke alarm). The intrusion detector is a method that Is called in the loop. This method relies on the motion detector readings, if there is motion detected and the intrusion detector is on, then intrusion detected variable will be set to true, this will allow us to trigger the alarm, similarly to smoke sensor we also log the last occurrence of intrusion to a string.

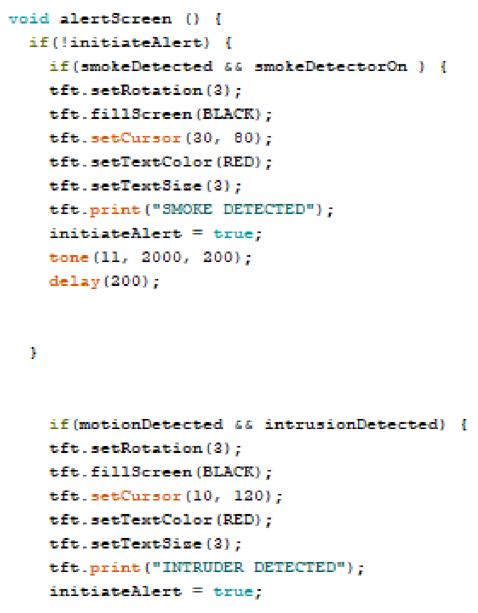


```
void intusionDetector() {

  if(motionDetected && intrusionDetector){
    alertScreenOn = true;
    intrusionDetected = true;
    homeScreenOn = false;
    initAllScreens();
    tone(11, 2000, 100);
    lastIntruder = DATE + " " + TIME;

  }else {

  intrusionDetected = false;
  alertScreenOn = false;
  noTone(11);
}
```

The alert screen is triggered through the loop when there is smoke detected, or if intrusion is detected. As previously discussed above in the User interface section, we trigger different screens based on Boolean values (states). In the alert screen, we check to see if we have an intrusion detection or smoke detection, based on which it is, a different tone is sounded, and flashing text is displayed.



```
if(smokeDetected) {
  alertScreen();

  }

  if(intrusionDetected) {
  alertScreen();

  }
```



```
void alertScreen () {
  if(!initiateAlert) {
    if(smokeDetected && smokeDetectorOn ) {
    tft.setRotation(3);
    tft.fillScreen(BLACK);
    tft.setCursor(30, 80);
    tft.setTextColor(RED);
    tft.setTextSize(3);
    tft.print("SMOKE DETECTED");
    initiateAlert = true;
    tone(11, 2000, 200);
    delay(200);


  }


    if(motionDetected && intrusionDetected) {
    tft.setRotation(3);
    tft.fillScreen(BLACK);
    tft.setCursor(10, 120);
    tft.setTextColor(RED);
    tft.setTextSize(3);
    tft.print("INTRUDER DETECTED");
    initiateAlert = true;
```

Android App

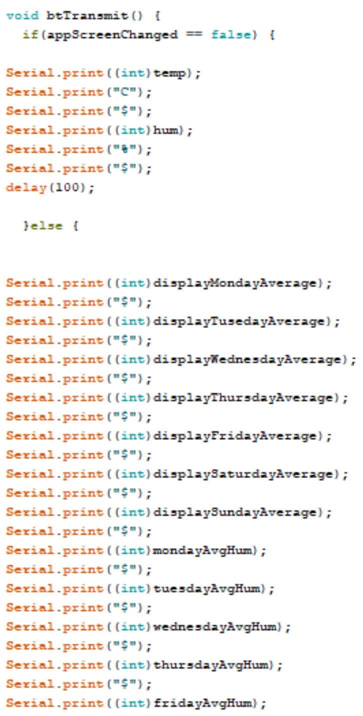
The android app is a vital part of the system. The application offers a versatile and easy wireless connection that allows the user to control and manipulate the system. The android application was made in MIT app inventor 2 which is a helpful tool used to develop and create Android applications. The Android app consists of two screens. The first screen is responsible for transmitting data (Controlling the system) whilst the second screen is responsible for retrieving data (Viewing the data gathered by the system). The purpose of splitting the screens up is to enable ease of management of serial communication. Depending on which screen is active, the Arduino will transmit different data to the application.

Wireless Control and communication

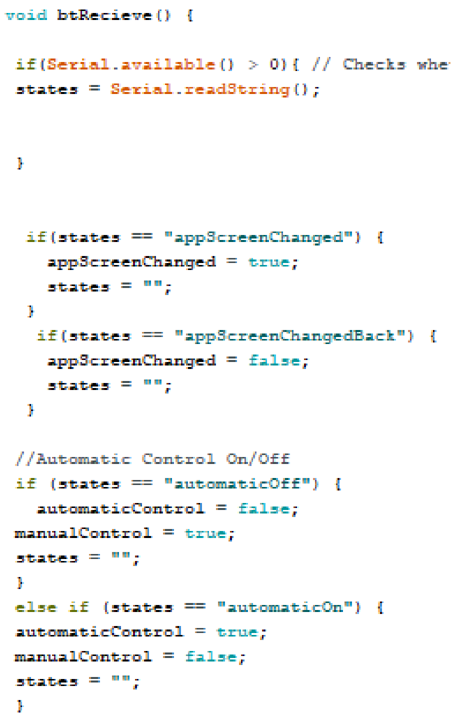
Wireless control and communication are established through two methods in the Arduino code. btTransmit() and btRecieve(). Each method retrieves and transmits different data.

btRecieve() method is called in the loop, this method will check if there is serial data available to receive, through the use of Serial.available() function. When there is data available, we will read the data and assign it to a string named “states”. Since we are sending text strings we must use the Serial.readString() function to retrieve the data.

