Northern Nomad Tiny House Systems Integration

By

Josh Campitelli

Ryan Ribeiro

Arsalan Sadiq

Supervisor: Professor Bailey

A report submitted in partial fulfillment of the requirements of the SYSC4907 Engineering Project

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University

Tuesday, April 9th, 2019

Abstract

The Northern Nomad Tiny House project is a fourth-year capstone project, under the Department of Systems and Computer Engineering, which has been speculated to achieve net zero water and energy consumption. The Tiny House is equipped with a combination of IoT and non-Iot devices which aim to prove this speculation correct. The Tiny House requires a central hub where the data read from these devices shall be displayed. The Home Assistant Operating System was selected as the choice of a central hub and flashed to the Raspberry Pi's SD card. The non-IoT devices were re-engineered using the NodeMCU board to provide the capability to transmit and receive data. Data collected from the NodeMCUs was published to the Raspberry Pi using the MQTT communication protocol. Combining these three accomplishments, data could be retrieved from non-IoT devices and the outputs displayed on the Home Assistant UI. Additionally, the collected data was able to be exported in a JSON file for further research and development on the practicality of net zero technologies.

Acknowledgements

There are several people who we wish to thank for their contributions. For starters, we wish to thank Professor Bailey for his ongoing support and assistance throughout the project. There were many unforeseen complications throughout the project that required assistance or guidance to solve. Professor Bailey was always there to offer his time and to go out of his way to make sure we were getting the information we needed to succeed. Next, we would like to thank Seungyeon Hong, Eric Ho, and Josh Reinhart — three graduate students who previously worked on the project — for providing information about the state of the project and for answering questions we had throughout the course of the project. Similarly, we thank Brock Conely and Chris Baldwin — two graduate students who were responsible for the wiring of the house — for providing information and answering questions. Finally, we wish to thank Jerry Buburuz — a computer network administrator for the Carleton Systems and Computer Engineering technical staff — for assisting us in setting up various parts of our project to work with the school environment. The contributions made by each of these individuals had an impact on our project and we appreciate the assistance they provided.

Table of Contents

List of Figures

Chapter 1 – The Big Picture

1.1 – Introduction

The Northern Nomad Tiny House project — henceforth referred to as the Tiny House — is a fourth-year capstone project that set out to design and build a small, portable, smart home that achieves a net zero water and energy consumption in order to have a low environmental impact. The Tiny House makes use of solar panels for energy generation, Tesla batteries for energy storage, dehumidifiers for water generation, water tanks for water storage, and wall sensors for monitoring the heat loss through the insulation. Additionally, the Tiny House is outfitted with Internet of Things (IoT) devices such as smart locks, smart bulbs, and light dimmers that can be connected and controlled through an open-source home automation project called Home Assistant. A key element of the Tiny House is that it has been designed to function without a connection to the internet while still providing the functionalities expected of a smart home.

1.2 – Problem Motivation

One of the main problems involved with setting up the Tiny House to work without an internet connection is figuring out a way to have all the IoT devices communicate with Home Assistant. Since IoT devices require a connection to a network in order to send and receive data, it is necessary for us to figure out how to setup a network that doesn’t require an internet connection. Additionally, not all the devices in the house are IoT, so we have to figure out a way to connect with these devices too. The main reason for establishing a network for all devices is to allow for data to be read in and fed to Home Assistant so that the occupants of the Tiny House can monitor and control the house. Going one step further, the collection of data is important for the research teams to be able to monitor the house and determine if the Tiny House is meeting its net zero energy and water consumption goals.

1.3 – Problem Statement

Our project team settled on two core problems that we would work towards solving over the course of this project. The first problem involves connecting all IoT and non-IoT devices to a wireless network so that we could send and receive data from Home Assistant. The second problem involves the collecting and storing of data from the smart devices installed in the Tiny House. This is so we can monitor the state of the house and provide research material for graduate students to analyze in and determine if the Tiny House achieves its net zero goals.

1.4 – Problem Solution

To address the first problem of setting up a network to connect all devices, we started by setting up a wireless network that would allow for communication between devices with the need of an internet connection. This type of network is called a wireless local area network (WLAN) and it allows for any device to connect and communicate with the other devices connected to the network. After establishing a WLAN, we needed to figure out a way to incorporate the non-IoT devices that were previously installed into devices we could receive information from. The solution we found was to use the NodeMCU board because it would allow for us to wirelessly send data over a network from any device connected to the NodeMCU. Finally, we needed to find a way to record and monitor all the data that we were collecting from the devices. Fortunately, Home Assistant had the option for us to create our own configurations for specific devices we had connected and for us to easily export the data being collected in an easily readable JSON file.

1.5 – Accomplishments

For the Tiny House project, we set out to accomplish three main goals. The first goal was to setup and configure Home Assistant in order for us to be able to display data and control the smart technologies in the Tiny House. We achieved this by installing Home Assistant on the Raspberry Pi, and then we customized the user interface through the configuration files. Our second goal was to find a way to retrieve data from non-IoT devices so that we could display it on the Home Assistant interface. We accomplished this by using the NodeMCU to act as a way to connect non-IoT devices to a network, thus allowing us to transmit data over a network. Our final goal was to establish a way to connect the devices with Home Assistant over a network. This goal was achieved by programming the NodeMCUs to use the MQTT communication protocol to connect to the Raspberry Pi, thus connecting to Home Assistant.

1.6 – Overview

The following sections of the report are as follows. Chapter 2 which discusses the details of an engineering project, specifically the Health and Safety precautions taken in the lab when dealing with circuits (2.1), Engineering Professionalism when communicating with colleagues and supervisors (2.2), Project Management roles which were taken by group members (2.3), the project's Suitability for our group (2.4), and finally each of the group member's individual contributions (2.5). Chapter 3 which defines the technical terminology and background which serves to remove any ambiguous terms or definitions, and to provide some insight on the background of the Tiny House. Chapter 4 which provides insight on the Project Details and goals which were solved, this section is broken down into the three goals which our group solved. The setup and configuration of Home Assistant on a Raspberry Pi (4.2), the extractions of data from sensors using NodeMCU boards (4.3), finally the connection between the Raspberry Pi and NodeMCU using MQTT (4.4). Chapter 5 discusses the project moving forward and our recommendations for future teams who will be partaking in the Software Development for the Northern Nomad Tiny House. This overview serves to inform the user of the upcoming chapters, and to provide an overview of the report in its entirety.

