Virtual
Intelligent
Remote
Ubiquitous
System

Contents

Authors note

At the time that this project has been undertaken, it would be remis to not comment on the global situation, specifically the effects of the COVID-19 pandemic. One of the impacts of the pandemic was the cessation of face-to-face teaching. This led to the choice to use Tinkercad software to design and simulate the home automation system. Subsequently, further development included the creation of an actual physical build.

Abstract

This project and report explore the topic of Home Automation and the wider scope of Smart Home's, through the specific viewpoint of a design spec that required four separate devices by designed and implement. Each device represents a basic yet normal activity that would typically have to be controlled by a user directly. The report shows how these goals were met using a virtual simulation of Arduino development environment and details the process, and successes.

Introduction

This report forms an essential part of the assessment for the Distributed and Mobile Computing module (7CC003). It serves as a written record, detailing all major milestones throughout the process.

The initial project brief described a portable wireless remote-control system for the home. Whereby the user would be able to control several specific functionalities from a 'master' device and then a single or multiple 'slave' device/s would perform the desired outcome. The project looks at home automation as well as wireless communication technologies working together.

Aims & Objectives

Aims

- To create a hardware device and software solution that allows for an internal light or similar analogous to be turned on and off as well as allow the user to control the luminosity

- To create hardware device and software solution that allows for an external light or similar analogous to be turned on and off as well as allow the user to specify an automated function that allows the light to be controlled by ambient light levels.

- To create a hardware device and software solution that allows for a gate or similar analogous to be opened and closed remotely based on the users input.

- To create a hardware device and software solution that provides passive fire detection or similar analogous and warning to the user.

- To create a hardware device and software solution for a remote-control device that allows the user to both monitor the state of all other devices in the system as well as control them within the scope of their designed operation.

- To integrate a mobile web/App interface with the system designed and created.

Objectives

- Research suitable hardware components to meet the specific requirements outlined within the project brief for each device
- Research appropriate software solutions that allow for control and communication of the various devices, wirelessly.
- Produce a working model for each device, either virtual or actual.
- Test and report on each device developed.

Initial Research

Home Automation

Since the early days of science fiction, from the page to television we have been slowly exposed to the idea of an automated home, be this from the repeated writings of novelists such as Arthur C Clarke and Phillip K Dick or in the 1966 debut of Star Trek where people saw the automatic doors open and close on the iconic star ship Enterprise. As time has marched forward, we have seen higher levels of home automation that typically started as fiction before becoming reality.

In the book Smart Home Automation (2015), Goodwin describes a Smart Home as being a home where the direct application or appearance of the application of intelligence is used to make living more beneficial or pleasant. There are many 'smart' devices now readily commercially available that allow users to have a greater control over their home, with some offering a level of automation. A clear example of this would be the Google Nest Learning Thermostat (2021), this device allows a user to control and regulate their home central heating system remotely. As well has having remote control functionality it also boasts the ability to learn your daily routine, automating and programming itself.

Some of the most common types of home automation devices that are found in homes are Smart bulbs such as the Phillips hue smart bulbs (2021), smart speakers like the Sonos One (2021) home wi-fi speaker. Emerging more is the usage of smart plugs, by being able to add a smart plug to any electrical device you are essentially turning the device into a smart device.

Wireless technologies

There are a number of wireless communication technologies being used day to day across society. Arguably the most common being the digital mobile telephone. Digital communication signals became the norm and replaced older analogue signals, for a number of reasons, data security being one of the primary motivators. Other examples of wireless technology involve things like RFID which powers contactless payments.

Bluetooth

From a system developmental viewpoint there are a number of Bluetooth based compatible components that can be easily added to an Arduino unit. One of the more readily available modules is the BM70/71. A module of this type contains both a transceiver and receiver and is capable of two-way communication up to a range of approximately 100 meters, though this would be under optimum conditions and may be unreliable nearer to the maximum range. The module operates on a 3 channel Pulse Width Modulation over an ISM band of 2.402 to 2.480ghz.

Wi-fi
Wi-fi communication is a relatively large umbrella term that covers a broad spectrum of protocols based upon the IEEE 802.11 family of network protocols. The most commonly used frequencies in home based wi-fi is the 2.4 gigahertz and 5 gigahertz frequency bands. Those two main bands are then subdivided into multiple channels. It is possible to have crossover when it comes to channel allocation, this can then lead to packets of data having to be resent due to possible corruption.

Accessing wi-fi functionality through an Arduino unit can be accomplished by the addition of a zigbee. The zigbee module can be attached via shield to the Arduino unit. The range can be extended from several meters up to several hundred meters with the application of additional aerials.

Sensors Input & Output
It is clear with this project that there are going to be a number of input and outputs that will be handled by specific components. Below is a list of all the main components identified as well as information regarding them and their uses.

Potentiometer
In his 2015 book Arduino Essentials, Perea, highlights the versatility of the component, using it in many examples. The potentiometer is a variable resistor that when used in conjunction with one of the analogue read inputs on an Arduino unit allows the user to measure a value typically between 0 and 1023. The resistance is measured between the flow of current between the two outer legs with the actual resistance value measurement being taken the centre leg of the component.

Photoresistor
A photoresistor has been identified as the ideal sensor for device two, a light that is set to turn on when then ambient light level drops below a certain threshold. One patent for a photoresistor from the Tsinghua University (2015) describes the photoresistor as being made up of multiple layers stacked upon each other, using carbon nanotubes either side of a photosensitive material layer, where the resistance offered can then be measured.

Temperature Sensor
One of the most commonly available sensors identified that can be used in conjunction with an Arduino device is the TMP36 Temperature Sensor. The sensor is small in size and comprises of a small flat sided cylinder with three legs. In her blog Ada (2012) comments on one of the main issues around using the TMP36 sensor in that if you connect it the wrong way around, you can very easily burn it out and destroy the sensor.

Design & Method

The design process initially looked at the functionality specified in the brief for each device. After this information was gathered the next step was to select a suitable analogous that would allow the core functionality to be demonstrated and implemented on a smaller scale, ultimately allowing for a proof of concept to be demonstrated.