The string that we receive is a predefined text string that we send from the App when a button is pressed, for example the OFF button for Automatic Control on the App will send the string “automaticOff”. In the method we have if statements to check the strings sent, depending on which string is detected, the system takes different actions. For example, when “automaticOff” is received, automatic control Boolean will be set to false and manual control Boolean to true. This enables us to control the settings of the system wirelessly without having to be next to it. This also encapsulates the second method of controlling the fan which I discussed in the previous section.



```
void btTransmit() {
  if(appScreenChanged == false) {

Serial.print((int)temp);
Serial.print("C");
Serial.print("$");
Serial.print((int)hum);
Serial.print("%");
Serial.print("$");
delay(100);

  }else {


Serial.print((int)displayMondayAverage);
Serial.print("$");
Serial.print((int)displayTusedayAverage);
Serial.print("$");
Serial.print((int)displayWednesdayAverage);
Serial.print("$");
Serial.print((int)displayThursdayAverage);
Serial.print("$");
Serial.print((int)displayFridayAverage);
Serial.print("$");
Serial.print((int)displaySaturdayAverage);
Serial.print("$");
Serial.print((int)displaySundayAverage);
Serial.print("$");
Serial.print((int)mondayAvgHum);
Serial.print("$");
Serial.print((int)tuesdayAvgHum);
Serial.print("$");
Serial.print((int)wednesdayAvgHum);
Serial.print("$");
Serial.print((int)thursdayAvgHum);
Serial.print("$");
Serial.print((int)fridayAvgHum);
```



```
void btRecieve() {

 if(Serial.available() > 0){ // Checks whe
 states = Serial.readString();


 }


  if(states == "appScreenChanged") {
    appScreenChanged = true;
    states = "";
  }
   if(states == "appScreenChangedBack") {
    appScreenChanged = false;
    states = "";
  }

 //Automatic Control On/Off
 if (states == "automaticOff") {
   automaticControl = false;
 manualControl = true;
 states = "";
 }
 else if (states == "automaticOn") {
 automaticControl = true;
 manualControl = false;
 states = "";
 }
```

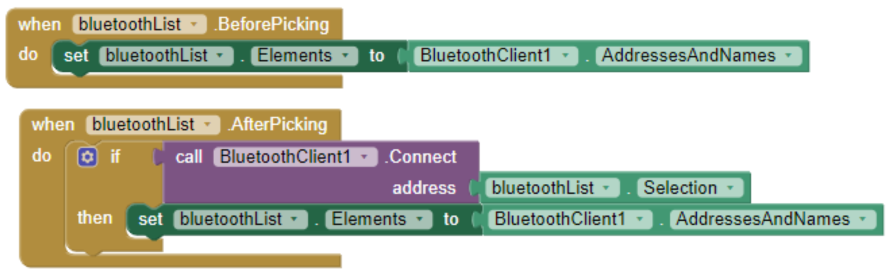
The btTransmit method transmits two sets of data. The set of data being transmitted depends on which screen is active. When a screen changes on the app, a string is sent through to the Arduino “appScreenChanged” or “appScreenChangedBack”, depending on which string the Arduino receives, it will set the appScreenChanged Boolean to either false or true. Now depending on this Boolean, we either transmit the temp and humidity values (when appScreenChanged == false. This means that Screen 1 on the application is active) or transmit the average temperatures of the week (when appScreenChanged == true. This means that Screen 2 is active). This is done to control the serial communication efficiently and enable a control for the flow of information.

Screen 1

The MIT app inventor offers an interface which allows you to create objects, buttons and many other elements easily and quickly. The Development platform offers different tools and functionalities to allow you to easily create the UI, as easily as dragging and dropping the element that you want to create.

The first screen consists of Buttons for turning the settings on and off, two buttons for each setting, ON and OFF. And also, buttons for setting the temperature and humidity targets. The first screen also displays live feed of the Current temperature and Humidity levels. Not only that but in the first screen you are also able to control the fan speed wirelessly, and update temperature and humidity targets.

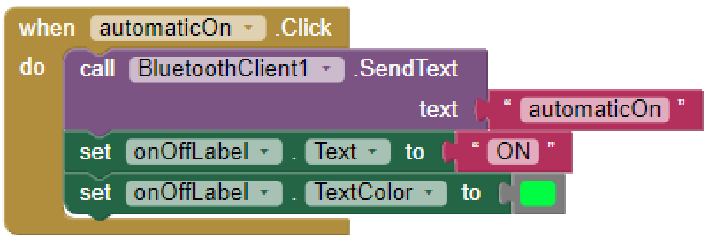
To allow all of this to take place wirelessly across two devices through a Bluetooth connection, we must establish a link between the App and the Arduino. To do this, we use Bluetooth client component on the MIT app inventor, this allows us to view and see all the paired devices on our mobile, not only that but it enables us to communicate to the Arduino through the use of its sending and receiving data functions. To establish a connection, we create a list, the elements of the list will be set to the Address and names of the Bluetooth client, after we pick a choice, we use the Bluetooth client to use one its functions which is the .Connect function to establish a connection to our desired paired device.



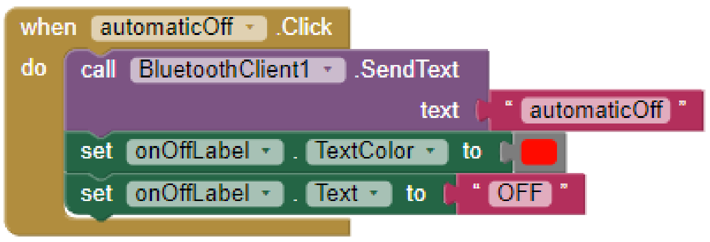
```
when bluetoothList .BeforePicking
do   set bluetoothList . Elements to BluetoothClient1 . AddressesAndNames

when bluetoothList .AfterPicking
do   if   call BluetoothClient1 .Connect
                              address bluetoothList . Selection
     then set bluetoothList . Elements to BluetoothClient1 . AddressesAndNames
```

As mentioned earlier, we control the Arduino by sending a String through serial communication, this text is sent by the App through the use of buttons, when a certain button is pressed a command is sent to the Arduino in a text format, the Arduino will then check if it’s a valid command and then act accordingly. When a button is pressed, we use the bluetoothClient.sendText function as shown in the figure below. This function allows us to send data through the Bluetooth client. The same principles are applied to each button in the system, each text string sent is different and it will result to a different command in the

Arduino system, as discussed earlier this enabled by the use of string checking to evaluate if a string sent matches any of the commands.