Chapter 2 – The Engineering Project

2.1 – Health and Safety

Health and Safety responsibilities were adhered to during the research and development of the fourth-year project; operations regarding this project were carried out in the Mackenzie Lab 4836. The health and safety precautions pertaining to Carleton Labs regarding protective gear weren't applicable to our project. However, in our solution our group worked with low-voltage circuits; in this we were mindful of recommended safety procedures. First, when creating circuits, a consistent wiring color-scheme was always followed to minimize the risk involved. Prior to powering any circuit, the polarity was double checked, and circuit diagrams were revisited to ensure correctness. When testing circuits, a Voltmeter was used to measure the current through each of the components and pins on the microcontrollers to ensure correctness. Prior to powering any LEDs, a current limiting resistor was verified as to prevent overheating and burning out the LED. Finally, abiding by the Ontario Regulation 851 Section 131 regarding food and beverages, we did not enter the lab with food or beverages and remained in the designated eating areas for these times.

2.2 – Engineering Professionalism

As fourth-year Software Engineering students at Carleton University we have many Professional responsibilities which we must abide by when interacting with colleagues and supervisors. The fourth-year projects are team based, and communication among team members is crucial to the productivity as we learned in ECOR 4995. Our team addressed this difficulty through several weekly meetings, during which we discussed issues such as our steps moving forward in the project, issues holding us back, and progress which had been made. As taught in CCDP 2100, our email correspondence with our supervisor remained professional and courteous, and refrained from using unprofessional language. During the Oral Presentation and Project Demo at the Poster Fair, each member of our team dressed in business attire to remain professional. Finally, we meet weekly with our project supervisor where we were always punctual and showed up on time.

2.3 – Project Management

The fourth-year project is a long-term project which required each team member to perform the role of project management in some sense. Delegation was used when completing several deliverables, and during development in general, where one member had to distribute the tasks among the team and ensure each person had completed their task. In addition, soft-deadlines had to be set by one member, or mutually agreed upon, while remaining conscious of the timelines and upcoming deadlines. A Slack messenger group was also created by a team member, along with specific topics for discussion regarding issues or deliverables in this application were established. Moreover, budget was a factor which had to be managed, in the sense that each item purchased went through a cost-benefit analysis process. The collection of receipts for each item purchased was also maintained by a group member. The overall documentation of the project was another form of Project Management, as with long projects such as these fine details tend to be forgotten. A GitHub repository was made as a solution to this. In addition, the source-code used in our development process was also uploaded to this repository. Finally, when conflicts in the project arose, we performed conflict resolution. Conflict resolution was in the form of a group meeting where we discussed the pros and cons, which were then brought to our supervisor for consideration and a final judgement.

2.4 – Justification of Suitability for Degree Program

Each of the team members — Josh Campitelli, Ryan Ribeiro, and Arsalan Sadiq — are in Software Engineering. This project is suitable as a fourth-year capstone project as the problem which this project aimed to solve was almost entirely software based. Even though the project consisted of some rudimentary circuit design and analysis, the concepts required were covered in ELEC 2501 which was a requirement for Software Engineering. The remaining parts of the project were written in programming languages such as C, Python and markup languages such YAML, XML, JSON. As part of the Software Engineering curriculum, these languages and the concepts required to understand them were covered extensively through several courses. Therefore, this project was suitable to express our knowledge, and problem-solving skills obtained over the past four years of engineering at Carleton.

2.5 – Individual Contributions

2.5.1 – Project Contributions

The individual contributions for this project regarding the design, development, and implementation processes were equally distributed among the team members. Each member of the team worked together on all parts of the project at some point in time. However, each member performed different duties regarding the management of the project. Josh Campitelli delegated tasks during the development and testing of the Home Assistant and created the GitHub repository for version control and documentation. Ryan Ribeiro delegated responsibilities for the circuit design and analysis as well as the cost-analysis and receipt collection for purchasing devices. Arsalan Sadiq created the Slack messenger group, maintained the discussion topics and set soft-deadlines prior to each deliverable. Project Management was an important factor for a large project and each group member contributed equally.

2.5.2 – Report Contributions

Josh Campitelli contributed to the report by writing section 1.6 of Chapter 1, Chapter 2, section 4.2 in the Project Details section, and contributed to Chapter 5. Ryan Ribeiro wrote Chapter 1, excluding section 1.5 and 1.6, Chapter 3, section 4.3 in the Project Details section, and contributed to Chapter 5. Arsalan Sadiq wrote section 1.5 of Chapter 1, section 4.4 in Project Details and formatted the figures and references.

Chapter 3 – Technical Terminology and Background

3.1 – Terminology

There are a few terms that are used in the background section that require a formal definition.

3.1.1 – IoT Devices

IoT stands for Internet of Things, as defined in the introduction. These are devices that can communicate and send data over the internet [1]. IoT devices can also be controlled remotely by sending data back to the device over a network. In the scope of this project, any device that can communicate over a network, without an internet connection, is considered an IoT device. While this strays from the traditional definition, it still holds true in spirit as the project goal is to move away from using the internet entirely.

3.1.2 – Smart Technology

Smart technology is a non-specific umbrella term for any technology that helps with the automation of a smart home [2]. For example, the Philips Hue light bulbs in the house are considered a form of smart technology as they can be controlled, and automated, through Home Assistant. For the purposes of our report, the terms smart technology and smart devices can be used interchangeably.

3.1.3 – Net Zero

Net zero, in the scope of this project, refers to the consumption and production of energy and water. The goal of being net zero means to produce and consume energy and water a rate such that the production and consumption rates are equal.

3.2 – Background

The idea behind introducing this IT infrastructure into the Tiny House is to help with automation and efficient use of resources and energy. For example, there are window sensors installed in the Tiny House that will send a signal to Home Assistant and provide a warning to the occupant that they have left their window open. This feature is helpful for notifying the occupants that they could potentially be wasting energy cooling down their house when the windows are letting in the heat. In a house without this IT infrastructure there would be no safe guard against this. Having this infrastructure also allows for the constant monitoring of the Tiny House to ensure that the energy and water consumption are in in line with our net zero goals.