In the following sections you will find a more detailed predesign assessment of each of the devices required. However, the remote-control device has been placed last in this list as ultimately, its functionality requirements were identified as being dependent on the choices made regarding the other devices. In each instance a non-remote based prototype was built within Tinkercad (fig1). The approach was to start with each device, emulate the functionality required, then add the next device. By employing this process, it was possible to ensure that the required number of inputs and outputs on the Ardunio unit was not exceeded with the choices of inputs and outputs used for each device, individually as well as collectively. In addition to exploring the functionality that each device is required to have, multiple solutions are explored and detailed.

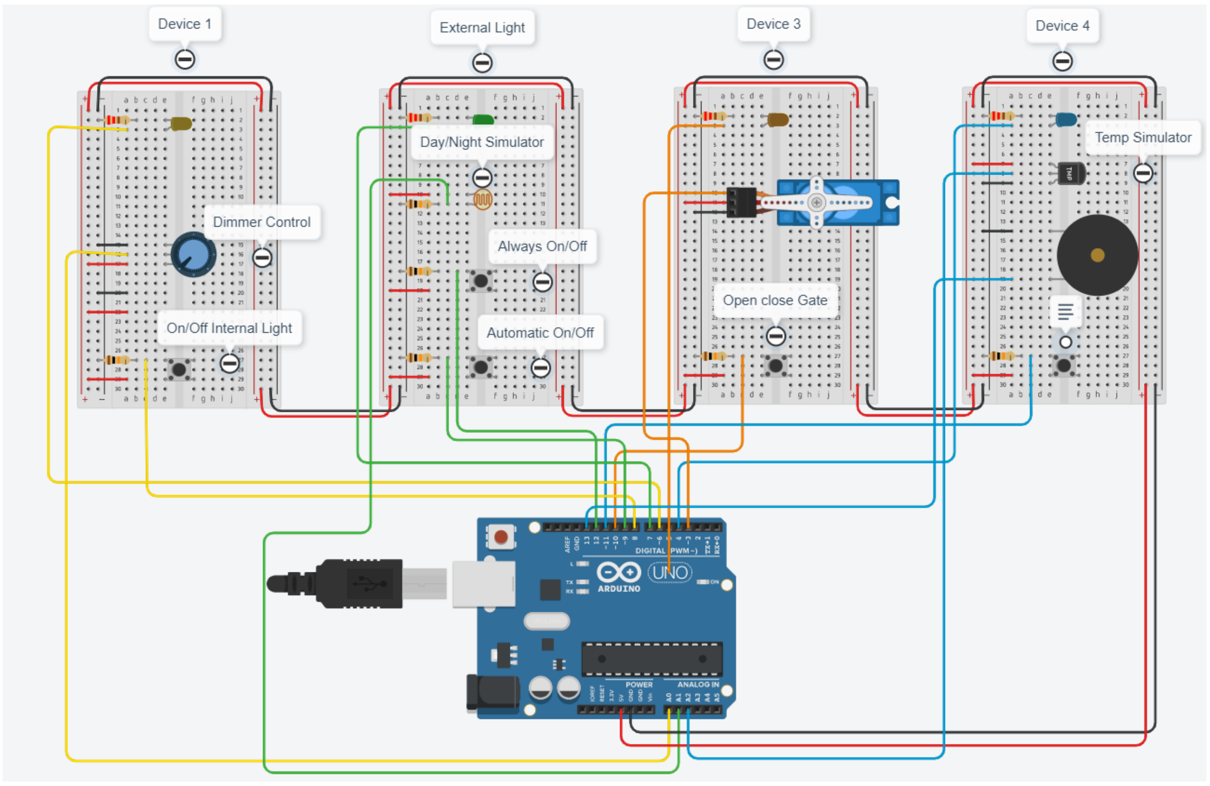

Figure 1 – Non Wireless four device build

Device 2

Original spec provided

"Device 2 is outside the house. It controls the outside light (turning it on automatically when the sun goes down and off when the sun comes up). It also controls the gates with a servo."

Clearly this is two separate devices, one to control an external light and a second to control a gate.

The light needs to be able to be turned on and off automatically based on when the sun comes up and goes down, this can be achieved by the use of a light sensor, commonly known as a photo-resistor, this will be in combination with software used to control the sensitivity so that when a threshold has been reached the light will activate and deactivate. An alternative that was considered, but not used, was to use the raw sunrise and sunset times from a reputable source such as that found on the website time and date .com (Sunrise and Sunset in the United Kingdom, 2020). Then use software to control the light in combination with an internal timer. However, this solution would add multiple points of possible failure such as a clock being reset, or the device not being able to read the time information from the source. It would also be fair to surmise that the software side of this option would equally be degrees more complex.

The device will need to report its state to the control device. Finally, it could be of additional benefit to have an override switch that allows for the light to be turned on and off, regardless of ambient light levels, this allows independent testing of the photo-resistor within the scope of the hardware system.

A simple LED will suffice to be used as the actual light, coupled with a photo-resistor controlled by the software and or user, Tinkercad allows for the user to simulate the input into the photoresistor by way of a slider (fig 2).

The second part of the brief describes a gate controlled by a servo. It is fair to assume that the basic and essential level of operation is to have the servo move the gate from closed to open and vice versa. As with the external light it is essential that the gate state be known within the system, this will allow the software to respond to any input correctly, specifically by changing the existing state to its opposite. The use of a servo, though small and clearly unable to move a gate under its own power, could be used however to control a switch, which would then in turn control a motor, which would then in turn move the gate. Consideration was given during the design process to if sensors were needed at the open and closed states of the gate, these could provide an additional layer of redundancy as well as more information to the controller.

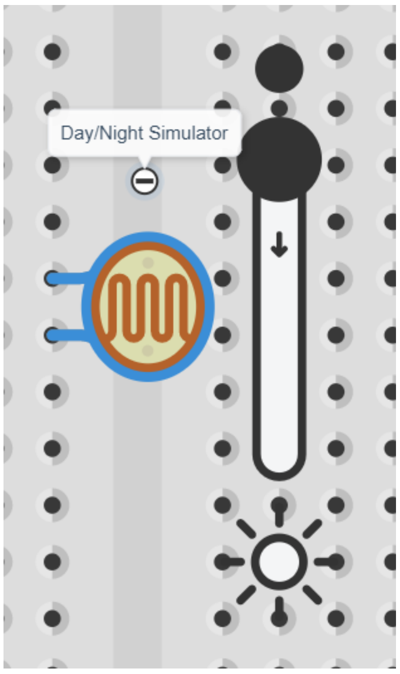

Figure 2 – Photoresistor

Device 3

Original spec provided

"Device 3 controls the interior light. The room has a simple (on/off) but also a dimmer function, meaning that the light level can be altered"

The functionality of this device can be described in three base statements. The light must be able to be turned on and off. The lights brightness level should be controllable by the user, with the remote. The light should be able to relay its state to the control.