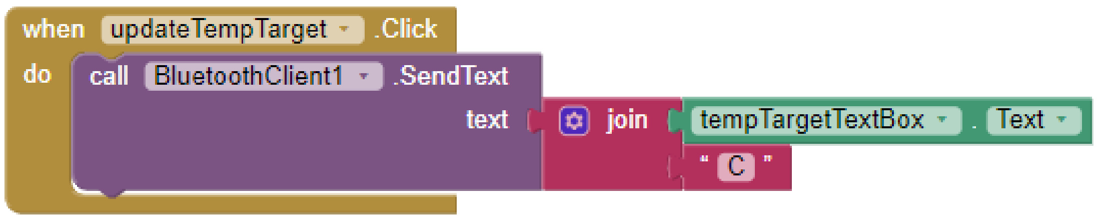


```
when automaticOn .Click
do  call BluetoothClient1 .SendText
                         text " automaticOn "
    set onOffLabel . Text to " ON "
    set onOffLabel . TextColor to 🟩
```

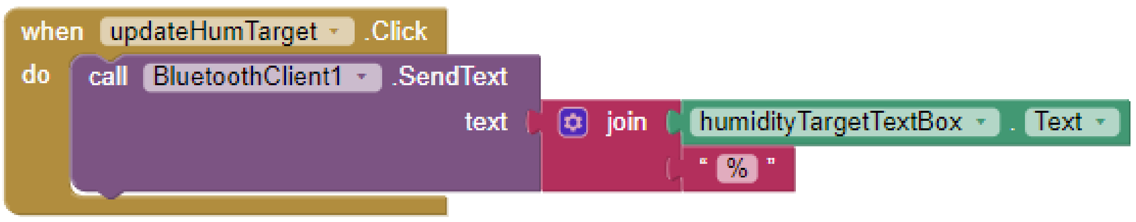


```
when automaticOff .Click
do  call BluetoothClient1 .SendText
                         text " automaticOff "
    set onOffLabel . TextColor to 🟥
    set onOffLabel . Text to " OFF "
```

To update our humidity and temperature targets we need to transmit the data to the Arduino, this is done by clicking the update Temp or Hum Button, when the button is pressed, we retrieve the data from the user text box, and send it using the bluetoothClient.sendText function. We also need to concatenate the user input with either “C” or “%”, this is extremely important.



```
when updateTempTarget .Click
do  call BluetoothClient1 .SendText
                         text join tempTargetTextBox . Text
                                   " C "
```



```
when updateHumTarget .Click
do  call BluetoothClient1 .SendText
                         text join humidityTargetTextBox . Text
                                   " % "
```

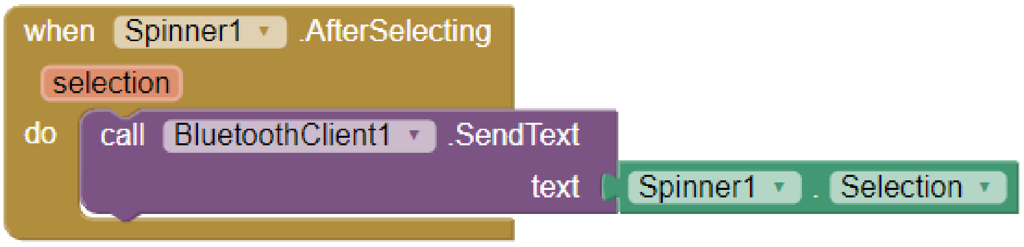
We add the characters “C” and “%” to allow the Arduino to recognise the input for temperature and humidity targets that the app is sending. In the btRecieve method we check if the string received, namely “states”` ends with either “C” or if it ends with “%”.

Using the endsWith() function we are able to specify if a string ends with a certain character. If the string ends with “C” then the string will be parsed as an integer using the .toInt() function and temperature target will be assigned that value.

```
if (states.endsWith("C") ) {
  tempTarget = states.toInt();
  initAllScreens();
  states = "";
}

if (states.endsWith("%") && states.startsWith("'") == false ) {
  humidityTarget = states.toInt();
  initAllScreens();
  states = "";
}
```

Similarly, if the string ends in “%” and **doesn’t** start with “ ‘ “ then the string will then be parsed, and humidity target variable will get assigned that value. The reason that we need to specify that it doesn’t start with a certain character is because we use that same character (“’”) to transmit a different string, namely the fan control string which allows us to wirelessly control the fan manually.