In figure 1 below, we have shown the IT infrastructure for the Tiny House when we began the project. This diagram was created as a combined effort with the other team working on the Tiny House project. As the legend depicts, there are four colours used to represent the different stage that each piece of infrastructure is at in terms of being implemented. Green denotes that the technology has been installed and is operational. Yellow denotes that the technology is installed, but not necessarily working. Red denotes technology that was promised, but not actually installed in the house at the start of the project. Finally, blue denotes that the technology has been installed, but it isn't of any use. For example, the EcoBee, a fancy thermostat, has been installed in the house but requires an internet connection. Since it requires an internet connection, it is useless to us.

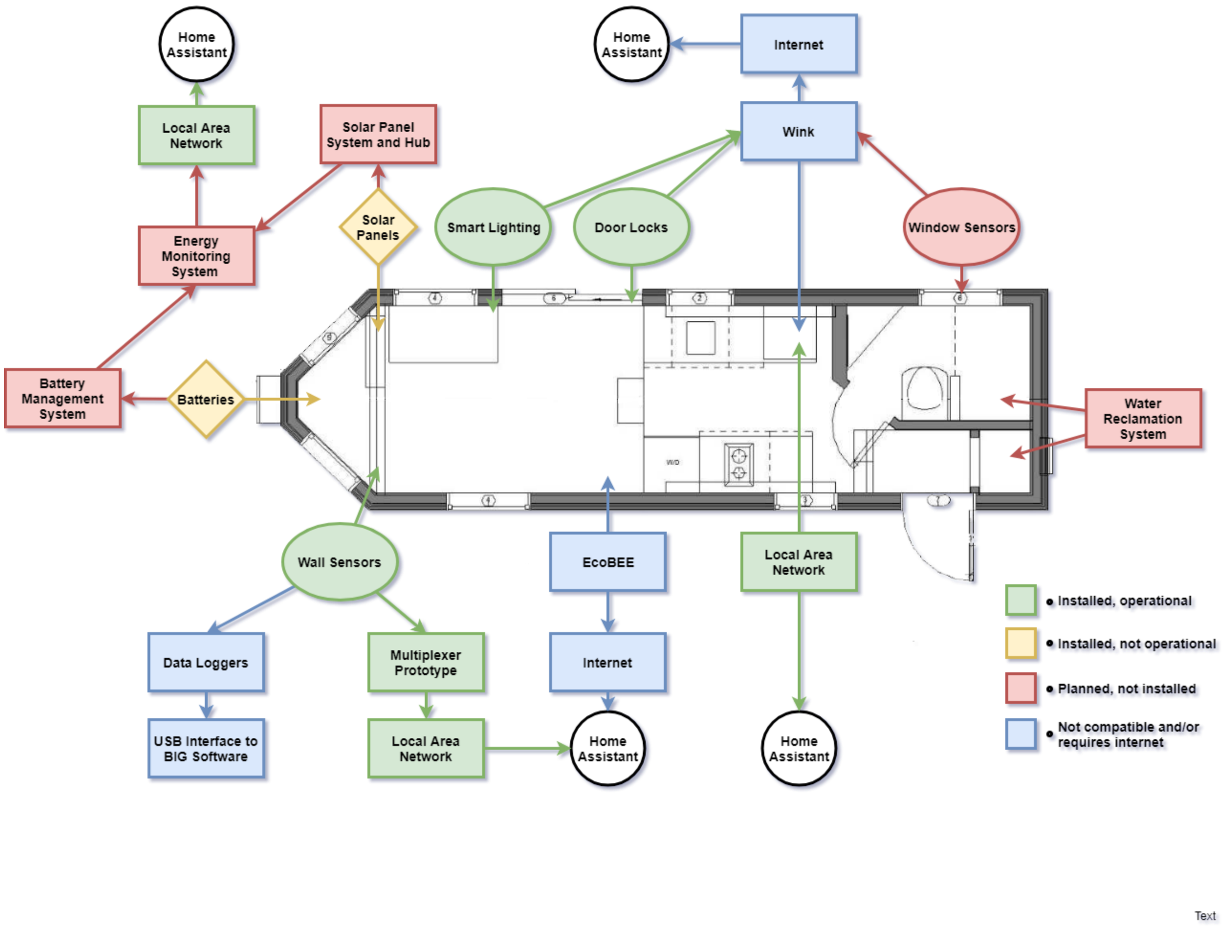


Figure 1: Expected System Overview of the Tiny House

In figure 2 below, we have highlighted which systems in the Tiny House we worked on for the project. We chose to focus on setting up a network that would remove the need to have any internet connection in the house. We would implement our own version of a window sensor and light sensor, explained in section 4.3.3, and we would be able to cut out the internet and Wink by using the NodeMCU, explained in section 4.3.

