University of Waterloo

Faculty of Engineering

Department of Electrical and Computer Engineering

Controlla

Final Report

Group 2023.19

Prepared by

Karan Dahiya	20768610
Colin Jones	20776388
Brian Newton	20765527
Aditi Lohtia	20754943
Steven Vallejo	20651375

Consultant: Ayman El-Hag

March 22nd, 2023

Abstract

Today’s world is facing several challenges at once, our energy footprint is rising at an alarming rate and in an increasingly connected world, people are bombarded with more distractions than ever. Controlla aims to try and alleviate part of the problem by automating energy-consumption related to home temperature control systems. Controlla detects your presence or lack-there-of and turns off your home’s lights and temperature control when you aren’t using them. Controlla reactivates these services just in time for you to stay comfortable. Controlla also works with you directly in the app through which you can set custom timings to disable/enable temperature control. As Controlla learns more about your lifestyle, it can also recommend settings for you in the app so you can make informed decisions about your power-consumption. We all often forget the simple things, so the main goal of Controlla is to pick up our slack and help us achieve a higher level of sustainability.

Acknowledgements

We would like to acknowledge the support of our project consultant, Professor Ayman El-Hag, for providing us constant feedback and suggestions for improving our project as a whole.

Table of Contents

List of Tables

List of Figures

1 High Level Description

1.1 Motivation

Since the start of the COVID-19 pandemic, self and government imposed restrictions have forced people to stay in their homes, leading to less social interaction, little travel, and more remote working and schooling [1]. As a result, energy consumption in households have greatly increased, with the average utility bill increasing by 9.71% post-COVID [2].

Canadians, and people in general, want to save money on their electricity bills and decrease their energy footprint [3]. The proposed solution, Controlla, allows individuals to achieve both of the previously mentioned goals. By utilizing proximity sensors in a user's home, Controlla detects an individual's presence or lack-there-of, and turns off their home's lights and temperature control. It also works with you directly in a mobile application through which you can control the switch remotely from any location over wifi, or switch it to Automatic mode to turn on and off based on the proximity sensors in your home. These two settings allow users to have more control of their cooling systems no matter where they are.

1.2 Project Objective

The objective of the Controlla project is to design a home temperature control system to help reduce household electricity consumption. Controlla allows users to have the convenience of accessing their home controlling system wherever they are, and not only contributes to reducing household energy bills, but it also helps us achieve a higher level of sustainability and create a healthier planet.

1.3 Block Diagram

The proposed solution is a system that consists of sensors which communicate with the main device to turn the AC on and off. The system architecture is given in Figure 1 with four subsystems: proximity sensors, the main device, backend, and the iOS application.

All components are designed by us except for the following: database service, electrical devices if able to find devices that meet project specifications. (e.g., MOSFETs, Relays, Transformers, MCUs etc.)

1.3.1 Main Device (Controlla)

The Controlla device itself is the core of the project and where the A/C system is disconnected it's power source so it turns off. The core of this block is the MCU, the brain of the Controlla which determines when power is and isn't flowing. The MCU is powered by being connected to the IoT Relay itself through and always on port, this allows the devices to stay more compact and interconnected within its own ecosystem. The MCU also has a receiver and a wifi module built in to communicate with the sensors in the room and then transmit data over wifi to the backend software of the system. Finally, the Controlla has three LED lights which are used as indicators of power being on, of the device being in automatic or manual mode, and for indicating when it is receiving information via the RF receiver.

1.3.2 Proximity Sensor

The proximity sensor devices are used to determine if people or pets are in rooms to determine if the A/C can be turned on or off. Like the Controlla device, the proximity sensors have MCUs that perform the tasks of picking up data from the sensors and transmitting the information to the Controlla using the built-in transmitter. We have also put batteries into the proximity sensor device to provide power for the MCU and the sensors.

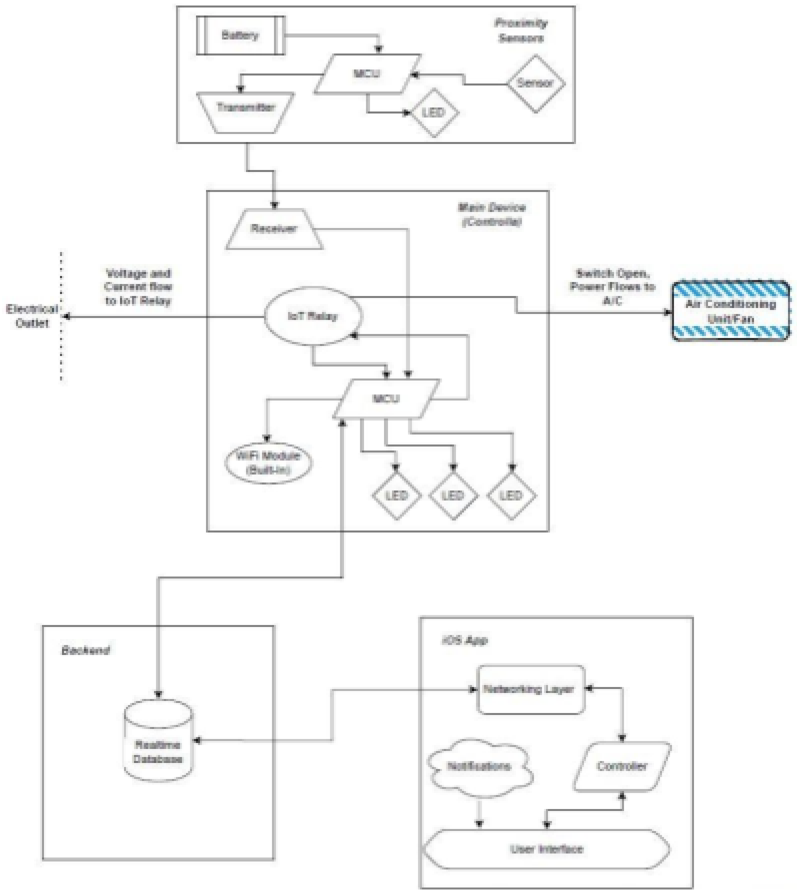


Figure 1: Final Block Diagram

1.3.3 Backend

The backend handles all communication between the user and the Controlla device. The Firebase cloud service allows both the mobile application and backend to send and receive data requests to turn the A/C on or off. The database stores whether the user has requested the A/C be turned on or off, as well as, boolean data indicating whether the system is acting in ‘Manual’ or ‘Automatic’ mode.