The wireless fan control is enabled by the use of a spinner, when the user chooses a selection (Between 0%, 20%, etc), the selection is sent through as a string , depending on the string we will receive, the variable “POWER” will be assigned different values. In order for this to take place the wireless mode needs to be on, namely the “wirelessControl” Boolean should be true. This is checked in void loop() and if it is indeed true then the power of the fans will be set to the variable “POWER”.



```
if(states == "'OFF") {

   POWER = 0;
  states = "";
 }

   if(states == "'20%") {

   POWER = 51;
   states = "";
 }
```

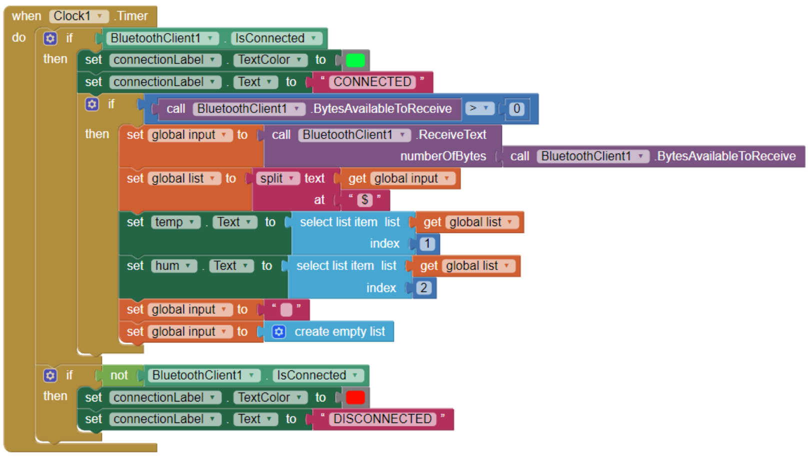
```
} else if(manualControl &&  wirelessControl ){
  analogWrite(5, POWER);
  analogWrite(7, POWER);


}
```

A live feed of the current temperatures and humidity is also sent to screen 1. As discussed in the previous section, depending on which screen is active we transmit different sets of data. To receive the two integers and differentiate between the two we use lists.

We make two labels; these labels get assigned the value of either temperature or humidity respectively. We use Serial. Print() function to send over a string, between each of the variables, we use a “$” to separate them.

Over in the App we create a list and assign the value of that list to the incoming stream of data, this is again done through the use of the Bluetooth client component. We then split the text using the split function and specify the function the split the string after the character “$” which we transmitted from the Arduino to separate the two.

Then temperature label is assigned to the first index which would be the temperature, and humidity is assigned the second index of the list which would be the humidity. This is done every time the clock components timer goes off, in our App its 1 second.

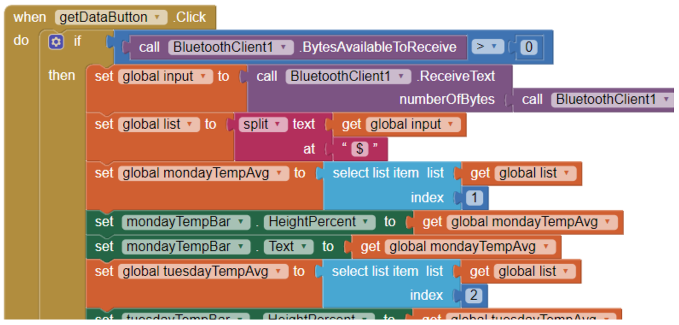


```
when Clock1 .Timer
do  if BluetoothClient1 . IsConnected
    then set connectionLabel . TextColor to [green]
         set connectionLabel . Text to " CONNECTED "
         if call BluetoothClient1 .BytesAvailableToReceive > 0
         then set global input to call BluetoothClient1 .ReceiveText
                                       numberOfBytes call BluetoothClient1 .BytesAvailableToReceive
              set global list to split text get global input
                                       at " $ "
              set temp . Text to select list item list get global list
                                                  index 1
              set hum . Text to select list item list get global list
                                                 index 2
              set global input to " "
              set global input to create empty list
    if not BluetoothClient1 . IsConnected
    then set connectionLabel . TextColor to [red]
         set connectionLabel . Text to " DISCONNECTED "
```

```
Serial.print((int)temp);
Serial.print("C");
Serial.print("$");
Serial.print((int)hum);
Serial.print("%");
Serial.print("$");
delay(100);
```

Screen 2

Screen two is responsible for displaying the data gathered by the Arduino, this is the average temperatures of each day of the week. To do this we use the same principle that we applied in the first screen to display the temperature and humidity live feed. We receive a string of data and use lists and text split functions to assign the data to its right variables in the app. We use the Bluetooth client component to check if there are bytes that are available to receive. To display the averages of temperature and humidity I use a Bar chart. This bar chart is a collection of buttons that are vertically aligned with an upwards growth, they are aligned at the bottom and displayed with the value number and the height of the bar is set to the value received at predefined indexes. i.e. Monday average temperature would be at index 1,whereas Monday humidity average would be at index 8. This happens when the user clicks the Update Temperatures button or the update humidity buttons on the screen.



```
when getDataButton .Click
do  if call BluetoothClient1 .BytesAvailableToReceive > 0
    then set global input to call BluetoothClient1 .ReceiveText
                                  numberOfBytes call BluetoothClient1
         set global list to split text get global input
                                  at " $ "
         set global mondayTempAvg to select list item list get global list
                                                      index 1
         set mondayTempBar . HeightPercent to get global mondayTempAvg
         set mondayTempBar . Text to get global mondayTempAvg
         set global tuesdayTempAvg to select list item list get global list
                                                       index 2
```

```
Serial.print((int)displayMondayAverage);
Serial.print("$");
Serial.print((int)displayTusedayAverage);
Serial.print("$");
Serial.print((int)displayWednesdayAverage);
Serial.print("$");
Serial.print((int)displayThursdayAverage);
Serial.print("$");
Serial.print((int)displayFridayAverage);
Serial.print("$");
```

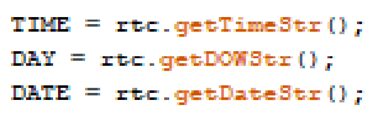
Saving Data (EEPROM) and Real Time Clock

Saving data is an extremely important part of this project, this is because we need to save the user settings, such as target temperature and also save the data that the Arduino gathers, this is the temperature and humidity levels of the days.