A push button will be representative of an input control device as it is analogous of a common household light switch also being similar to touch screen button on an mobile app. This is because its use, is intuitive from a HCI point of view. As with the external light an LED will also serve as a suitable equivalent for the light source. The ability to control the lights luminosity will be achieved by incorporating a potentiometer (fig 3) into the system. The Ardunio is able to read in a value from the potentiometer between 0-1024, when reviewed in the software. With some simple code, it is possible to use this input value to control the output to the LED. As with the light sensor used in device 2, the Tinkercad software allows the user to manipulate the component and observe the results.

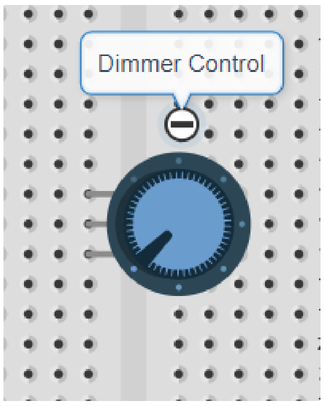

Figure 3 - Potentiometer

Device 4

Original spec provided

"Device 4 passive fire detection (infra-red). Detection is from one single source."

Of all the devices to be designed and created, this device proved to be the most difficult. The difficulty came not from the functionality that was requested, but instead from the limitations of using Tinkercad. Within the software there is no infra-red detection components that are available to be used. Instead the functionality of detecting a fire can only feasibly be simulated using a temperature sensor.

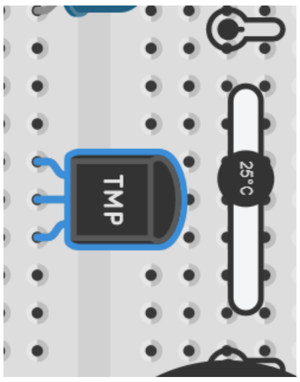

Figure 4 Temperature Sensor

The designed system will meet the requirements of the brief in that it will be passive. It will be possible to have the system read in from the sensor, continually, then when a threshold has been exceeded, a speaker can sound to alert the user. If this system were to be used in a real home setting, it may be worth considering the addition of a strobe like light to alert individuals who may have hearing problems. As with the light sensor used in device 2, the Tinkercad software allows the user to manipulate the temp sensor (fig 4) and observe the results.

Device 1

Original spec provided

"Device 1 is a remote control device. It can be used with serial monitor on a host PC, but higher marks will be given to those who use the LCD screen. (portable)"

From the spec it is clear that the remote-control device requires several inputs to control the slave devices, and a visual output to relay to the end user the state of the system at any given time. As discussed above in the device 3 section, buttons will be used to control the individual devices. There will need to be five buttons in total each performing the following functionality, depending on the device state. Specifically, each press of a button will look at the present state and through software and hardware change it to the second state associated with the button.

1. Internal Light on / Internal Light off
2. External Light always on / External Light always off
3. External Light auto mode on / External Light auto mode off
4. Gate open / Gate close
5. Fire detector on / Fire detector off

In addition to the inputs listed above there are other additional inputs that will be used. Though only one of which will be used on Device 1 (fig 5), the potentiometer, which is represented as a dial to control the luminosity remotely. The other inputs will be controlled via the Tinkercad web-based interface and will be the sensors shown in fig,2,3 and 4. Lastly the device states will be displayed to the user by means of an LCD screen, updated in real time by the software programmed into the Arduino. The screen is capable of displaying two lines of 8 common ascii characters, so consideration must be given to exactly what text is displayed.

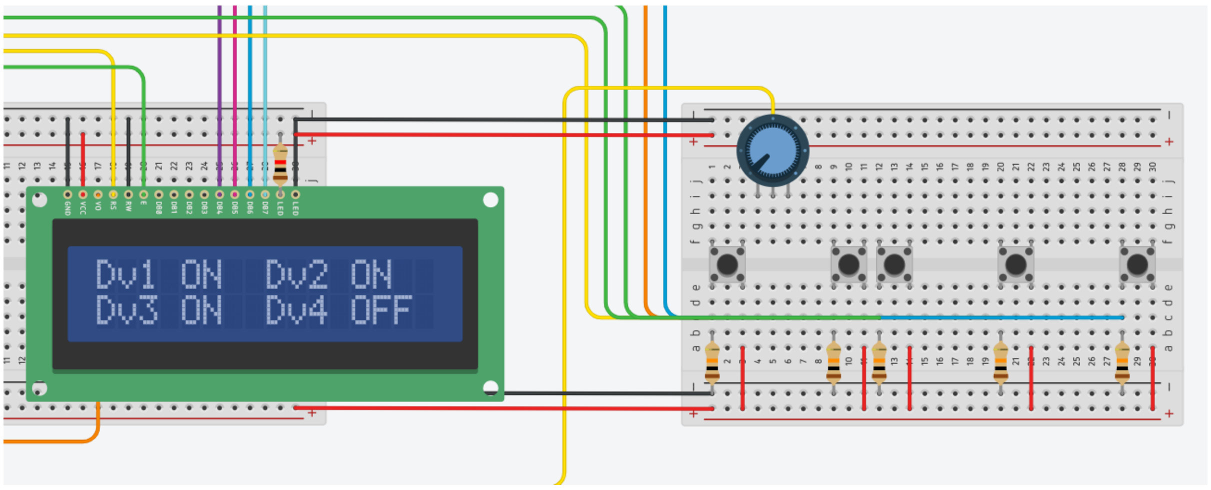

Figure 5 – LCD output and input controls

Implementation

Overview

All of the devices that were outlined in the brief were able to be implemented (fig 6). However, due to limitations of the Tinkercad software, some changes and concessions had to be made. There is no way within the Tinkercad software to simulate 'true' wireless communication between two Ardunios. Instead, it is possible to communicate using a serial connection between the master and slave Arduino units. From an implementation perspective, the serial connection from a software and hardware point of view is very similar to that which would be used should a Bluetooth or wi-fi based solution such as Zigbee be employed instead. Indeed in some cases it is as simple as plugging into the same ports as the serial connection wire on the Arduino unit. A complete list of all components used can be found within the appendix