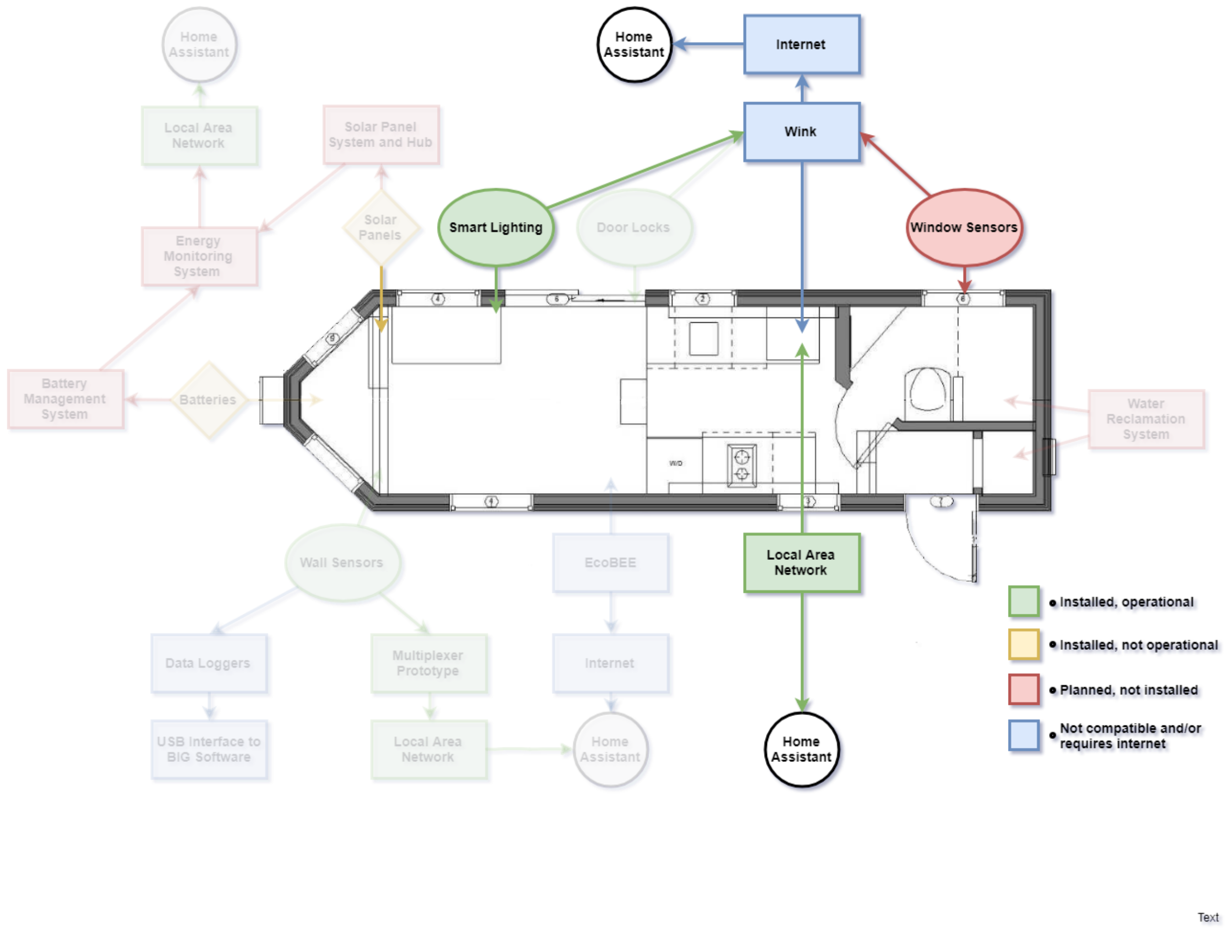


Figure 2: Partial System Overview of the Tiny House

Chapter 4 – Project Details

4.1 – Problem Overview

The goals for this project were centralized around the desire to automate and monitor the Tiny House and its smart devices. Regarding this, the first goal proposed to our group was a central hub, Home Assistant, to perform monitoring and automation of the various smart devices. Secondly, the sensors in the house were non-IoT, thus retrieval of the data was another problem. Finally, establishing the link between the Home Assistant Hub and the retrieved data, a communication protocol which was suitable for the situation had to be chosen. The following sections shed light on our solution, problems encountered, and alternatives pertaining to these goals.

4.2 – Home Assistant

4.2.1 – About Home Assistant

Home Assistant is an Open Source project on GitHub with over a thousand contributors worldwide. The goal of the project is to put local control of your system first and ensure privacy in your automation tools [3]. Home Assistant is an Event-Driven platform which integrates all the smart devices in your home, provides the ability to develop automations, and is highly configurable. Home Assistant was chosen because of these attributes as it directly solves the need for the project to have one central home automation tool. Home Assistant’s operating System is called Hass.io which is their all in one solution to turning a Raspberry Pi or another device into a home automation hub. A Raspberry Pi is a low-cost, powerful, credit card sized computer which in the case of model B+ is equipped with a WIFI chip and Ethernet port [4]. The Raspberry Pi's SD card is flashed with the Hass.io operating system which can then be booted from to display the Home Assistant Dashboard. The Home Assistant Dashboard, available in Appendix A, is the user interface for all the devices that have been connected to Hass.io and automations which have been created. Hass.io ensures that the Home Assistant software is installed and updated correctly, which can be configured through the Home Assistant UI. After installing Home Assistant on a Raspberry Pi the UI is available at hassio.local:8123 [5]. Hass.io generates a configuration.yaml file which is used to configure the UI and connected devices. YAML (YAML Ain't Markup Language) allows developers to provide powerful configuration settings without learning more complex programming languages [6]. Using YAML configuration files the connection to smart devices can be established, groups of devices and pages can be created on the dashboard, and much more.

4.2.2 – Rationale for Using Home Assistant

Home Assistant was initially recommended by the project team working on the house prior to us, however it does appropriately suit the problem at hand. Home Assistant was chosen for this project mainly because it is a powerful Open Source home automation software with a very active community on their forums and GitHub [7]. Home Assistant is powerful in the sense that it allows you to connect hundreds of IoT devices to the Raspberry Pi which is compatible with several different communication protocols. The main protocol which was used is the MQTT communication protocol which is discussed in section 4.4. Home Assistant's compatibility with this protocol allowed many devices in the house to communicate their raw data to the Raspberry Pi. Home Assistant is also very useful as it logs all the data, which is sent to the Raspberry Pi, and may persist for months depending on the size of the SD card. This data is stored in JSON format which is convenient for exporting and using by analysis software. Home Assistant provides visualizations of the data which has been persisted, Hass.io generates graphs which are displayed on the Dashboard for the user.

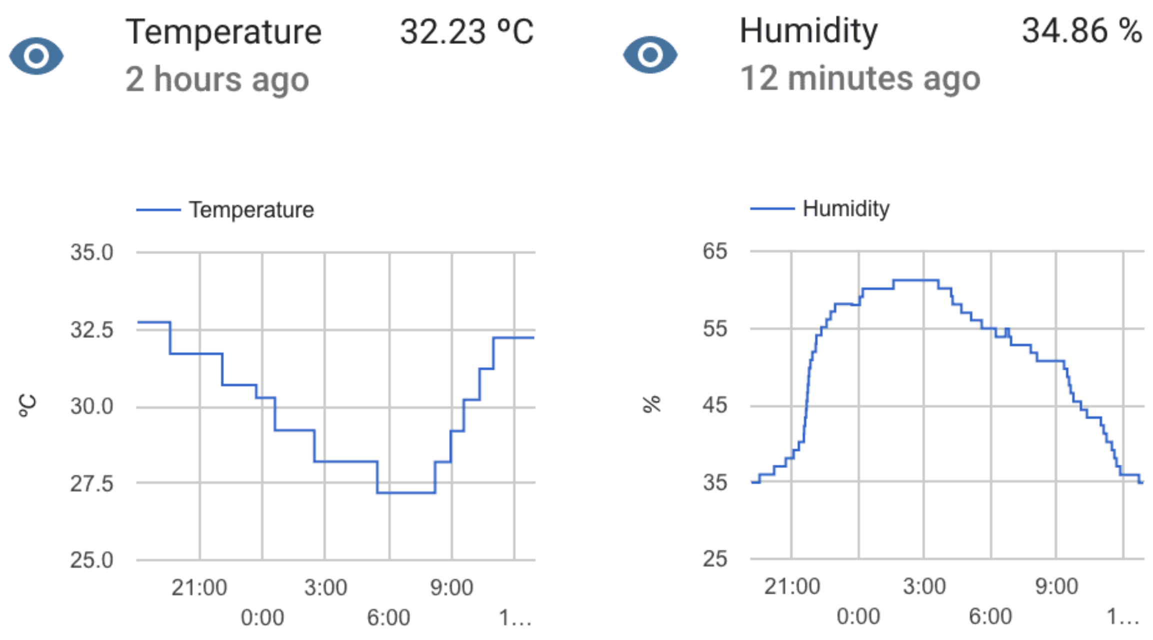


Figure 3: Home Assistant Generated Graphs [8]