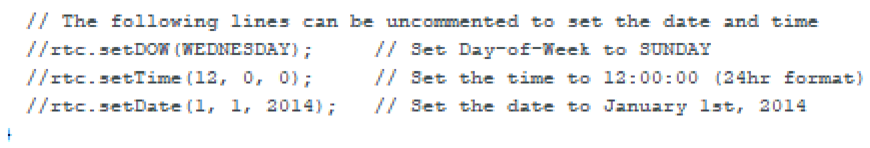
To do this we needed two include two things, the first is a way for the Arduino to recognise what time it is, and for a place to store the data gathered.

Real Time Clock

As previously discussed we use an RTC DS3231 module to keep time on the system, this clock keeps ticking even the module is off. But first we must set the date and time to initialize it, we only do this once when we first use it. After we initialize it, we can make two variable strings named DAY, DATE, and TIME. This will give us the current time and date in a variable which we can check.

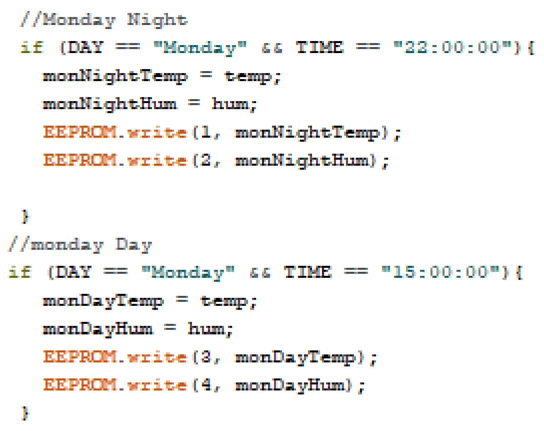


```
TIME = rtc.getTimeStr();
DAY = rtc.getDOWStr();
DATE = rtc.getDateStr();
```



```
// The following lines can be uncommented to set the date and time
//rtc.setDOW(WEDNESDAY);     // Set Day-of-Week to SUNDAY
//rtc.setTime(12, 0, 0);     // Set the time to 12:00:00 (24hr format)
//rtc.setDate(1, 1, 2014);   // Set the date to January 1st, 2014
}
```

The reason that we need to have the time and date in a string is to be able to check it against a parameter in an if statement, this is to capture the current temperature and humidity on a set time. For example, if the day is Monday and the time is 22:00 then we capture the temperature and assign it to a variable. This will allow us to capture the temperatures and work out averages through that to display on the analysis screen.

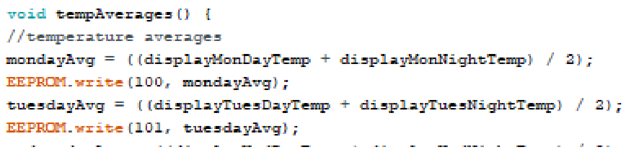


```
 //Monday Night
 if (DAY == "Monday" && TIME == "22:00:00"){
   monNightTemp = temp;
   monNightHum = hum;
   EEPROM.write(1, monNightTemp);
   EEPROM.write(2, monNightHum);

 }
//monday Day
if (DAY == "Monday" && TIME == "15:00:00"){
   monDayTemp = temp;
   monDayHum = hum;
   EEPROM.write(3, monDayTemp);
   EEPROM.write(4, monDayHum);
 }
```

Saving Data (EEPROM)

Saving data is done through the use of EEPROM library. The Arduino already has EEPROM which is small storage space that is available to use, we can write bytes to an address on the EEPROM and retrieve the data using the EEPROM.write() and EEPROM.read() functions.

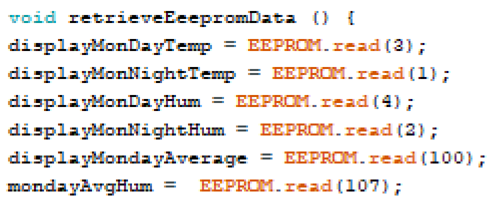
Saving data and retrieving data to the EEPROM in my program is split into two parts. Data is written to an address in many locations, for example, tempAverages() method writes the average temperatures, whereas the rtcAnalysis() method writes the temperatures and humidity captured throughout the day.



```
void tempAverages() {
//temperature averages
mondayAvg = ((displayMonDayTemp + displayMonNightTemp) / 2);
EEPROM.write(100, mondayAvg);
tuesdayAvg = ((displayTuesDayTemp + displayTuesNightTemp) / 2);
EEPROM.write(101, tuesdayAvg);
```

The important method to consider here is the retrieveData() method. This Method allows us the retrieve the data that is stored in the EEPROM. This is done through the use of EEPROM.read() function. This method is called in the void loop() section of the program, so each time the loop is executed, the EEPROM data is updated.

This is important, since when we boot the system up after it has been shut down, the actual variables are empty, we need to assign the values to the variables accordingly again.



```
void retrieveEeepromData () {
displayMonDayTemp = EEPROM.read(3);
displayMonNightTemp = EEPROM.read(1);
displayMonDayHum = EEPROM.read(4);
displayMonNightHum = EEPROM.read(2);
displayMondayAverage = EEPROM.read(100);
mondayAvgHum =  EEPROM.read(107);
```

Testing

Testing is an extremely important phase. Testing ensures that our system is operational and that it is indeed in fact ready for deployment. The testing carried out should reflect on the requirements captured and intended for the project. Through do so, it allows us to validate our final system against the intended objectives of our project.

There are different types of testing, any testing phase should be comprised of more than 1 type of testing. In this section I shall define these types of testing and also illustrate my testing process.

Functional and Non-Functional Testing

There are two distinct categories of testing, Functional and Non-Functional, each category has different types of tests such as Unit Testing, integration testing, security testing, performance testing, etc (Aebersold, 2019)

Functional Testing focuses on testing the system against our captured requirements, this category of testing focuses on ensuring that all components of the system behave as expected towards their intended purpose. Functional Testing encompasses Unit Testing, Integration Testing, System Testing, and Acceptance Testing (Aebersold, 2019).