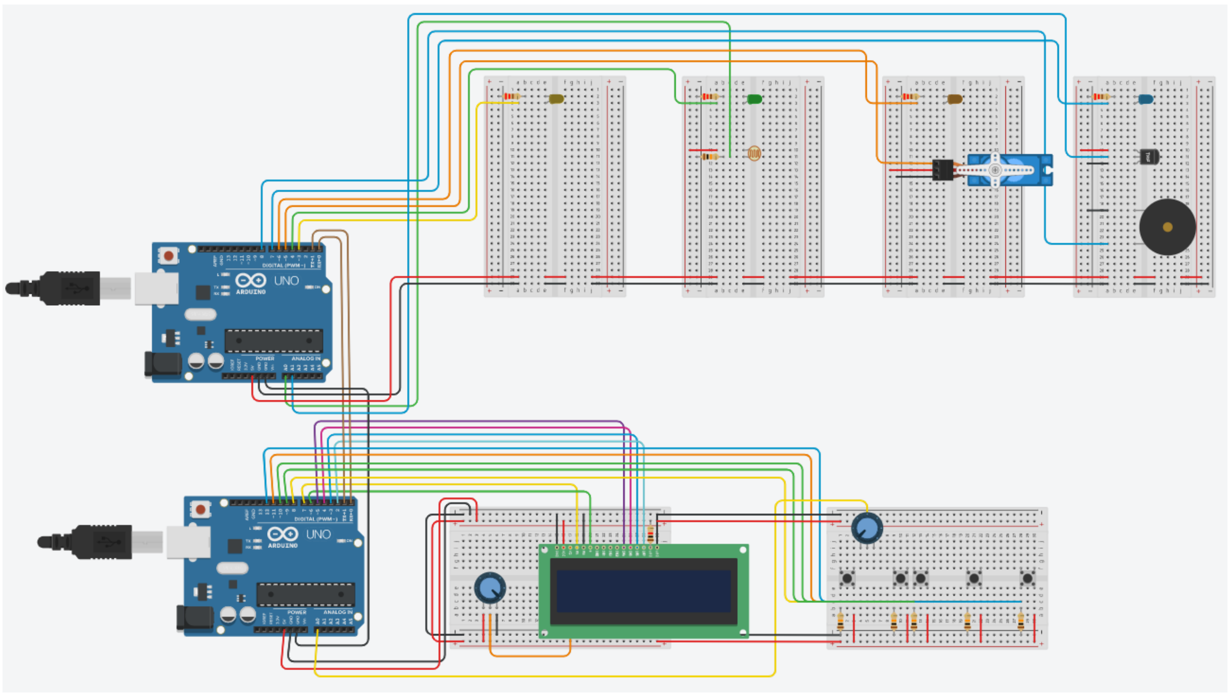

Figure 6 – Final master slave system in Tinkercad

Master and Slave device code

A complete version of the code used on the master device as well as the slave device can be found in the appendix.

In the first instance, the code used for each of the Arduino's was written using the Ardunio.cc IDE, integrated development environment, then transferred into Tinkercad by means of simple copy and paste. Overall, there was increased functionality present within the Arduino.cc IDE program such as error detection and formatting options that generally made implementing the software less arduous from a developmental perspective.

There are several main areas that each of the code files can be broadly broken down into,

- Libraries
- Variable declaration
- Setup
- Main loop
- Methods & Functions

Libraries

With the choice of components made for this project it was needed to use only two additional libraries, the first is for the LCD screen and the second is for the servo. Both libraries are commonly used and included with the Arduino.cc IDE as well as within the Tinkercad simulation software. The library lines of code are found at the top of each file and can be noted by the prefix #include.

Variable declaration

Across the two programs there are more than 60 global variables (34 master program, 27 slave program). When naming the variables there was a conscious choice to choose suitable, yet shortened descriptors as can be seen in the table below, this was to allow easier debugging and overall program flow.

Variable name	Full name
`int dv1btn1State = LOW;`	Device one Button one State
`int dv2btn2 = 10;`	Device two Button Two
`char device2AutoOn[5] = "5100";`	Device two Automatic Mode On

A mixture of variable types were created and used, from int variables being used to hold pin declarations to bool variables being used to hold device states. The variable declarations are found at the beginning of the code files before the setup section yet after the library statements.

Setup

The setup section of the code is an area where you are able to place code that is required for the system to function correctly, but it is only needed to be run once. On the master device code the initialization for the LCD screen can be found in here, as can the be the INPUT statements for the corresponding pins. On the slave device, the corresponding OUTPUT pins are set up as is the servo initialization. Finally, on both the serial log is started.

A statement such as **"pinMode(dv1btn1, INPUT);"** is calling the built in method pinMode, it is passing in an integer, which in this case is a global variable "dv1btn1", and then the second argument is either INPUT or OUTPUT, this is telling the Arduino to treat that pin in the corresponding fashion.

Main Loop

Each of the two Arduino's function by way of continuously processing the code contained in the main void loop function. As soon as they reach the bottom they simply start again at the top and continue in this cycle. For this reason, when creating the code, consideration was given to use of methods and functions external to void loop, allowing it to cycle more efficiently.

For example, the main loop on the master device is only five lines of code long, it calls a method for each of the four devices the details of which will be discussed in the next section, as well as one additional method that sends the current brightness that the potentiometer is being read at.

Whilst still not overly bloated the main loop on the slave device is slightly more involved, as it was not possible to create a method that would read the incoming serial communications and work successfully.

Testing

The process of testing the overall design as well as the individual devices was relatively straight forward. As the entire system was created within Tinkercad, it was possible to simply click the start simulation button and observe the behaviour of the system.

Each element of the system was tested individually in line with the aims of this project and found to be working correctly and as desired. The informal tests carried out per device were.