The graphs shown above are entirely editable through the configuration files, the data types can be specified, the size of the graph and axis' can be modified. Home Assistant being Open Source provides a very flexible platform which can be added to or modified to fit your needs, allowing developers to build their own modules. The modules built are executed by the Hass.io operating system and can consist of conditional automations such as turning lights on when a user enters their home, or toggling heat or AC depending on the user’s presence in their home. There is an endless amount of automations which can be performed depending on the devices you have configured with Home Assistant; therefore, the platform is very popular. Home Assistant also has a very active community of developers who are active on the Home Assistant website forums. The developers are beneficial as they provide advice and solve common issues encountered when using Home Assistant which makes it a valuable resource when debugging. The large community of developers also means that Home Assistant is very frequently updated and new features are released almost daily with the latest technology quickly adopted. However, this rate of release and development comes at a cost of stability. As a result of stability issues, the application and plugins frequently have bugs or are broken, and documentation often doesn’t reflect the latest version. There is also a wide variety of tutorials and YouTube videos explaining the concepts of the Home Assistant Platform as well as tutorials on creating automations, connecting devices, and configuring the user interface. For each of these reasons Home Assistant was chosen over other options such as OpenHAB, or creating our own Home Automation software from scratch.

4.2.3 – Solution Using Home Assistant

Home Assistant was configured for the purpose of connecting the devices in the Tiny House and accessing them through one central hub. Home Assistant was flashed to an SD card, which was then put in the Raspberry Pi. Home Assistant is booted from this SD card, which automatically installs the latest version of the Home Assistant Operating System; Hass.io. Once Hass.io was running on the Raspberry Pi, and the Raspberry Pi was connected to the router via ethernet or WIFI, the Home Assistant Dashboard could be viewed from hassio.local:8123 [5]. The idea was to then bring this Raspberry Pi to the Tiny House and connect each of the smart devices to it, however the House became unavailable due to power issues. Home Assistant was then being used to mock the Tiny House as a prototype in the Lab; test circuits and devices were created to be configured with Home Assistant. We then began the development process using Home Assistant, which is done mostly using the configuration files mentioned above. Using these configuration files, we setup the MQTT publish-subscribe protocol mentioned in section 4.4. The Broker and the Server for this protocol both run on the Raspberry Pi, which needed to be configured to listen on port 8123 for clients who wish to publish data to the Raspberry Pi. With the Server and Broker initialized we could then connect our development circuits to this and begin transmitting data as if it were in the house. The Home Assistant Dashboard was configured to display this data and maintain a log of the data received. This is where we encountered the problem with Home Assistant, covered in more detail in section 4.2.4, where the core dependency for the MQTT protocol (Mosquito Broker) was updated. This update caused our communication with the development devices to be blocked and is still pending a patch to fix this issue.

4.2.4 – Problems Encountered Using Home Assistant

One of the problems encountered while developing using Home Assistant was related to the point mentioned in section 4.2.2 about the Stability of the platform. The Home Assistant software is very frequently updated and patched, thus some parts of it tend to become volatile. During our development one of our core dependencies, Mosquito Broker, for the MQTT communication, was updated and caused out application to break. The Mosquito Broker was depended on for the intercommunication between the NodeMCU boards which are both discussed below. This dependency broke towards the wrapping up phase of our project after having finished most of our development. Therefore, a portion of our final project was broken for the demo which was the communication between the NodeMCU boards and the Raspberry Pi. This communication is necessary to receive any information from the devices such as temperature, humidity, power levels, etc. This communication is also necessary for the automation scripts to execute successfully, as the devices cannot receive commands from the Raspberry Pi without this connection being established. Another problem which was encountered was the initial phase of learning how to use the Home Assistant software and understanding the project as it is very large. Beginning to work on any large Open Source project takes a lot of work as you need to research several different foreign topics, read segments of the code base, and read through the forums to get an initial understanding. After this process there was a realization that Home Assistant is mostly configuration rather than development. Once your devices are all connected to the Raspberry Pi, and your dashboard layout has been configured there is some development of the automation scripts.

4.2.5 – Alternatives to Home Assistant

There are several alternative all-in-one software choices for Home Assistant, and another alternative to using the Open Source software could be to develop the software from scratch. There are alternative Open Source platforms such as OpenHAB. OpenHAB is developed using Java and was released in 2017 with the idea of allowing people who are less technical to create an automated home. The drawback of OpenHAB is that the user interface is not as intuitive and easy to use as Home Assistant and not as easily customizable. Besides this drawback the two platforms are very comparable. Home Assistant is much older and thus has more documentation and a larger community however, OpenHAB has a very interactive community of developers as well. Each of the two products would be sufficient for the requirements of the project, however Home Assistant emphasizes the front-end to make it intuitive therefore, for our purposes was more desirable. The alternative to choosing a pre-built home automation software was to build one from scratch, this way it would be custom tailored to meet the needs of the project. This option was not taken as the Home Assistant software tends to become very complex and the Open Source projects which are pre-built allow for vast amounts of customization, which allowed us to tailor the project to our needs.

4.3 – NodeMCU

4.3.1 – About the NodeMCU

The most important contribution we have made to this project comes in the form of a small board called the NodeMCU as the NodeMCU allows for us to read data from non-IoT devices. The NodeMCU ties in directly with our goal of being able to read in data from the various devices throughout the house as most of the devices are non-IoT and have no built-in way for us to access the data over a network. The NodeMCU is made up of two parts: the ESP8266 WiFi chip and the ESP-12 core. When combined, the ESP8266 WiFi chip — henceforth referred to as the ESP8266 — and the ESP-12 core make up the NodeMCU board. In Figure 4 below you can see the complete NodeMCU device. The chip labelled "WiFi" at the top of the figure is the ESP8266 module, and the entire device is referred to as the NodeMCU henceforth.