1.3.4 iOS App

The iOS Application is the user’s main interface for controlling our device’s functionality. The user receives notifications from the backend requesting permission to turn on or off the A/C when movement or no movement has been detected (Automatic Mode). In addition, the user can also remotely request the A/C be turned on or off via the user-interface (Manual Mode). The controller handles all of the application’s business logic (such as passing incoming data to the user-interface). The network layer interfaces with the Firebase cloud service to send and receive data.

2 Project Specifications

2.1 Functional Specifications

Table 1 shows the functional requirements for Controlla, along with descriptions and their necessity.

Table 1: Functional Specifications

Subsystem	Specification	Description	Necessity
Main Device (Controlla)	Electrical	Controlla device must be able to turn off and turn on the AC/heating unit.	Essential
Proximity Sensors	Electrical	Sensors should be able to be installed on door frames.	Non-essential
iOS App	User Interface	User must receive push notifications for permission to turn off the AC/heating unit.	Essential
iOS App	User Interface	User must be able to manually set the AC unit to be ON or OFF and toggle the AC's operation mode between "MANUAL" and "AUTOMATIC" modes	Essential

2.2 Non-functional Specifications

Table 2 shows the functional requirements for Controlla, along with descriptions and their necessity.

Table 2: Non-Functional Specifications

Subsystem	Specification	Description	Necessity
Main Device (Controlla)	Network	The Controlla device must be able to connect to a wireless network via the user's phone application.	Essential
Main Device (Controlla)	Electrical	Controlla device must be able to safely handle the load used by the AC/heating unit.	Essential
Main Device	Communication	Controlla device must be able to send and receive	Essential

(Controlla)		data wirelessly to backend services.	
Proximity Sensors	Electrical	Sensors must be able to reliably detect the presence of people and pets.	Essential
Proximity Sensors	Communication	Sensors must be able to send data wirelessly to the Controlla device.	Essential
Backend Services	Data Analysis	Backend services must be able to store data of AC/heating being turned off/on by Controlla or the user.	Non-essential
iOS App	Communication	iOS app must be able to wirelessly send and receive data from backend services.	Essential

3 Detailed Design

3.1 Main Device (Controlla)

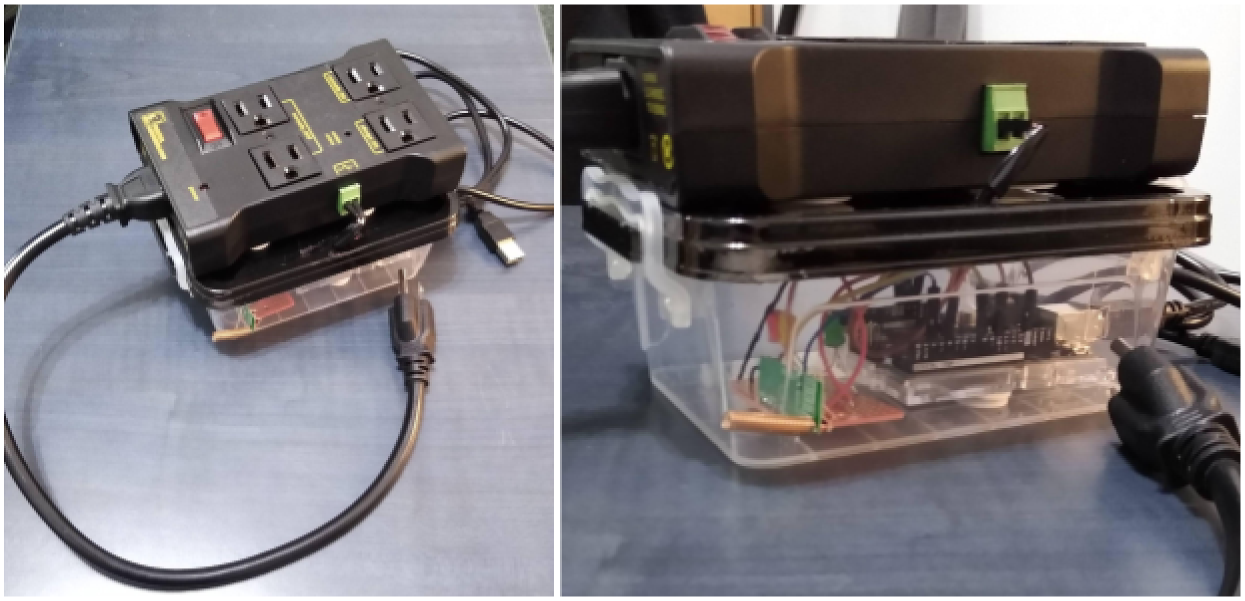


Figure 2: Controlla Circuit

3.1.1 Choice of MCU

When considering MCUs for the main device, we need to make sure that several factors are taken into consideration. Most of the time when selecting an MCU, it is important to keep in mind factors such as the RAM, processing speed and power consumption. For our project, we are mainly focused on the size of the board and making sure it has the right power to run our devices. We had picked some Raspberry Pis to take a look at as the main controller, which include Raspberry Pi 4, Raspberry Pi 3 and the Raspberry Pi Zero 2 W. Unfortunately when we started to work on our prototype, we found that most of these Raspberry Pi's lacked some core functionality that we would need, as such we made the choice to switch to using an Arduino to run our main Controlla device. Here is a comparison chart between two of the three Raspberry Pis we considered compared with the Arduino we went with:

Table 3: MCU Decision Table

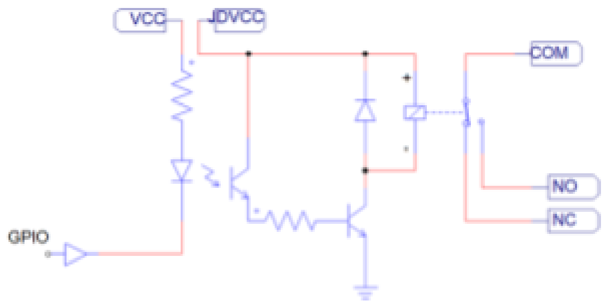
Specification	Raspberry Pi 4	Raspberry Pi 3	Arduino UNO Rev2	Arduino Nano
WiFi Module	Yes	Yes	Yes	No
Bluetooth Module	Yes	Yes	Yes	No

Size (mm)	85x56 (Large)	65x56 (Medium)	68.6x53.4 (Medium)	18x45 (Small)
GPIO Pins	40-Pin GPIO Header	40-Pin GPIO Header	25-Pin GPIO	22-Pin GPIO
Extra Additions	N/A	N/A	N/A	Must Attach Pins to the MCU

For our project, we decided that the Arduino Uno Wifi Rev2 was the best option as mentioned above. This MCU is still fairly small and has all the components required plus the addition of the many libraries that are already native to the Arduino ecosystem. It may have less pins than the Raspberry Pi 3, but it has better ease of use overall at a similar power level and price point. This board is maybe slightly large and consumes too much power, but since it will be plugged into the wall directly as well, it doesn't matter if it consumes a bit more power as it will be saving power in the long term anyways. The sensors, we used other smaller boards, but we will discuss that in a later section. All the data in the table and discussion are from the Raspberry Pi [7] and Arduino[19] websites discussing the various MCUs compared.