Device 1

- Does the light turn on and off?
- Is the light controllable with the potentiometer?

Device 2

- Does the light turn on and off?
- Does the light operate automatically?

Device 3

- Does the servo move to the open position?
- Does the servo move to the closed position?

Device 4

- Does the buzzer sound when a temperature threshold has been reached?

Remote device

- Do the buttons operate as desired?
- Does the LCD screen show the current state for each device?

Critical Evaluation

Overall, the project appears to have gone relatively well, with all aims and objectives being met. One of the main positives that has come out of this process is the flexibility and versatility of the TinkerCad software as a prototyping tool. Within a relatively short period of time, it was possible to create a home automation analogous system with almost all of the functionality of a real hardware-based system, but without any of the issues that are typically found within that process such as loose connections etc.

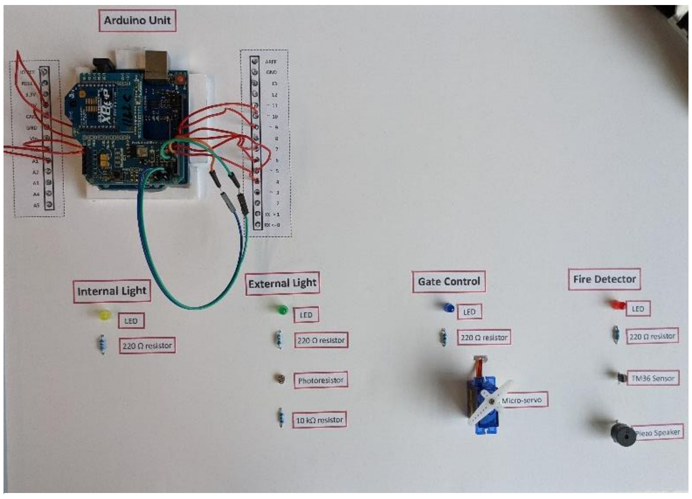

Figure 7

During this project as well as creating the home automation system in the virtual environment, substantial development also took place in recreating it using physical hardware (fig 7 & 8). Due to unrelated issues and hurdles external to the project the reproduction of the system using the actual hardware was not completed.

Therefore, the end result created represents everything that is possible using Tinkercad. However, it must be noted that the wireless communication elements were not "truly" shown, also the absence of the web app interface using a ESP8266 device or equivalent.

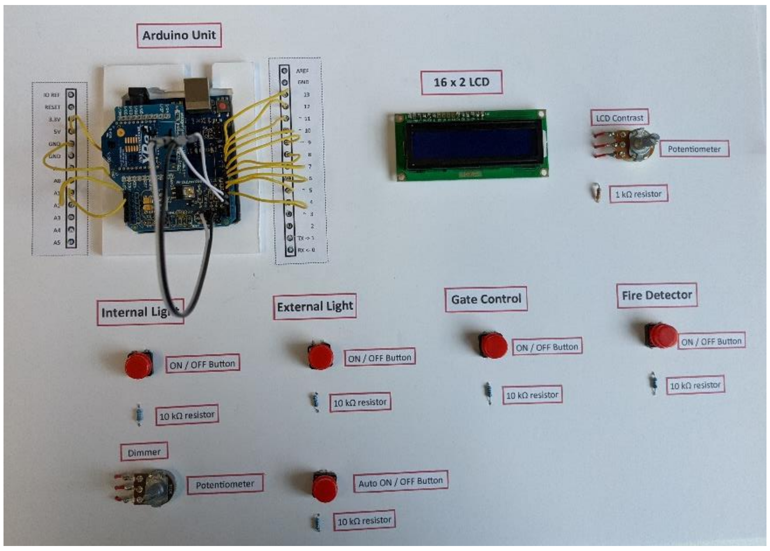

Figure 8

References & Bibliography

References

Goodwin, S. (2013) *Smart Home Automation with Linux and Raspberry Pi.* 2nd edn. Berkeley, CA: Apress.

Google (2021) *Nest Learning Thermostat.* Available at: https://store.google.com/product/nest_learning_thermostat_3rd_gen (Accessed: 12/01/21).

Lady Ada *TMP36 Temperature Sensor.* Available at: https://learn.adafruit.com/tmp36-temperature-sensor/overview (Accessed: Jan 13, 2021).

Perea, F. (2015) *Arduino essentials : enter the world of Arduino and its peripherals and start creating interesting projects.* 1st edn. Birmingham, England: Packt Publishing Ltd.

Philips (2021) *Smart Bulbs.* Available at: https://www.philips-hue.com/en-gb/products/smart-lightbulbs (Accessed: Jan 12, 2021).

Sonos *One: The Voice Speaker for Music Lovers.* Available at: https://www.sonos.com/en-gb/shop/one-white.html (Accessed: Jan 12, 2021).

TimeAndDate *Sunrise and Sunset in the United Kingdom.* Available at: https://www.timeanddate.com/astronomy/uk (Accessed: Jan 12, 2021).

Tsinghua University (2015) *Photoresistor*.

Bibliography

Faludi, R. (2011) Building wireless sensor networks . Sebastopol, CA: O'Reilly.

Appendix

Component list

Name	**Quantity**
Arduino Uno	2
Breadboard small	6
Potentiometer 250kΩ resistance	2
Push button	5
Temperature sensor TMP36	1
Resistor 10 kΩ	6
Resistor 1 kΩ	1
Resistor 220 Ω	4
LED	4
Micro servo	1
16 x 2 LCD Screen	1
Photoresistor	1
Piezo Speaker	1
Length of wire	70