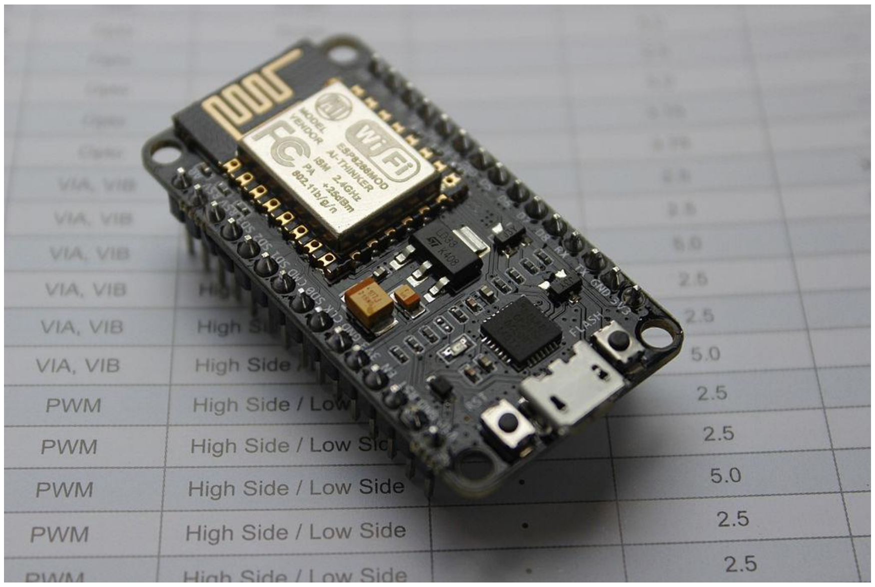


Figure 4: NodeMCU Board [9]

The ESP8266 is a WiFi module that can connect to a wireless network and allow for the transfer of data to and from the NodeMCU [10]. The ESP8266 is an Open Source project written in C and C++, and the project fully supports the programming of the ESP8266 using the Arduino IDE [11]. The ESP8266 can function as a standalone unit and the NodeMCU firmware and hardware increase the capabilities of the device.

The NodeMCU is an Open Source firmware developed in LUA and implemented in C [12]. The NodeMCU has been developed to support and run on all ESP modules, including the ESP8266 [13]. The NodeMCU simplifies the process of wiring devices to the ESP8266 as the NodeMCU board comes equipped with 16 general purpose I/O (GPIO) pins, four grounding pins, three 3.3V power pins, and a handful of other pins too [9]. The full mapping of the pins on the NodeMCU is available in Appendix B. When it comes to developing on the NodeMCU, we can create a sketch in the Arduino IDE and flash the sketch to the NodeMCU using a micro USB cable. The technical specifications and documentation for the NodeMCU and details regarding the ongoing development can be found on the NodeMCU GitHub repository.

4.3.2 – Rationale for Using the NodeMCU

The decision to use the NodeMCU was based on several factors that took the project's philosophies into account regarding cost and energy consumption. For starters, we wanted an inexpensive device that would allow us to read data from non-IoT devices. Since we had several devices that were non-IoT, we also took into consideration that we would likely need to purchase many NodeMCU units. In our research, we found that the NodeMCU to be the cheapest device — roughly $12, depending on the vendor — that still met our requirements. In addition to being low cost, the NodeMCU power consumption is approximately 16mA while idle and it peaks at approximately 70mA while actively transmitting data [14]. Given that the NodeMCU will not always been in a transmitting state, depending on the device the NodeMCU is connected to, these energy consumptions are acceptable.

The NodeMCU was also chosen for this project due to the low barrier of entry for getting started. As mentioned before, programming of the NodeMCU is supported by the Arduino IDE. The Arduino IDE is one that we were familiar with before this project, and thus we were able to save time by not having to learn as many new technologies. Equipped with only a laptop and a micro USB cable, we were able to program the NodeMCU from anywhere.

Lastly, the NodeMCU is physically quite small. The dimensions are 49 x 24.5 x 13 mm, which puts it at roughly the size of a small match box. While the size of the device was not a make or break requirement, we took it into consideration due to the nature of the project. Since we are working with a Tiny House that contains many IoT devices, each with their own NodeMCU, having a smaller sized board is smart decision.

4.3.3 – Solution Using the NodeMCU

Once we acquired the NodeMCU, we configured the Arduino IDE so that we could create sketches and read the outputs. The initial sketches were to test that the NodeMCU would be able to run continuously and persist through rebooting several times. We tested for this by loading sketch that cycled through turning on and off several LEDs, and we left the NodeMCU plugged in and running over night.

The next step was to set up a motion sensor circuit using the NodeMCU and a motion sensor in order to mock up the window sensors in the Tiny House. The idea behind the window sensors was to alert the occupants that their windows are left open as they are leaving the house. This could help the occupants not waste as much energy on cooling or heating by alerting them to not leave their windows open when they are leaving the house.

The specific motion sensor that we used was the HC-SR04; an ultrasonic range sensor that functions in the dark and has a range of 400 cm [15]. The HC-SR04 was chosen because it was low cost and simple to set up. In figure 5, shown below, you can see the circuit that we constructed.