Unit Testing – Testing performed at the lowest level. This encapsulates testing of individual components of the software at the code level, to ensure they are functional (softwaretestingfundamentals.com, 2019).

Integration Testing – After each unit is tested, the units are combined together to form modules that operate together, these modules are tested to see if they function as required together (Aebersold, 2019).

System Testing – The system is tested as a whole in a black box manner, this means that we use the system as a user does; evaluating the user side performance and reliability of the system (Software Testing Fundamentals, 2019).

Acceptance Testing – Last phase of the functional testing category this decides whether the system as a whole is ready for deployment or not (Software Testing Fundamentals, 2019).

Non-functional testing focuses on the operation of the system rather than the functionality and code side of the system. This category includes 4 distinct types of testing. Performance Testing, Security Testing, Usability Testing, Compatibility Testing (Aebersold, 2019).

Performance Testing – performance testing focuses on finding out how the system will behave when put under different loads of usage. This includes Load Testing, Stress Testing, Endurance Testing, and Spike Testing (Rouse, 2018).

Security Testing – This testing is focused on the security of the system against .

Usability Testing – usability Testing focuses on evaluating if the system is easy to use. This is done by using real users to test the system (Usability, 2019).

Compatibility Testing – This focuses on the compatibility of the system with other devices and evaluate how the system behaves in different environments.

Test Results

To ensure my system was reliable I had to carry out both functional and non-functional types of tests. This is to ensure that all aspects of my system are stable and operating as expected. Below is a table detailing all the tests that I carried out. The table contains the following details:

- Test **–** Functionality or procedure that is being tested
- Description **–** Description of the Test
- Expected Outcome **–** The expected behaviour that reflects our requirements
- Actual Outcome **–** The Actual outcome of the test
- Pass/Fail **–** If the test outcome was a failure or success

The second table details the type of testing it is. The table contains the following information:

- Test – The name of the test
- Functional/Non-Functional – Whether the test was a functional or non-functional category test.
- Type – the type of test that it is, e.g. Unit, Integration, security etc.
- Form – The form of the test, either Black Box or White Box.

Test	Description	Expected outcome	Actual Outcome	Pass/Fail
Default automatic control	This is the default automatic cooling mode. This cooling is based on Current temperature and the target temperature that the user has set.	The fan speed should adjust automatically when the temperature rises up and down.	The fan speed changes accordingly. If there is an increase in temperature, or decrease, the difference is calculated, and the duty cycle of the fans change.	Pass
Motion Based Control	This is the motion-based control, depending on whether the user is in the room or not, the system will engage different cooling modes.	Case 1: If the user is in the room, then default automatic cooling should resume. Case 2: If user is not in the room and room *doesn't require cooling then Fans should be* off. Case 3: If user is not in the room and it does require cooling, then fans are cooling the room down at a much slower rate.	Case 1: The system resumed default cooling as anticipated. Case 2: The fans successfully switched off when required to. Case 3: The fans cool the room down at a lowered rate when it requires cooling, but no one is in the room.	Pass
Distance Based Control	This is the distance based cooling method, depending on how far away you are from the fan, the speed of the fan should change	Depending on where the user is standing, the speed of the fans should increase or decrease respectively.	The fans correctly adjusted their speed automatically. Longer distances resulted faster speed and shorter distances resulted in lower speed.	Pass
Manual Fan control	This is the functionality that allows the user to change speed of the fans manually through the use of the LCD display. 4 Buttons in total, buttons Down and Up for each fan.	Case 1: When the user holds the + button, the fan speed of should gradually increase. Case 2: When the user Holds the – Button down, the speed should gradually decrease.	The fans successfully Sped or slowed down depending of which button we pressed.	Pass
Wireless Manual Fan Control	Functionality responsible for manually controlling the fan speed wirelessly.	When the Wireless Mode is on, and automatic control is off, the user should be able to set the speed manually from the android application	The Android application successfully changes the speed of the fans on the Arduino when the requirements are met (wireless mode on, and automatic control off)	Pass

Airflow Mode	This mode is responsible for changing how the other 3 modes work. Airflow mode interacts with each other mode differently	Case 1: Default automatic Cooling in airflow; *the fans shouldn't turn off when cooling isn't required. And top speed of fans is* reached at a lower temperature & humidity difference between Target and current. Case 2: Motion Based Control in Airflow Mode. If *the user isn't in the room the fans should cool the room at a lower rate, If it doesn't* require any cooling then they are off. If the user is present in the room and the room requires cooling then fans are on max speed, if *it doesn't require cooling, then fans are set at* 60%. Case 3: Distance based in airflow mode; *fans shouldn't switch off even if its below* minimal range. Fans should also reach maximum speed at a smaller distance.	Case 1: The fans behaved as they shou*ld, they didn't switch off and also* reached top speed faster than normal mode. Case 2: The fans were operating as they should; depending on whether the user is in the room or not, or if the room need cooling or not. Case 3: The distance-based mode in airflow was operating as it should, reaching a faster speed at a lower distance and the fans not switching off at lower than minimum distance.	Pass
Navigation Buttons	The GUI Buttons on the LCD display are responsible for the navigation through the system.	Each button should lead to its right destination, i.e. Home button should take the user back home.	One button failed the tests, this was the settings screen 2 backwards button. This button is supposed to take back to Settings screen 1 but instead it takes the user back home.	Fail
Temperature and humidity data capture	This is the functionality that is responsible for storing the data, this allows us to calculate averages and display statistics of past temperatures.	At predefined set times, the system should capture the temperature & humidity of each day at both day and night time.	The system successfully captures the temperature and humidity each day both at night and day time.	Pass
Analysis Screen	This screen is responsible for displaying the past temperature & humidity levels and the averages.	The analysis screen should display the past and average temperature and humidity levels. It should also update in real time when new data is captured.	The analysis screen successfully displays all of the data captured. It also displays the averages and updates in real time when new data is captured.	Pass