Image list

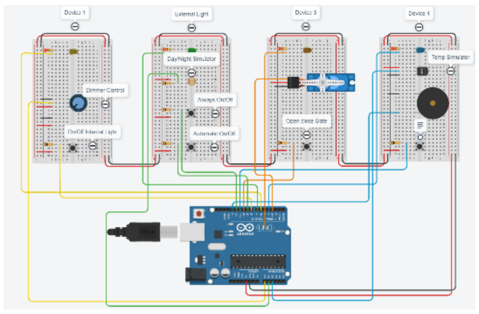

Figure 1 – Non Wireless four device build

Source: Screenshot taken from Tinkercad by author

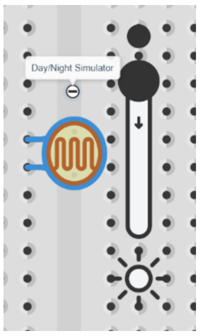

Figure 2 – Photoresistor

Source: Screenshot taken from Tinkercad by author

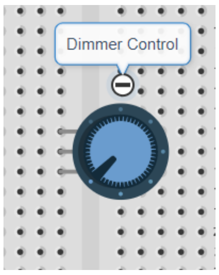

Figure 3 – Potentiometer

Source: Screenshot taken from Tinkercad by author

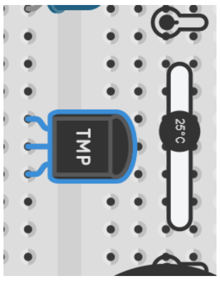

Figure 4 – Temperature Sensor

Source: Screenshot taken from Tinkercad by author

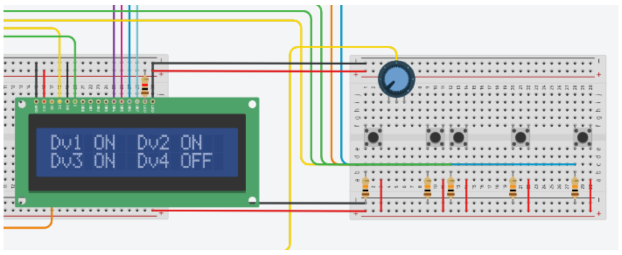

Figure 5 - LCD Output and input controls

Source: Screenshot taken from Tinkercad by author

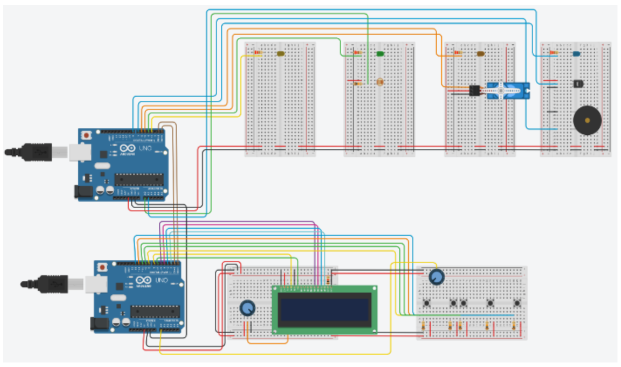
Figure 6 – Final master slave system in Tinkercad

Source: Screenshot taken from Tinkercad by author

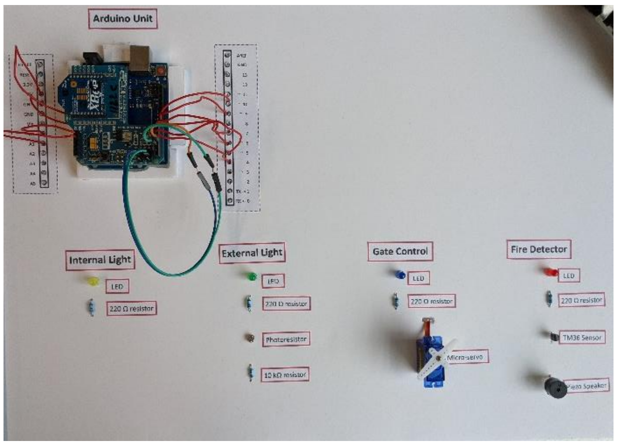

Figure 7 – Arduino slave device created by the author

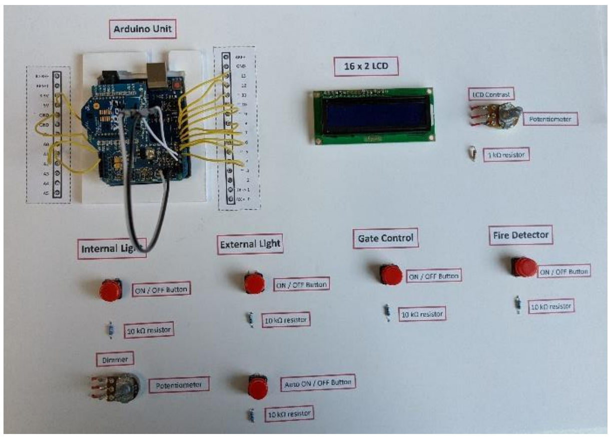

Figure 8– Arduino master device created by the author

Master device code part 1

```
/*
   Author David Pearson 1725412
   VIRUS Version 1
   This is the master control program for the 7CC003 module
   2020-2021
*/
// libraries to be used
#include <LiquidCrystal.h>
#include <Servo.h>

// lcd pin declarations
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);

// button input to pin declarations
int dv1btn1 = 8;
int dv2btn1 = 9;
int dv2btn2 = 10;
int dv3btn1 = 11;
int dv4btn1 = 12;

// button states and previous states
int dv1btn1State = LOW;
int dv1btn1Previous = LOW;
int dv2btn1State = LOW;
int dv2btn1Previous = LOW;
int dv2btn2State = LOW;
int dv2btn2Previous = LOW;
int dv3btn1State = LOW;
int dv3btn1Previous = LOW;
int dv4btn1State = LOW;
int dv4btn1Previous = LOW;

// potentiometer to pin
int dimmer = 0;

// booleans to hold each device state
bool dv1 = false;
bool dv2 = false;
bool dv3 = false;
bool dv4 = false;
bool dv2Auto = false;

//these deal with debounce on button presses
long time = 0;
long debounce = 200;

//Serial control messages
char device1On[5] = "1100";
char device1Off[5] = "1000";
char device2On[5] = "2100";
char device2Off[5] = "2000";
char device2AutoOn[5] = "5100";
char device2AutoOff[5] = "5000";
char device3On[5] = "3100";
char device3Off[5] = "3000";
char device4On[5] = "4100";
char device4Off[5] = "4000";

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // set up the pins to be either INPUT or OPUT
  pinMode(dv1btn1, INPUT);
  pinMode(dv2btn1, INPUT);
  pinMode(dv2btn2, INPUT);
  pinMode(dv3btn1, INPUT);
  pinMode(dv4btn1, INPUT);

  // set up the serial communication
  Serial.begin(9600);
}

/*
   This is the main loop for the program
   it simply calls the individual device functions
*/
void loop() {
  device1();
  sendBrightness();
  device2();
  device3();
  device4();
}
```