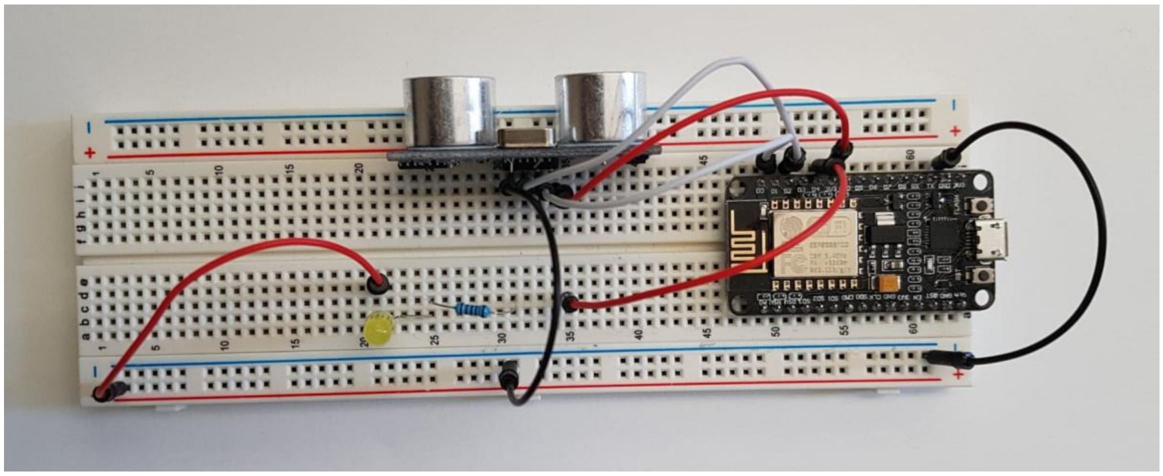


Figure 5: NodeMCU and HC-SR04 circuit

A signal is sent to the NodeMCU when an object passes in front of the HC-SR04 sensor, and the NodeMCU then toggles the LED to the on state. This code for this motion sensor circuit can be found in Appendix C. The purpose of the motion sensor circuit was to test if we could read in the data from a sensor to the NodeMCU, and then use that data to cause something else to happen in the house.

We also built a circuit that used a light dependent resistor (LDR) sensor and a NodeMCU. This circuit was created to mock up how a light sensor toggle on the interior, or exterior, lights of the Tiny House as it became darker outside. Unlike the previous circuit — the window sensor mock up — the light sensor serves as both a way to save energy on lights during the day, but also as a quality of life improvement for the occupants of the house.

Using the LDR sensor, we constructed a circuit similar to that of figure 5, and the new light sensor circuit can be seen below in figure 6. The code for the NodeMCU in the light sensor circuit is different than the motion sensor circuit, and the code for the light sensor circuit can be found in Appendix D. In the light sensor circuit, the LDR sensor outputs a different level of resistance based on the intensity of the light [16]. The sketch running on the NodeMCU then reads this outputted resistance and determines if the LED should be toggled on or off.

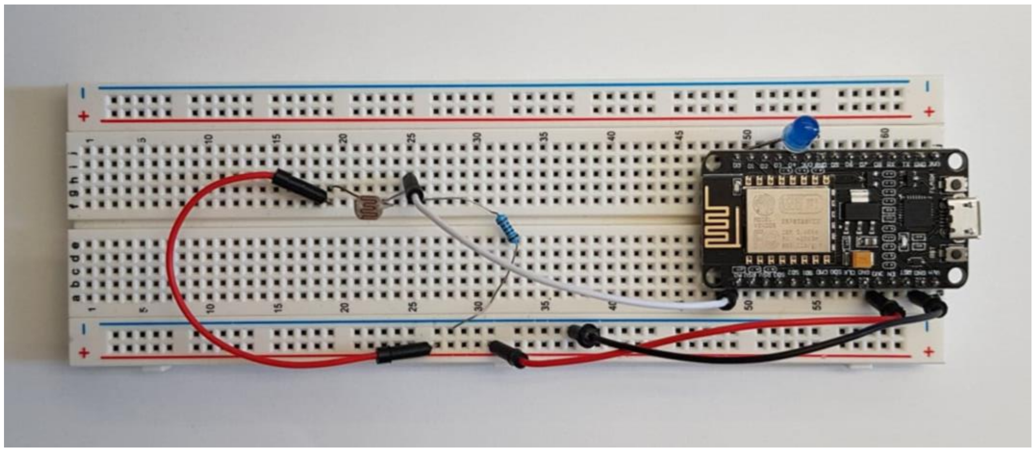


Figure 6: NodeMCU and LDR circuit

By building these circuits, we were able to demonstrate that the NodeMCU could be programmed to read in the data from a sensor and then perform some action through another device. This is important because it now provides us a way to control the devices in the house remotely through sensors. These circuits are only connected to the NodeMCU, and all the logic is contained on the board. The next step would be to connect the NodeMCU to Home Assistant over a network so that we could control the devices through Home Assistant too. In section 4.4

we go into detail regarding how we modified these circuits to control an LED using data sent over a network using MQTT.

4.3.4 – Problems Encountered Using the NodeMCU

One of the biggest issues we had with the NodeMCU board was the initial setup of the Arduino IDE. While there is full Arduino support for the NodeMCU, the setting up of the environment is not well outlined. The NodeMCU requires several libraries to be installed for the Arduino IDE to recognize the board. These libraries were not clearly defined, and we had to discover this through third party tutorials online.

Another issue we had was with the board architecture itself. In Appendix B you can find the pin mapping for the NodeMCU. One of the notable issues is the presence of only a single analog pin — depicted by a purple label. This single analog pin proved troublesome when we were testing out a wall sensor circuit. The wall sensors produce an analog output, and they can only be read in using the analog pin. As a result, if we wanted to use more than one wall sensor, then we would need a single NodeMCU for each sensor. The work around for this is to set up a multiplexer so that several analog signals could be fed into the same pin.

The biggest issue that we ran into during the testing of the NodeMCU was the lack of devices to test on. The devices inside of the Tiny House were not available to us for most of the year for reasons outside of our control. As a result, we had to build these mock circuits that were discussed above. While these circuits work and act a proof of concept for the real devices, we would have preferred to have tested our NodeMCU solution on the actual devices that were being used in the house.

4.3.5 – Alternatives to the NodeMCU

One of the alternatives to the NodeMCU that we considered initially was the standalone ESP8266 WiFi chip. The NodeMCU has a built in ESP8266, however the ESP8266 can be used as a standalone device. The ESP8266 is smaller and cheaper when compared to the NodeMCU. Additionally, the ESP8266 provides the same data collection and sending capabilities as the NodeMCU. This would be a big plus as we could purchase many ESP8266 chips and they would take up very little space in the Tiny House. However, the main drawback of the ESP8266 when compared to the NodeMCU is the lack of pins and the manual wiring required. The NodeMCU comes with all the pins prewired and all we have to do is wire our circuit into these pins. The ESP8266 would require us to manually wire these connections in order to provide the same functionality that the pins on the NodeMCU provide. This manual setup could introduce error that we, as Software Engineering students, may not be able to easily track down. As a result, we determined that the drawbacks of using the ESP8266 outweighed the slightly cost and size benefits.