3.1.2 Power Switch Circuit

For controlling the flow of power to the window A/C unit (or fan for our demo), there are two main options that we can use, an Electromechanical Relay (EMR) or a Solid State Relay (SSR). Each has their own advantages and disadvantages, but we know that both work well. Our choice is the EMR type as it is a cheaper option to prototype with, and the bonus speed that SSR provides is not needed as our switching doesn't need to be at a particularly high frequency. Once we decide on using this Relay, we determine how to use it with our MCU. To properly control the EMR, a circuit similar to Figure 3 is made.



VCC and JDVCC = Power Terminals
GPIO = MCU Control Signal
COM = Ground for the Relay
NO = Normally Open Connection
NC = Normally Closed Connection

Figure 3: Relay Optocoupler Control Circuit

The circuit for relays can vary, but we are mainly considering the Figure 3 circuit as this is the most common configuration. This particular circuit consists of an MCU sending in power and control signals, a diode, resistors, an optocoupler and a transistor. The general operation of this system involves the power for the circuit coming from the 5 V rail of a Microcontroller and the control signal being sent into the system via GPIO. When the control signal is low, the LED in the optocoupler triggers the phototransistor to allow current to flow, which in turn causes current flow through the whole system, activating the relay to switch to the NO terminal. When the control signal is high, the optocoupler opens and causes the relay to deactivate, moving the switch back to the NC terminal. The diode near the relay is to make sure that the demagnetization of the coil doesn't affect the rest of the circuit, and mainly the MCU [8]. All the components except the MCU are usually included on Relays that are purchasable online at the standard 120 VAC with a number of different current options available. While doing research on these relays, it is very apparent to us that dealing with these higher voltage values is not only more dangerous, but also if we create a device that hasn't been approved for use on home outlets and it causes any issues in the electrical distribution, that we may have some possible legal issues. It is possible to purchase an isolation transformer to then test isolated from the direct wall plug, but we still have the high voltage and current considerations to keep in mind. As such, we are using a premade approved device that ensures safety in our project, the IoT Relay created by Digital Loggers Inc shown in Figure 4.

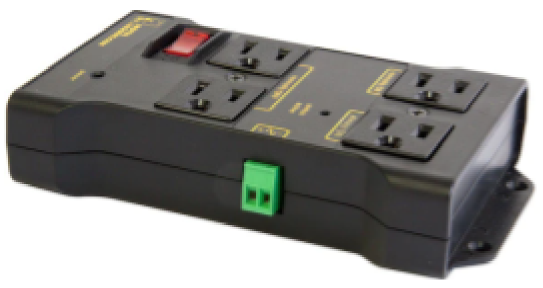


Figure 4: IoT Relay by Digital Loggers Inc.

This device has all the features we require for our project in a package that we can purchase online. It makes the design safer and ensures that we don't have to go through any legal testing to use our device on a normal 120 VAC outlet. The IoT Relay works for 3-48 VDC or 12-120 VAC, with the current varying depending on the power wire gauge. The maximum current that the device can handle is 12 A with the correct gauge [9]. We simply have to attach the Arduino via the green terminals seen on the side of the device and we can thus control the NC and NO terminals of the device.

3.1.3 Arduino Power and LED Circuit

To power the Arduino Uno Rev2 Wifi, we simply connect the MCU to the "Always On" outlet included on the IoT Relay. This means we can simply use a normal USB cable alongside a standard wall outlet to USB converter to power our Arduino. This keeps the system simple and again doesn't require us to manipulate wall outlet A/C power directly and therefore avoids the possible issues that may occur from making our own power converter.

For the LEDs, we are using three simple LED circuits connected to the MCU. These three LEDs represent the Controlla being ON, in Automatic/Manual Mode, or receiving data depending on which colour is ON green, yellow, or red respectively. These circuits are very straightforward and generally have the form of the circuit in Figure 5.

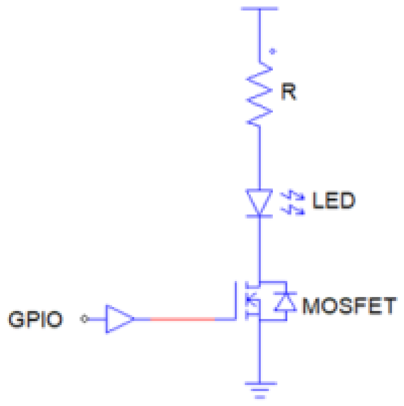


Figure 5: Basic LED Circuit

3.1.4 Receiver Circuit

The receiver circuit on the Controlla is made with the receiver half of a 433MHz Mini RF transmitter and receiver pair. The receiver does not need to receive complex data, simply a binary signal from the proximity sensor device to indicate whether the room is occupied or not. The receiver we use is the Geekcreit 433 MHz RF receiver. It connects to the Controlla Arduino as follows:

- ATAD to GPIO I/O - Pin 10
- VCC to 5V Power Pin
- GND to GND Pin

RF messages are received by the Arduino using this receiver and the RCSwitch library. These two in combination allow the Arduino to control the relay based on the data received. The finite state machine takes care of the actual data processing and determines which state Controlla is in.

3.1.5 Finite State Machine

Our main device, which actually controls power to the AC unit, determines when to turn the AC on or OFF. Thus, the Controlla device has a 3-state finite state machine. The three states are ON, OFF, and SETUP. Actual temperature hysteresis is still performed by the AC unit itself (the way it is without the Controlla attached). The Controlla allows the AC to be turned on and off remotely or based on the presence of people and pets.