Wireless Control	Here we test the wireless control of the system. This is done through using the app and it allows the user to control the whole system.	The user should be able to control all of the settings using the android application. The user should be able to set the targets wirelessly also.	The system can be fully controlled from the android application, each button pressed correlates to its right command and the user is able to easily update targets.	Pass
Smoke and Intruder alert	Functionality responsible for alarming us when there is smoke or an intruder.	Smoke alarm should go off only if it is turned on in the settings, and only if the smoke sensor value crosses a certain threshold. Intruder alarm should only go off if it is turned on in the settings, the intruder alert should render the system not accessible via the LCD for security measures. The user should be able to use the android application to unlock the system again by turning the intrusion detector off.	The smoke alarm operates as intended. The intruder alert operates as intended, it locked the system down and was unlockable from the phone.	Pass
Real time clock	Functionality for keeping real time. Needed for data capture.	The real time clock should keep ticking even if the system is off.	The real time clock worked as intended; keeping time even whilst switched off.	Pass

Test	Functional / Non-Functional	Type	Form
Default Automatic Control	Functional	Integration	Blackbox
Motion based control	Functional	Integration	Blackbox
Distance Based Control	Functional	Integration	Blackbox
Manual Fan Control	Functional	Integration	Blackbox
Wireless manual fan control	Functional	Integration	Blackbox
Airflow Mode	Functional	Integration	Blackbox
Navigation buttons	Functional	Unit	Whitebox
Temperature and Humidity data capture	Functional	Unit	Whitebox
Analysis Screen	Functional	Integration	Whitebox
Wireless Control	Functional	Integration	Blackbox
Smoke and Intruder Alert	Functional & Non-Functional	Integration & Security Testing	Blackbox
Real Time Clock	Functional	Unit	Whitebox

Final System Evaluation

Now that we have completed the implementation of the project, we can evaluate how well the system meets the original intended objectives of the system and how the implemented functionality reflects off the captured requirements.

The intended objective of the project was to create an automated room cooling system that requires minimal interaction and effortless accessibility to the system. The system needs to offer a smart a range of settings that allows the user to customize their preferences. Through choosing these preferences, the system can intelligently make decisions and automate the process of cooling.

The system I developed successfully meets the intended objectives we set out with. The system offers a wide range of settings(Automatic Control, Manual Control, Wireless Control, Default automatic Cooling, Motion-Based Cooling, Distance Based Cooling) not only that but the system also offers Airflow mode. This mode changes the way that 3 main methods of cooling operate, so essentially, we have 6 different ways of automating the process of cooling.

This means the user has wide range of choice and preferences, which allows the system to operate flawlessly within any environment. As stated, the system needs to be easily useable and accessible, this was achieved by the use of Bluetooth technology and through a mobile app that I created which allows the user to view data gathered by the Arduino as well as fully control the system.

The system also met every single requirement that was captured in the early stages of the project, we can use a table here to look at the met requirements. This was thanks to careful planning and research, correct development and project management. However, although the system met all of the requirements, one of the objectives was to provide analysis of the data gathered. Our system does indeed gather and display some statistics such as Temperature & humidity level from all of the days at different times, I wanted to have a smarter analysis where the system can suggest to the user what mode to use based on past data. Although Arduino is extremely limited when it comes to AI capabilities because of its limited processing power, this would enable the system to display a more prominent AI like behaviour.

Requirement	Priority	Met / Not Met
Automatic Fan Control	5	Met
Manual Fan Control	5	Met
Wireless Fan Control	4	Met
Set Target Temperature and humidity	5	Met
Intruder Alert	3	Met
Smoke Alert	2	Met
Motion based Control	3	Met
Distance Based	3	Met
Airflow Mode	5	Met
Analysis	4	Met
Real Time Clock	2	Met
LCD Touchscreen	5	Met
Android Application	5	Met
Saving User Data	5	Met

Future Plans and Experimental Work Done

Even though the project has come extremely far, I think there is still far more room for improvements, upgrades and additionally functionality and expansions. The nature of the system opens up a wide range of opportunities since there are so many digital devices, objects and components and sensors that can function and interact with an Arduino.

Here I will discuss the future plans I have for expanding the project further than it is now. This system was designed to be a cooling system; however, the system can be expanded into both a cooling system(both Fan and AC) and a heating system. This can be done with the use of Peltier modules, which is also known as Thermoelectric cooling. The Peltier module cools down and heats up by transferring heat from one side to the other, allowing one side to be cold and one to be hot. Through the use of heatsinks and fans located at each side, we can choose to draw either hot or cold air.

I experimented with Thermoelectric cooling at the starting phase of my project, however, I had problem when trying to use an N-Channel MOSFET Transistor to control the PWM value of the Peltier. Since I didn’t fully understand the power consumption and output of the Peltier module I ended up burning the transistor and causing one of pins on the Arduino to get fried.

Another possible feature to implement In the future would be through web control, where the user is able to control the system through a webpage. This could be done through using Wi-Fi technology with the Arduino (Arduino, 2019). This would mean that we can control the system even when we’re outside of the house, this could also substantially increase the effectiveness of the intruder alarm as it would now be able to warn you of intrusions when you’re not actually present at home, the same principle applies for the smoker alarm.

To further expand the system, we can increase the number of fans that operate. This can be easily done by just attaching the Fans to the power supply and either feeding the ground pin into the first or second transistor, allowing us to control 2 sets of fans. This means that we get an increase in cooling power and speed up the cooling process, thus increasing the effectiveness of the system

Future improvements and possible upgrades to a system is extremely vital, this is to ensure that the system is kept up to date, and always improving. From a business point of view, it’s important to retain customers, but from a development point of view it is also important to maintaining a robust and successful system.