Master device code part 2

```
/*
   This function reads the potentiometer to a variable
   then converts that reading to a scale of 1-9 and returns
   that value
*/
int getBrightness() {
  int dimmerReading = analogRead(dimmer);
  int dimmerLevel = map(dimmerReading, 0, 1023, 1, 9);
  return dimmerLevel;
}
/*
   This function sends the brightness level from the porentiometer to the slave
   it first checks to see if device 1 is on, then it reads the brightness,
   depending on the level read in, then the coressponding message will be sent.
*/
void sendBrightness() {
  if (dv1 == true) {
    int brightness = getBrightness();
    switch (brightness) {
      case 1:
        Serial.write("1110", 5);
        break;
      case 2:
        Serial.write("1120", 5);
        break;
      case 3:
        Serial.write("1130", 5);
        break;
      case 4:
        Serial.write("1140", 5);
        break;
      case 5:
        Serial.write("1150", 5);
        break;
      case 6:
        Serial.write("1160", 5);
        break;
      case 7:
        Serial.write("1170", 5);
        break;
      case 8:
        Serial.write("1180", 5);
        break;
      case 9:
        Serial.write("1190", 5);
        break;
      default:
        break;
    }
  }
}
/*
   This function updates the LCD screen to the current state of
   each of the devices. It takes two parameters, the first spcifies
   the device, the second the state, with a 0 to represent off and a
   1 to represent on.
*/
void updateScreen(int device, int state) {
  switch (device) {
    case 1:
      if (state == 1) {
        lcd.setCursor(0, 0); lcd.print("Dv1 ON ");
      } else if (state == 0) {
        lcd.setCursor(0, 0); lcd.print("Dv1 OFF ");
      }
      break;
    case 2:
      if (state == 1) {
        lcd.setCursor(8, 0); lcd.print("Dv2 ON  ");
      } else if (state == 0) {
        lcd.setCursor(8, 0); lcd.print("Dv2 OFF ");
      }
      break;
    case 3:
      if (state == 1) {
        lcd.setCursor(0, 1); lcd.print("Dv3 ON ");
      } else if (state == 0) {
        lcd.setCursor(0, 1); lcd.print("Dv3 OFF ");
      }
      break;
    case 4:
      if (state == 1) {
        lcd.setCursor(8, 1); lcd.print("Dv4 ON ");
      } else if (state == 0) {
```

Master device code part 3

```
        lcd.setCursor(8, 1); lcd.print("Dv4 OFF ");
      }
      break;
    case 5:
      if (state == 1) {
        lcd.setCursor(8, 0); lcd.print("Dv2A ON ");
      } else if (state == 0) {
        lcd.setCursor(8, 0); lcd.print("Dv2A OFF");
      }
      break;
    default:
      lcd.setCursor(0, 0);
      lcd.print("Dv1 ??? Dv2 ???");
      lcd.setCursor(0, 1);
      lcd.print("Dv3 ??? Dv4 ???");
      break;
  }
}
/*
   Device 1 function logic
   1. read the state of the button, on press, checking for debounce :-
   2. if the current state of the device is on then set it to off.
   3. update the LCD display to show the device state.
   4. send the device 1 off message via serial write function
   5. else if the current state of the device is off then set it to on.
   6. update the LCD display to show the device state.
   7. send the device 1 on message via serial write function
   8. update the button state
*/
void device1() {
  dv1btn1State = digitalRead(dv1btn1);
  if (dv1btn1State == HIGH && dv1btn1Previous == LOW && millis() - time > debounce) {
    if (dv1 == true) {
      dv1 = false;
      //Serial.println("Internal Light Off");
      updateScreen(1, 0);
      Serial.write(device1Off, 5);
    } else {
      dv1 = true;
      //Serial.println("Internal Light On");
      updateScreen(1, 1);
      Serial.write(device1On, 5);
    }
    time = millis();
  }
  dv1btn1Previous == dv1btn1State;
}
/*
   Device 2 function logic
   1. read the state of the always on button, on press, checking for debounce :-
   2. if the current state of the device is on then set it to off.
   3. update the LCD display to show the device state.
   4. send the device 2 off message via serial write function
   5. else if the current state of the device is off then set it to on.
   6. update the LCD display to show the device state.
   7. send the device 2 on message via serial write function
   8. update the button state
*/
void device2() {
  //read in the always on off button
  dv2btn1State = digitalRead(dv2btn1);
  if (dv2btn1State == HIGH && dv2btn1Previous == LOW && millis() - time > debounce) {
    if (dv2 == true) {
      dv2 = false;
      //Serial.println("External Light Off");
      updateScreen(2, 0);
      Serial.write(device2Off, 5);
    } else {
      dv2 = true;
      //Serial.println("External Light On");
      updateScreen(2, 1);
      Serial.write(device2On, 5);
    }
    time = millis();
  }
  //read in the auto on off button
  dv2btn2Previous == dv2btn2State;
  dv2btn2State = digitalRead(dv2btn2);
  if (dv2btn2State == HIGH && dv2btn2Previous == LOW && millis() - time > debounce) {
    if (dv2Auto == true) {
      dv2Auto = false;
      //Serial.println("External Light Off");
      updateScreen(5, 0);
      Serial.write(device2AutoOff, 5);
```

Master device code part 4

```
    } else {
      dv2Auto = true;
      //Serial.println("External Light On");
      updateScreen(5, 1);
      Serial.write(device2AutoOn, 5);
    }
    time = millis();
  }
  dv2btn2Previous == dv2btn2State;
}

void device3() {
  dv3btn1State = digitalRead(dv3btn1);
  if (dv3btn1State == HIGH && dv3btn1Previous == LOW && millis() - time > debounce) {
    if (dv3 == true) {
      dv3 = false;
      //Serial.println("Gate closed");
      updateScreen(3, 0);
      Serial.write(device3Off, 5);
    } else {
      dv3 = true;
      //Serial.println("Gate open");
      updateScreen(3, 1);
      Serial.write(device3On, 5);
    }
    time = millis();
  }
  dv3btn1Previous == dv3btn1State;
}
void device4() {
  dv4btn1State = digitalRead(dv4btn1);
  if (dv4btn1State == HIGH && dv4btn1Previous == LOW && millis() - time > debounce) {
    if (dv4 == true) {
      dv4 = false;
      //Serial.println("Fire detector off");
      updateScreen(4, 0);
      Serial.write(device4Off, 5);
    } else {
      dv4 = true;
      //Serial.println("Fire detector on");
      updateScreen(4, 1);
      Serial.write(device4On, 5);
    }
    time = millis();
  }
  dv4btn1Previous == dv4btn1State;
}
```