The FSM framework is implemented in a generic manner first, then provides the specific state and transition information of our problem. In any state, the generic FSM is repeatedly performed an in-state action (performed while in the state) and a transition test (to test whether the state should be changed). When transitioning, the generic FSM performs a transition action (a specified arbitrary action).

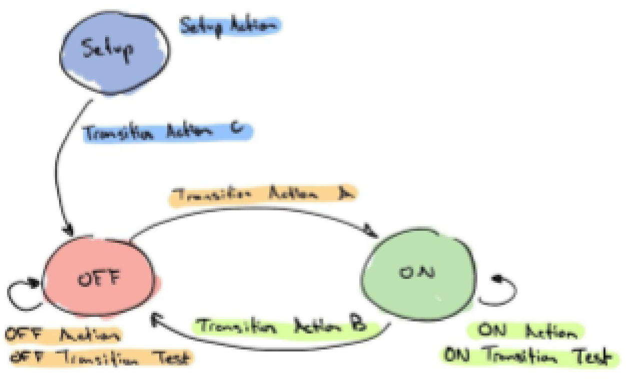


Figure 6: Finite State Machine

A **state** is a data type that we create which contains *state ID*, *in-state action*, *transition test*, and *transition action*. When calling our generic FSM framework to **run**, it is provided with a list of states containing all necessary information. Figure 6 shows our Finite State Machine.

SETUP State

- <u>State ID:</u> ‘SETUP’
- <u>in-state action:</u> enter “search” or “connect” mode for BT connection, connect to WiFi via BT device, keep yellow LED indicator on

- <u>transition-test:</u> check if connected to WiFi
- <u>transition-action:</u> send a message to the backend that the Controlla is finished set-up

OFF State

- <u>State ID:</u> ‘OFF’
- <u>in-state action:</u> read sensor input, read input from backend, keep power to AC off, keep the red LED indicator on
- <u>transition-test:</u> check data from sensors and backend to see if we should turn the AC on
- <u>transition-action:</u> send a message to the backend that the AC is being turned on, turn off the red LED indicator

ON State

- <u>State ID:</u> ‘ON’
- <u>in-state action:</u> read sensor input, read input from backend, keep power to AC on, keep the greed LED indicator on
- <u>transition-test:</u> check data from sensors and backend to see if we should turn the AC off
- <u>transition-action:</u> send a message to the backend that the AC is being turned off, turn off the green LED indicator

3.1.6 WiFi Connection

The microcontroller that we use for the main device is an Arduino Rev2. The microcontroller comes with a built-in WiFi module. There exists an Arduino library called WifiNINA which allows easy setup of Wifi connection on MEGA boards (our microcontroller falls into this category). By supplying the following parameters to the library, Wifi connection can be established. [2]

```
#define SECRET_SSID "YOUR_NETWORK_SSID"
#define SECRET_PASS "YOUR_NETWORK_PASSWORD"
```

Now, Controlla is ready to communicate with our backend server.

3.1.7 Networking/Backend-interface

For the user to remotely turn their AC on and off or set timings for their AC, Controlla needs to communicate with our **Firebase** backend server. In order to do this, we install the **Firebase_Arduino_WiFiNINA** module on our Arduino. This module allows the Controlla to be easily programmed to communicate with the servers.

The following key-value pairs are obtained from our Firebase server and inserted into the code. These pairs are provided to the **Firebase** module to initialize the connection to the Firebase database. [1]

```
#define FIREBASE_HOST "your-firebase-project-id.firebaseio.com"
#define FIREBASE_AUTH "Your_Firebase_Database_Secret"
```

The status of database values indicating whether the AC is on/off or timings are set is available to access after Pyrebase is initialized.

3.2 Proximity Sensor

The proximity sensor is an integral part of the overall system. It is used to determine the presence of anyone in the user's home and relay that data to the main controlla which then uses this information to efficiently regulate the user's connected devices.

The proximity sensor is mounted at a key access point in the user's home, for example the doorway of their main entrance.

3.2.1 Choice of MCU

An appropriate MCU for the proximity sensor component of the device needs to meet specific needs. It needs to power whichever sensor ends up being selected for proximity and motion detection, as well as the transmitter to the main board. It also needs to have sufficient pins to interface with said sensor and transmitter, and must be programmable to run a custom state machine to fully optimize the interaction between the sensor and transmitter. We decided to go with the Arduino Nano. We had looked at Raspberry Pi Zero's before, but the interfacing with the Arduino Nano was much smoother and it met our project requirements. It is small, has sufficient pins and power and can process the required data easily. Considering all this, we used this board for both sensors that we developed.

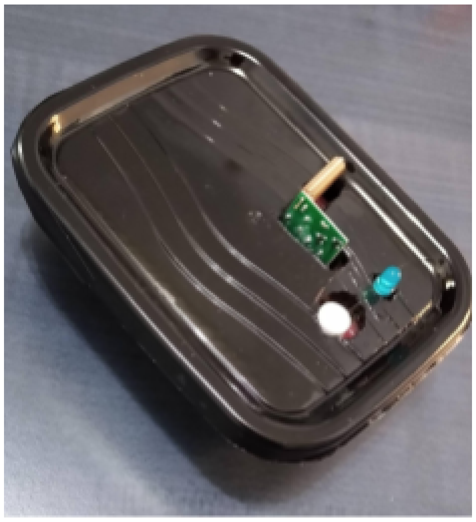


Figure 7: Proximity Sensor Circuit

3.2.2 Battery

For the proximity sensor circuit, we need a power supply, in our case, a battery that provides power to the Arduino and in turn the sensors and transmitter circuit. To do this, we need to see how much power the Arduino draws as it provides power for itself and the other components. To do this, we use it's rated voltage and current values: 5V * 0.9A = 4.5W. This is the maximum amount that the Arduino would use, but since the sensor circuit is fairly small it should not use this much at all times. The other components have low currents and are provided for by the Vcc and GPIO pins of the Arduino. To power the device, we use a 9V battery as we want to keep the circuit small. We are using the AmazonBasics 9V battery as it will provide enough power and keep the device compact enough.

In order to connect the batteries to the sensors, we purchased 9V battery cases that came with power and ground wires to connect to the Arduino directly via its ground and Vin pins. This will provide enough power to feed all the components present on the board. Using the Arduino and these cases makes it very easy and convenient to deliver power to the Arduino to power the other devices present.