Conclusion

Throughout this report, we were able to explore the project from start to finish, looking between the finer lines of how the system operates and how it evaluates the settings into a decision-making bases that automates the cooling process and provides a form of security and health measures. The report covered many aspects of the developing the system, from the literature review chapter where we look at relevant technologies to the hardware and software implementation chapters where we look at the intricate details of the components and modules, the science behind them and also look at the software programming operating them.

The main focus of the project was oriented around the Arduino microcontroller, and not the android application. This is because the android application is just a simple wireless control program, whereas the Arduino is where the data is gathered, processed and where the system is controlled from. The mobile application is an extension of the Arduino that's required to achieve full functionality and objectives.

I think, overall, the project has been very rewarding in knowledge and has taught me many new skills and allowed me to explore tools that can help me in any future project. I also think that the project was successful in achieving all the requirements that were captured. Furthermore, the intended objectives of the system almost completely matched the finalized system which leads me to believe that this was evidently a successful project.

Many different aspects played a role in the success of the project. Key factors such as Testing, are vital to ensuring a high-quality system, and also project planning; where it is vital to ensure you have a plan with set start and finish dates to meet crucial deadlines. We also explored different ways in which we can improve the system in the future, given the possible options we have, the project could be developed into something larger, more effective and better. Software development life cycle was also a very important aspect that we considered, as it allowed us to plan the development and validation of the software implementation phase.

References

Currey, M. (2019). HC-05 and HC-06 zs-040 Bluetooth modules. First Look | Martyn Currey. [online] Martyncurrey.com. Available at: http://www.martyncurrey.com/hc-05-and-hc-06-zs-040-bluetooth-modules-first-look/ [Accessed 6 Mar. 2019].

Gudino, M. (2017). Arduino Uno vs. Mega vs. Micro. [online] Arrow.com. Available at: https://www.arrow.com/en/research-and-events/articles/arduino-uno-vs-mega-vs-micro [Accessed 9 Mar. 2019].

Huang, B. (2019). Logic Levels. [online] Learn.sparkfun.com. Available at: https://learn.sparkfun.com/tutorials/logic-levels [Accessed 15 Mar. 2019].

maximintegrated.com. (2019). [online] Available at: https://datasheets.maximintegrated.com/en/ds/DS3231.pdf [Accessed 9 Mar. 2019].

Circuitdigest.com. (2019). What is the difference between microprocessor and microcontroller? [online] Available at: https://circuitdigest.com/article/what-is-the-difference-between-microprocessor-and-microcontroller [Accessed 17 Mar. 2019].

Choudhary, H. (2019). Difference between Microprocessor and Microcontroller | Microcontroller vs Microprocessor. [online] Engineersgarage.com. Available at: https://www.engineersgarage.com/tutorials/difference-between-microprocessor-and-microcontroller [Accessed 19 Mar. 2019].

Arduino.cc. (2019). Arduino - Compare. [online] Available at: https://www.arduino.cc/en/products.compare [Accessed 21 Mar. 2019].

GitHub. (2018). Arduino build Process. [online] Available at: https://github.com/arduino/Arduino/wiki/Build-Process [Accessed 24 Mar. 2019].

Medium. (2017). https://medium.com/globalluxsoft/5-popular-software-development-models-with-their-pros-and-cons-12a486b569dc. [online] Available at: https://medium.com/globalluxsoft/5-popular-software-development-models-with-their-pros-and-cons-12a486b569dc [Accessed 22 Mar. 2019].

Wikipedia.org. (2019). App Inventor for Android. [online] Available at: https://en.m.wikipedia.org/wiki/App_Inventor_for_Android [Accessed 27 Mar. 2019].

Wikipedia.org. (2019). Fritzing. [online] Available at: https://en.m.wikipedia.org/wiki/Fritzing [Accessed 28 Mar. 2019].

Medium. (2017). 5 Popular Software Development Models with their Pros and Cons. [online] Available at: https://medium.com/globalluxsoft/5-popular-software-development-models-with-their-pros-and-cons-12a486b569dc [Accessed 29 Mar. 2019].

cprime.com. (2019). What is AGILE? | What is SCRUM? |. [online] Available at: https://www.cprime.com/resources/what-is-agile-what-is-scrum/ [Accessed 5 Apr. 2019].

Simplilearn.com. (2018). Scrum Project Management Pros and Cons | CSM Training. [online] Available at: https://www.simplilearn.com/scrum-project-management-article [Accessed 18 Apr. 2019].

wikipedia.org. (2019). Arduino. [online] Available at: https://en.wikipedia.org/wiki/Arduino [Accessed 23 Apr. 2019].

Aebersold, K. (2019). Software Testing Methodologies. [online] Smartbear.com. Available at: https://smartbear.com/learn/automated-testing/software-testing-methodologies/ [Accessed 26 Apr. 2019].

softwaretestingfundamentals.com. (2019). Unit Testing - Software Testing Fundamentals. [online] Available at: http://softwaretestingfundamentals.com/unit-testing/ [Accessed 27 Apr. 2019].

Software Testing Fundamentals. (2019). Acceptance Testing - Software Testing Fundamentals. [online] Available at: http://softwaretestingfundamentals.com/acceptance-testing/ [Accessed 26 Apr. 2019].

Software Testing Fundamentals. (2019). System Testing - Software Testing Fundamentals. [online] Available at: http://softwaretestingfundamentals.com/system-testing/ [Accessed 26 Apr. 2019].

Usability. (2019). Usability Testing. [online] Available at: https://www.usability.gov/how-to-and-tools/methods/usability-testing.html [Accessed 25 Apr. 2019].

Arduino. (2019). Arduino - WIFI. [online] Available at: https://www.arduino.cc/en/Reference/WiFi [Accessed 28 Apr. 2019].

Sparkfun. (2019). Ultrasonic Ranging Module HC - SR04. [online] Available at: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf [Accessed 9 Apr. 2019].