Slave device code part 1

```
/*
   Author David Pearson 1725412
   VIRUS
   This is the slave control program for the 7CC003 module
*/
// libraries to be used
#include <Servo.h>

// device pin declarations
int device1LED = 3;
int device2LED = 4;
int device2LightSensor = 0;
int device3LED = 6;
Servo servo_5;
int device4LED = 7;
int device4TempSensor = 1;
int device4Speaker = 8;

// LED States
int stateDevice1LED = LOW;
int stateDevice2LED = LOW;
int stateDevice3LED = LOW;
int stateDevice4LED = LOW;

// booleans to hold each device state
bool dv1 = false;
bool dv2 = false;
bool dv3 = false;
bool dv4 = false;
bool dv2Auto = false;

//boolean to hold light level
bool day = false;

//Strings used for message conversion
char message[5];
String deviceIDChar = "";
String deviceStateChar = "";
String deviceOneBrightnessChar = "";
String deviceTwoAutoChar = "";

//ints used for message conversion
int deviceID = 0;
int deviceState = 0;
int deviceOneBrightness = 0;
int deviceTwoAuto = 0;

void setup()
{
  // set up the pins to be either INPUT or OPUT
  pinMode(device1LED, OUTPUT);
  pinMode(device2LED, OUTPUT);
  pinMode(device3LED, OUTPUT);
  pinMode(device4LED, OUTPUT);
  pinMode(device4Speaker, OUTPUT);
  servo_5.attach(5);
  // set up the serial communication
  Serial.begin(9600);
  servo_5.write(0);
}
void loop()
{
  //read in the message from the master
  Serial.readBytes(message, 5);

  //seperate of the first character as the device id
  //convert the char to an int
  deviceIDChar = message[0];
  deviceID = deviceIDChar.toInt();

  //seperate of the second character as the device state
  //convert the char to an int
  deviceStateChar = message[1];
  deviceState = deviceStateChar.toInt();

  //seperate of the third character as brightness level
  //convert the char to an int
  deviceOneBrightnessChar = message[2];
  deviceOneBrightness = deviceOneBrightnessChar.toInt();

  //seperate of the forth character as device two
  //automode state convert the char to an int
  deviceTwoAutoChar = message[3];
  deviceTwoAuto = deviceTwoAutoChar.toInt();
```

Slave device code part 2

```
//auto mode for external light
  if (dv2Auto == true) {
    checkLight();
    if (day == false) {
      digitalWrite(device2LED, HIGH);
    } else if (day == true) {
      digitalWrite(device2LED, LOW);
    }
  }
  //update the brightness of the internal light
  if (deviceOneBrightness > 0 ) {
    setBrightness(deviceOneBrightness);
  } else {
    readMessage(deviceID, deviceState);
    //Serial.println(message); //this is here for debugging
  }
  //Check for fire
  checkForFire();
}
/*
   This function takes in three paramters the first is the device and the second is the state
   and the third is the auto function for device 2, then using a swicth statement turns on
   or off the corressponding device to the message recieved
*/
void readMessage(int device, int state ) {
  switch (device) {
    case 1:
      if (state == 0) {
        dv1 = false;
        stateDevice1LED = LOW;
        digitalWrite(device1LED, stateDevice1LED);
      } else if (state == 1) {
        dv1 = true;
        stateDevice1LED = HIGH;
        //digitalWrite(device1LED, stateDevice1LED);
      }
      break;
    case 2:
      if (state == 0) {
        dv2 = false;
        dv2Auto = false;
        stateDevice2LED = LOW;
        digitalWrite(device2LED, stateDevice2LED);
      } else if (state == 1) {
        dv2 = true;
        stateDevice2LED = HIGH;
        digitalWrite(device2LED, stateDevice2LED);
      }
      break;
    case 3:
      if (state == 0) {
        dv3 = false;
        stateDevice3LED = LOW;
        digitalWrite(device3LED, stateDevice3LED);
        servo_5.write(0);
      } else if (state == 1) {
        dv3 = true;
        stateDevice3LED = HIGH;
        digitalWrite(device3LED, stateDevice3LED);
        servo_5.write(180);
      }
      break;
    case 4:
      if (state == 0) {
        dv4 = false;
        stateDevice4LED = LOW;
        digitalWrite(device4LED, stateDevice4LED);
      } else if (state == 1) {
        dv4 = true;
        stateDevice4LED = HIGH;
        digitalWrite(device4LED, stateDevice4LED);
      }
      break;
    case 5:
      if (state == 0) {
        dv2Auto = false;
      } else if (state == 1) {
        dv2Auto = true;
      }
      break;
    default:
      break;
  }
}
```

Slave device code part 3

```
/*
   This function takes in the recieved brightness level then maps
   it back to a scale od 0-255 before writing it to the LED
*/
void setBrightness(int level) {
  int bright = map(level, 1, 9, 0, 255);
  analogWrite(device1LED, bright);
}
/*
   This function checks the light level and amends a boolean to
   reflect day or night;
*/
void checkLight() {
  int photoRead = analogRead(device2LightSensor);
  if (photoRead < 900) {
    day = false;
  } else {
    day = true;
  }
}
void checkForFire() {
  if (dv4 == true) {
    float voltage = analogRead(device4TempSensor) * 0.004882814;
    float degreesC = (voltage - 0.5) * 100.0;
    if (degreesC > 37) {
      digitalWrite(device4Speaker, HIGH);
      delay(100);  //delay half a second
      tone(12, 10000, 100);
    } else {
      digitalWrite(device4Speaker, LOW);
    }
  }
}
```