3.2.3 Sensor Circuit

The proximity sensor employs passive infrared (PIR) sensors to determine whether somebody is present in a room. A PIR sensor has two slots in it, each made of IR sensitive material. When a body emitting infrared radiation passes by the sensor, it first passes one of the two slots in the PIR sensor then the other. This produces first a positive voltage differential between the halves, then a negative differential as the body is leaving. Figure 8 shows the output behaviour.

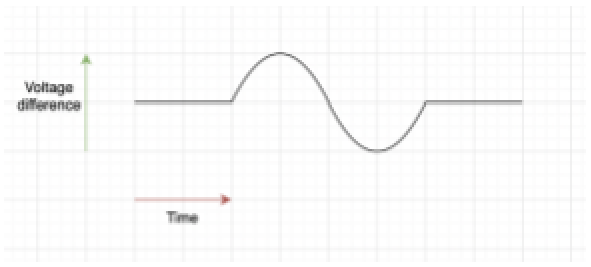


Figure 8: PIR Voltage Activation

This pulse is measured by the MCU to determine that something has moved past the sensor. PIR sensors typically have one data output pin which needs to be interfaced to an available proximity sensor MCU GPIO pin. The code to obtain information from a PIR sensor simply must poll this output pin periodically, interpreting high values as detected motion and low values as otherwise. For this circuit, Vcc is connected to the 3.3V pin, ground to the ground pin, and the data is being sent into pin 11.

The chosen PIR sensor is the Panasonic Electric Works EKMC1607113 (Digi-Key part no. 255-6547-ND), for its availability, low-cost and complete functionality.
Since the PIR sensor can only detect movement and not stationary targets, two PIRs are employed along with a finite state machine to determine complete occupancy information. One PIR sensor points towards the doorway and the other points inwards of the room.

3.2.4 Finite State Machine

A finite state machine is used alongside the PIR motion sensors to determine whether there's occupancy in the room. The state machine is implemented onboard the Arduino Nano using C++. If someone is entering the room, they first trigger PIR 1 and then PIR 2, the opposite is true if they are leaving the room. If someone walks past the doorway without entering the room, they only trigger PIR 1 and if they are moving around inside the room, they only trigger PIR 2. The diagram in Figure 9 shows an overview of how this logic is implemented in a finite state machine.

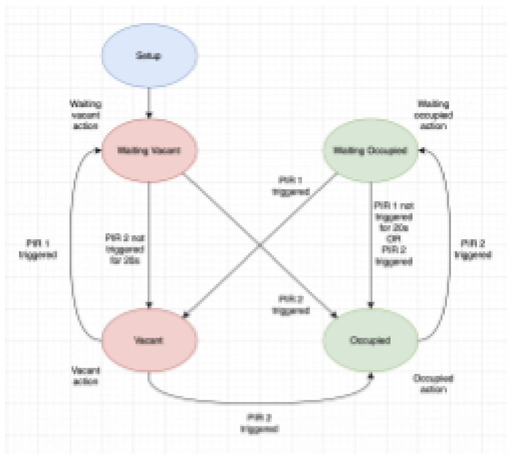


Figure 9: Proximity Sensor Finite State Machine

SETUP State
- <u>State ID:</u> 'SETUP'
- <u>in-state action:</u> Initialise GPIO and PIR sensors
- <u>transition-test:</u> check if PIR sensors are online
- <u>transition-action:</u> send a message to the main controlla that the prox sensor is online

Waiting Vacant State
- <u>State ID:</u> 'WAITING_V'

- in-state action: Time 20 seconds
- transition-test: Check PIR sensors output

Vacant State
- State ID: ‘VACANT’
- in-state action: send “vacant” message to main controlla device
- transition-test: check PIR sensors output

Waiting Occupied State
- State ID: ‘WAITING_O’
- in-state action: Time 20 seconds
- transition-test: check PIR sensors output

Occupied State
- State ID: ‘OCCUPIED’
- in-state action: Send “Occupied” message to main controlla device
- transition-test: check PIR sensors output

3.2.5 Transmitter Circuit

The transmitter circuit on the proximity sensor is made with the transmitter half of a 433MHz Mini RF transmitter and receiver pair. The transmitter does not need to transmit complex data, simply a binary signal to the main Controlla device to indicate whether the proximity sensor has detected motion passing through its door frame or not.

The transmitter we use is the Geekcreit 433 MHz RF transmitter. It is connected to the proximity sensor Arduino as follows:

- ATAD to GPIO Pin 4.
- VCC to 5V power
- GND to Ground pin.

Once again this RF device uses the RCSwitch Library to send out data to the receiver in the area. This allows the transmitter circuit to be used very easily with the currently implemented RF receiver.

3.3 Backend

The backend component is used as a communication channel between the main device and the mobile application. The main criteria to evaluate the choice of service for our backend component are: Ease of use, and Cost efficiency.

3.3.1 Choice of Service (Firebase)

Firebase is a Google based platform that provides several services as well as purposeful APIs that contribute to rapid and scalable building of mobile/web applications.

We decided to opt for firebase because of its ability to manage data in real-time and improved user experience. In addition to that, firebase has an easy integration with iOS which for the purposes of this project fits our constraints as we plan to develop an iOS mobile application. The reason why we linked our project with Firebase is because of its availability of several features that with other services, we may not obtain the same result. For example: the Authentication Libraries, Realtime Database [10].

Typically it takes months to appropriately set a functional **authentication process**; however, Firebase allows us to set up the entire system with minimal difficulty. This built-in support comes with authentication services such as Google, Facebook, and Twitter; therefore, it makes the service more robust and developer friendly.

For our project, a **real-time database** is crucial since we are posting time sensitive data into our database, and by using Firebase we enable user data to be stored and synchronized in real-time, as well as enable application data, even when the user is offline [10].

In terms of cost efficiency, compared to any other service, we found that Firebase offers the majority of services that we are going to use at no cost, and it aids for the general purpose of a cost efficient project.

3.3.2 Database (NoSQL, Database Set up)

Firebase is a NoSQL Database, meaning that it does not have relational data and its structure is not tabular. It is based on a **key-value relationship**, and the format in which firebase stores information does not have to be consistent with any schema; we store data in JSON files, which have nested structures and are represented by a tree. For our purposes we come up with several formats to save information in the database [11].

```
"ManualSwitchOn" : {
    "displayName" : "Test Name",
    "ManualSwitch" : true,
    "TimeStampOn" : yyyy/mm/dd HH:MM:SS,
}

"ManualSwitchOff" : {
    "displayName" : "Test Name",
    "ManualSwitch" : true,
    "TimeStampOff" : yyyy/mm/dd HH:MM:SS,
}

"SensorSwitchOn" : {
    "displayName" : "Test Name",
    "SensorSwitch" : true,
    "TimeStampOn" : yyyy/mm/dd HH:MM:SS,
}
```

```
"SensorSwitchOff" : {
    "displayName" : "Test Name",
    "SensorSwitch" : true,
    "TimeStampOff" : yyyy/mm/dd HH:MM:SS,
}
```

In these examples, the key is denoted by *SensorSwitchOn, SensorSwitchOff, ManualSwitchOn, ManualSwitchOff,* and the value for each corresponding key in our case would be a JSON formatted file which contains all the information we need for the corresponding validation [11].

Since we are making **smart recommendations** for energy usage based on the times in which the system is on and off, we can potentially store enormous amounts of data within our database, and the data retrieval is still very efficient due to the simplicity of a key-value relationship.

Creating and setting up a Firebase Database is very simple. All we need to do is access the Firebase Console [10], select the project we want to add a database to, then we are prompted to select the Firebase Security Rules. For our purpose we put it under **test mode.** Finally, we select the location of the database which follows this format
DATABASE_NAME.firebaseio.com, and we successfully set up a Firebase Database for our project, so we now start writing and reading to and from the database [13].

To check if we write into our database, we need an instance of it, which we get by using the method getInstance() and reference the location we want to write. After that, we only need to set the value we want stored in the database, like such [14].

```
// Write a message to the database
FirebaseDatabase database = FirebaseDatabase.getInstance();
DatabaseReference myRef = database.getReference("message");

myRef.setValue("Hello, World!");
```

As for reading the values in real time from the database, we need a *ValueEventListener* to the reference that we created during the writing procedure. Once the listener is attached, the method *onDataChange()* is triggered again every time data changes. We set such a codebase for this simple example [14].

```
// Read from the database
myRef.addValueEventListener(new ValueEventListener() {
   @Override
   public void onDataChange(DataSnapshot dataSnapshot) {
       // This method is called once with the initial value and again
       // whenever data at this location is updated.
       String value = dataSnapshot.getValue(String.class);
       Log.d(TAG, "Value is: " + value);
   }
```

```
    @Override
    public void onCancelled(DatabaseError error) {
        // Failed to read value
        Log.w(TAG, "Failed to read value.", error.toException());
    }
});
```

With this simplistic yet functional example, we now review the generic procedures that we are going to use during the development of our project, i.e. writing/reading data to/from the database.

3.3.3 Notifications

As stated in the design of the project, we aim to develop such a system that sends out smart notifications directly to our users based on the appliance usage, in our case Heater/AC. Our mobile application uses a service called **Firebase Cloud Messaging** which, as the name suggests, is a messaging solution that lets us send messages to our users with no cost [16].

There exists an implementation path that we need to follow in order to properly set up this service alongside our mobile application.

First, we need to set up the Firebase Cloud Messaging SDK within our mobile application. By installing this SDK, we create an **authentication key** which helps register our application for remote notifications within the FCM service [16].

Second, we must add **message handling logic** to our client application to, among others, accept the message request, and do any additional processing such as the creation of **metadata** from the data such as the message ID, and message received time [16] .

Lastly, we need to develop our app server, for this step we need an extra SDK installation of Firebase Admin SDK. The **message requests** are built in a trusted server environment. such as the one the Firebase Admin SDK provides. This Cloud Service has support for different modern technologies that we used, such as C and C++ [17].

3.4 iOS App

3.4.1 Choice of Technologies

For the mobile app of our project, we decided to use iOS since one of our members has significant experience in iOS app development. However, there are multiple development frameworks for iOS applications including UIKit (imperative programming) and SwiftUI (declarative programming). Table 4 shows a comparison of the two technologies. [4]

Table 4: UIKit vs SwiftUI

UIKit	**SwiftUI**
supports all devices	supports only iOS 13+ and Xcode 11
very well documented, most common complex problems have solutions in forums already	still young, so there isn't as much data on support sites
can be mixed with SwiftUI	can be mixed with UIKit
more boilerplate code is required	easy to learn and code is simple
cannot interact with new Apple features such as Shortcuts and Siri	easy to produce quality layouts

We are primarily using **UIKit** for our application because SwiftUI is a newer framework that Apple released in 2019. This means that SwiftUI still lacks the ease of performing some more complex engineering tasks. Since our UI requirements are fairly minimal, and that is where SwiftUI is more useful, we do not really need SwiftUI. However, we intend to have Bluetooth functionality as part of our application to connect the Controlla to WiFi. This Bluetooth functionality is easier to implement in UIKit.

Additionally, we may make use of Swift packages (modules) to make our development process easier. There are multiple package managers such as CocoaPods, SPM, and Carthage. We are using **SPM (Swift Package Manager)** for the simple reason that it is built into the Xcode IDE and is the least complicated method. The other methods may improve our app size by decreasing it slightly, however, we do not intend to use many modules. Since reliance on modules isn't a large concern for this project, we can go with the simplest approach.

3.4.2 UI/UX design

The User Interface (UI) for the iOS Subsystem is built using the fundamental UI library provided by Apple UIKit. UIKit offers many resources for views, controls, layouts, animations, and more.

For example, in order to create a view, the program below is implemented.

```
let rect = CGRect(x: 10, y: 10, width: 100, height: 100)
let myView = UIView(frame: rect)
```

We also implement the UI Activity Indicator View to allow users to see if the AC/heating system is active or inactive.

```
@MainActor class UIActivityIndicatorView : UIView
```

3.4.3 App Architecture

The app architecture we are using is **MVC (Model-View-Controller)**. We chose MVC over MVVM (Model-View-ViewModel) because although MVVM is more scalable, MVC is faster and easier to implement, and the number of lines of code in our app is not intended to grow a large amount over time after the initial implementation. MVC is a design pattern where code has functionality split between three layers: View, ViewController, and Model.
MVC is the pattern recommended by Apple and is what UIKit is built around. [5]

The View is made up of the UI elements that the user of the app sees and interacts with. The View does not contain any logic or interactability in itself, it is just the UI elements and their visual style.

The Model is made up of the objects holding data which must be presented to the user, altered by the user, and/or fetched from the backend servers. The Model does not contain any logic to display, fetch, or update the data in itself, it is just the object holding the data.

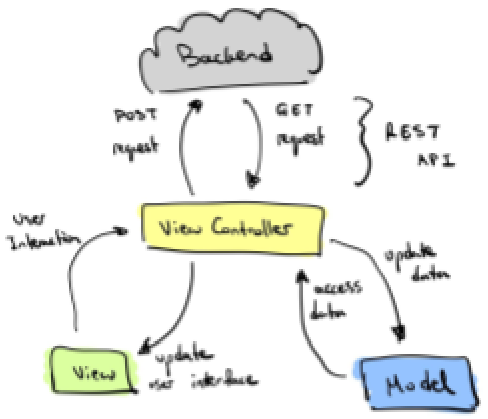


Figure 10: ViewController

The ViewController is where the business logic of the application resides. The ViewController sends network requests to fetch data from the backend, updating the Model, and presents the Model's data to the user, updating the View. When the user makes changes to the data in the VIew, the ViewController updates the Model with these changes. The ViewController may also send network requests to update the backend data based on user changes. Any kinds of actions that should occur as a result of user interaction are handled by logic in the ViewController in Figure 10.

The main benefit of MVC is the separation of concerns between the Model (handling persistent data), View (handling user interface and styles), and ViewModel (handling business logic). This makes it easier to test the View, Model, and ViewController independently. However, one thing to be careful of is that the logic in the ViewController may become more complex as time goes on, making the ViewController functions difficult to test in isolation. This is a risk of MVC that we are accepting to reap the benefits of rapid prototyping and development.

3.4.4 Backend Interface

In order to control the physical Controlla device remotely, the iOS app needs to communicate with our backend Firebase server and database. Firebase provides an SDK (software-development-kit) which is imported into our iOS app as a package (or module). Since we are using SPM (Swift Package Manager) to manage our modules, we can use it to install the Firebase SDK.

In order to use the Firebase SDK, we must adhere to the following constraints.

- Xcode 13.3.1 or later
- project must target iOS 10 or later

Once our Firebase project is created, and the SDK has been installed we connect to our Firebase server and database by adding Firebase to our app's *Info.plist* file which functions as a configuration file for iOS apps. We then initialize the Firebase module in our app's initialization delegate. Then we use the module's provided methods for sending and receiving data from our database. [6]

4 Prototype Data

4.1 Delays

4.1.1 Mobile App Communication

The following are the results of tests which measure **time from user input in the mobile app** to change the state of the AC (turning it ON or OFF) or operating mode (MANUAL or AUTOMATIC) **to the change being visible at the main device** (i.e. the AC actually turning ON/OFF, etc.)

Table 5: Time from User Input to Change Being Visible

Test	Turn AC "ON"	Turn AC "OFF"	Toggle "AUTOMATIC" Mode
Average time from 10 total attempts (seconds)	4.55	2.37	6.21

It should be noted that these delay metrics, though valuable for testing the responsiveness of our app, server, and microcontroller, are not necessarily indicative of the quality of our final product. In a real-word smart AC, it would not be desirable for power to the device to be turned ON/OFF extremely quickly or without any delay, as this could result in damage to electrical components.

4.1.2 RF Communication and State Machine Delays

In our testing of the RF TX and RX devices, we noticed that the messages are transferred almost instantly, which is ideal for our purposes as it makes sure that there are no long time delays for the transfer of information. The delay for the RX and TX devices however comes from the state machine. We have multiple delays built in and are using methods for ensuring that the relay doesn't swap between ON/OFF too quickly. These delays in the code lead to sometimes the device being more unresponsive than hoped. This however wouldn't be an issue in a proper retail version as we would have longer delays built in to account for temporary lack of movement. This would make sure that that device doesn't turn off at the wrong times if the user is too still for too long.

4.2 Detection

When performing testing on the PIR sensors, we would slowly walk into range of the sensor and then use a protractor and measuring stick to determine the approximate detection angle and the range it could detect up to. We found that the distance it detected was fairly universal at all angles and could therefore detect people up to 2.5m away consistently.

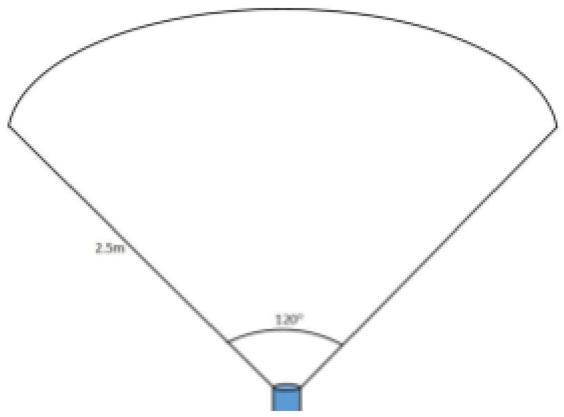


Figure 11: Detection Distance and Angle

Our results gave us the diagram shown in Figure 11. This gives a rough idea of how to set up the sensors in a room to ideally cover the most surface area.

As for speed of detection, for any person within its detection range, the speed of detection is quite quick. We do not have a precise number as the detection happens quicker than we can measure, however it consistently detects movement within its cone in less than a second which is ideal for our uses.

5 Discussions and Conclusions

5.1 Evaluation of Final Design

Our final design meets all of the essential project objectives and specifications. Our objective was to design a main controller that could control home appliances such as air conditioners and heaters alongside sensors to determine when the device owner is coming and going. The main controller alongside the sensors would constitute a system that optimizes the owners power-hungry A/C appliances, lowering both their energy footprint and power bills.

Our final design successfully implemented both the main controller and the proximity sensors. The controller and proximity sensors are both designed around Arduino microcontrollers. The sensors use IR motion detection to determine when the owner is nearby, and send this information to the main controller using 433 MHz RF communication. The main controller is connected to the enable/disable input of a power relay which is connected between the A/C appliance and a power outlet. Upon receiving information from the sensors, the controller then sends a signal to the relay, either to deactivate the appliance or turn it on, depending on whether or not it's needed.

5.2 Use of Advanced Knowledge

The technical abilities required for the development of Controlla are drawn from a variety of upper-year engineering courses. Knowledge of cloud services and network communication to communicate with our cloud server are drawn from ECE 358 (Computer Networks). The design of our Realtime Database is drawn from concepts in ECE 356 (Database Systems). The ability to run our network communication code on a separate thread from the UI in our mobile app comes from ECE 350 (Real-time Operating Systems). State machine design and real-time code considerations come from ECE 455 (Embedded Software).

In implementing the hardware of Controlla, advanced knowledge in circuitry, power electronics and RF communication was required. To this end, knowledge learned in ECE 318 (Communication Systems), ECE 340 (Electronic Circuits 2) and ECE 360 (Power Systems) was used.

5.3 Creativity, Novelty, Elegance

Unlike most smart ACs and smart plugs in the market, Controlla is a middle ground between traditional, traditional appliances, and modern smart ones. Its novelty lies in the fact that instead of being an expensive, entire smart AC, it is an adapter that can cheaply convert old single-room AC units into something approximating a smart AC in functionality.

Existing smart plugs do not have any kind of sensor component to provide or stop providing power to appliances based on the presence of people or pets. Existing smart home solutions (such as Ecobee) offer expensive, all-or-nothing solutions to the consumption of electricity in appliances based on the presence of homeowners. Unlike those systems, Controlla will work with the existing infrastructure of its users' homes and existing functionality of their traditional AC unit.

5.4 Quality of Risk Assessment

In the final implementation of our project, we attempted to mitigate any potential hardware risks by following safe circuit conventions and the advice following the circuit safety assessment with laboratory staff. This involved fuse-protecting all power inputs to the sensors and the main controller, ensuring that there are no exposed wires or sharp edges, and ensuring optimal thermal flow for all microcontrollers.

5.5 Student Workload

Each member of the group contributed an equal amount of work over the course of the terms 4A and 4B. Approximately 120 hours were dedicated towards the development of the project by each member of the group during 4A. The initial distribution of the tasks was equal among every member, and so for the electrical subsystems in the hardware section, 3 people were allocated to its development, and the other 2 members were focused on the software development section of the project. This workload enabled circuitry to be ready on time for software testing, and symposium presentation.

References

[1] Idris Zainal Abidin. (2020, November 24). Send Data to Firebase Using Arduino Uno WiFi Rev2. Cytron Technologies; Cytron Technologies.
https://www.cytron.io/tutorial/send-data-to-firebase-using-arduino-uno-wifi-rev2
[Accessed 10 Jan 2023]

[2] Getting Started with the Arduino Uno WiFi | Arduino Documentation | Arduino Documentation. (2023). Arduino.cc. https://docs.arduino.cc/retired/getting-started-guides/ArduinoUnoWiFi
[Accessed 10 Jan 2023]

[3] Delport, R. Setting up the Raspberry Pi to transmit RF signals. (2016). Retrieved June 21, 2022, from behind-the-scenes.net website:
https://behind-the-scenes.net/433mhz-rf-communication-from-a-raspberry-pi/#Setting_up_the_Raspberry_Pi_to_transmit_RF_signals
[Accessed 21 June 2022]

[4] Zaitseva, A. (2020, January 23). SwiftUI vs UIKit: Benefits and Drawbacks. Retrieved June 19, 2022, from Steelkiwi.com website: https://steelkiwi.com/blog/swiftui-vs-uikit/
[Accessed 18 June 2022]

[5] Laso-Marsetti, F. (2019, April 15). Model-View-Controller (MVC) in iOS – A Modern Approach. Retrieved June 19, 2022, from raywenderlich.com website:
https://www.raywenderlich.com/1000705-model-view-controller-mvc-in-ios-a-modern-approach
[Accessed 18 June 2022]

[6] Add Firebase to your Apple project | Firebase Documentation. (2022). Retrieved June 19, 2022, from Firebase website: https://firebase.google.com/docs/ios/setup
[Accessed 18 June 2022]

[7] Raspberry Pi Ltd. (2022). Buy a Raspberry Pi – Raspberry Pi. Retrieved June 22, 2022, from Raspberry Pi website: https://www.raspberrypi.com/products/
[Accessed June 18, 2022]

[8] Dejan. (2015, September 9). How To Mechatronics. Retrieved June 22, 2022, from Howtomechatronics.com website:
https://howtomechatronics.com/tutorials/arduino/control-high-voltage-devices-arduino-relay-tutorial/
[Accessed 19 June 2022]

[9] Digital Loggers Inc. (2019). IoT Relay II: Safely Control AC power from logic. Retrieved June 22, 2022, from digital-loggers.com website: http://www.digital-loggers.com/iot2spec.pdf [Accessed 20 June 2022]

[10] Firebase Features. (2019, April 1). Retrieved June 20, 2022, from NBN Minds website: https://www.nbnminds.com/when-you-should-and-shouldnt-use-firebase/ [Accessed 19 June 2022]

[11] NoSQL Databases. (2016, May 10). Retrieved June 23, 2022, from Couchbase.com website: https://www.couchbase.com/resources/why-nosql [Accessed 20 June 2022]

[12] Firebase console. (2022). Retrieved June 23, 2022, from Google.com website: https://console.firebase.google.com/ [Accessed 21 June 2022]

[13] Installation & Setup on Android | Firebase Documentation. (2022). Retrieved June 23, 2022, from Firebase website: https://firebase.google.com/docs/database/android/start [Accessed 21 June 2022]

[14] Read and Write Data on Android | Firebase Documentation. (2022). Retrieved June 23, 2022, from Firebase website: https://firebase.google.com/docs/database/android/read-and-write [Accessed 21 June 2022]

[15] Firebase Database REST API. (2022). Retrieved June 23, 2022, from Firebase website: https://firebase.google.com/docs/reference/rest/database [Accessed 21 June 2022]

[16] Firebase Cloud Messaging | Firebase Documentation. (2022). Retrieved June 23, 2022, from Firebase website: https://firebase.google.com/docs/cloud-messaging [Accessed 22 June 2022]

[17] Set up a Firebase Cloud Messaging client app on Apple platforms | Firebase Documentation. (2022). Retrieved June 23, 2022, from Firebase website: https://firebase.google.com/docs/cloud-messaging/ios/client
[Accessed 22 June 2022]

[18] Delport, R. 433MHz RF Communication to a Raspberry Pi. (2016). Retrieved June 21, 2022, from behind the scenes website: https://behind-the-scenes.net/433mhz-rf-communication-to-a-raspberry-pi/
[Accessed 21 June 2022]

[19] *Arduino*. Arduino Online Shop. (n.d.). Retrieved July 2022, from https://store-usa.arduino.cc/collections/arduino