The other device we considered using in place of the NodeMCU was the Arduino Uno. The Ardunio Uno is supported natively by the Arduino IDE, making it easier to setup and use out of the box. Additionally, the Arduino Uno has a higher number of pins and thorough documentation [17]. On the other hand, the Arduino Uno is more expensive, consumes more energy, and is physically larger than the NodeMCU [17]. The drawbacks of the Arduino Uno are a deal breaker as we chose the NodeMCU because it excelled in these exact areas.

4.4 – MQTT Communication Protocol

4.4.1 – About the MQTT Communication Protocol

Message Queuing Telemetry Transport (MQTT) is a publish-subscribe messaging protocol [18]. MQTT satisfies the goal by bridging the gap between the NodeMCU and Home Assistant, allowing for the data to be collected and displayed. MQTT provides the capability to transfer various types of data to other MQTT devices on the same topic [19]. MQTT is built upon a pattern that is commonly known as publish-subscribe. This pattern allows the devices to transmit data to many devices at a time as long as each device has the same topic. Topics are a type of message tag that are specified upon publishing or subscribing from the client or server side.

The broker is part of the publish-subscribe pattern, and it acts intermediary between the client (NodeMCU) and server (Raspberry Pi) [20]. To initiate the connection between the devices and the broker, the port and IP address of the broker is configured, and the client is set to connect to this configuration. Once configured, the devices will be able to locate the broker to initiate the process of transferring data. The devices that are strictly receiving data are connected to the broker as subscribers. The devices that are intended to strictly send data to other devices are connected as publishers [20]. The devices are invisible to the broker which means that the broker is not directly connected with any device.

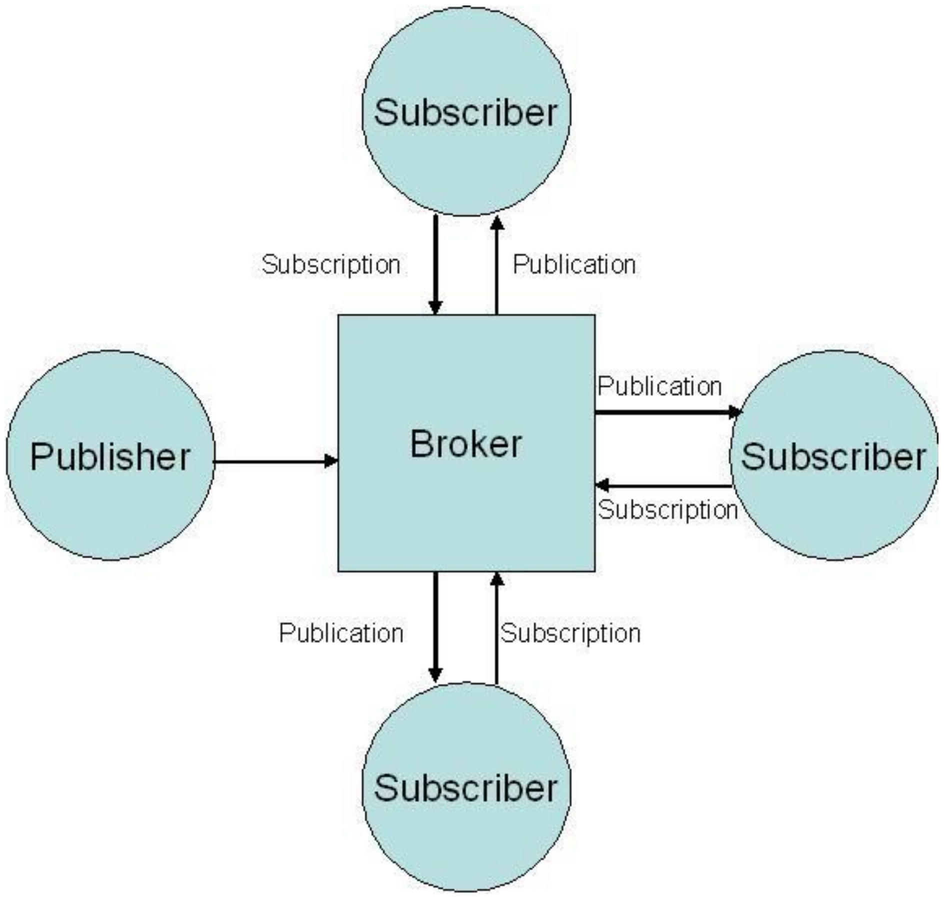


Figure 7: A simple publish-subscribe application [24].

The NodeMCU, in our case the client, is the device that is responsible for sending data, which is the publisher in this system. Whenever there is data that is required to be sent, the publisher creates a topic to ensure that the data is accessible from the client by the server. After a subscriber is connected, the server can listen to the broker through the MQTT channel by subscribing to the topic. Therefore, by setting up a MQTT publish-subscribe connection on our NodeMCUs and our Raspberry Pi, we can send data back and forth with Home Assistant over our network.

4.4.2 – Rationale for Using the MQTT Communication Protocol

Most of the devices in the house are non-IoT, and only some of the devices in the house are IoT. MQTT is a protocol for connecting the devices using WiFi, which provides means of transferring data to other devices. The machine-to- machine intercommunication is difficult to manage without the use of a protocol such as MQTT. Most of the devices are non-IoT based, therefore it is required that the devices are either physically connected to the Raspberry Pi pins or connected to the NodeMCU and sent to the Raspberry Pi using MQTT. Physically connecting the devices to the Raspberry Pi would be impractical as a large number of devices would need to be connected. Therefore, it would rather be preferable to send the data from the devices to the Home Assistant using MQTT.

4.4.3 – Solution Using the MQTT Communication Protocol

It was decided that MQTT would be used to allow for communication between devices and the Home Assistant through the Raspberry Pi. MQTT based communication allows the devices to send data through WIFI to Home Assistant [21]. We used the NodeMCU as it has a built in WiFi module that is required for sending and receiving data. The MQTT setup requires changes to be made in the Home Assistant configuration files. Home Assistant is the client that will receive all the data readings from the NodeMCU devices by subscribing to the topic.

In our system, the RaspberryPi is used as an MQTT broker that acts as a server for all the communication between the devices and the Home Assistant. The NodeMCU acts as a publisher in this architecture and it uses the WiFi to connect to the MQTT communication protocol. The devices are connected to the NodeMCU that gets the measured reading by the devices and publish those readings to the MQTT broker which is the RaspberryPi. The Raspberry Pi running Home Assistant, acts as the server which the NodeMCU is publishing data to, via the use of topics [22]. The data is then used by the Home Assistant for further observations. The architecture is explained more on figure 8 below.

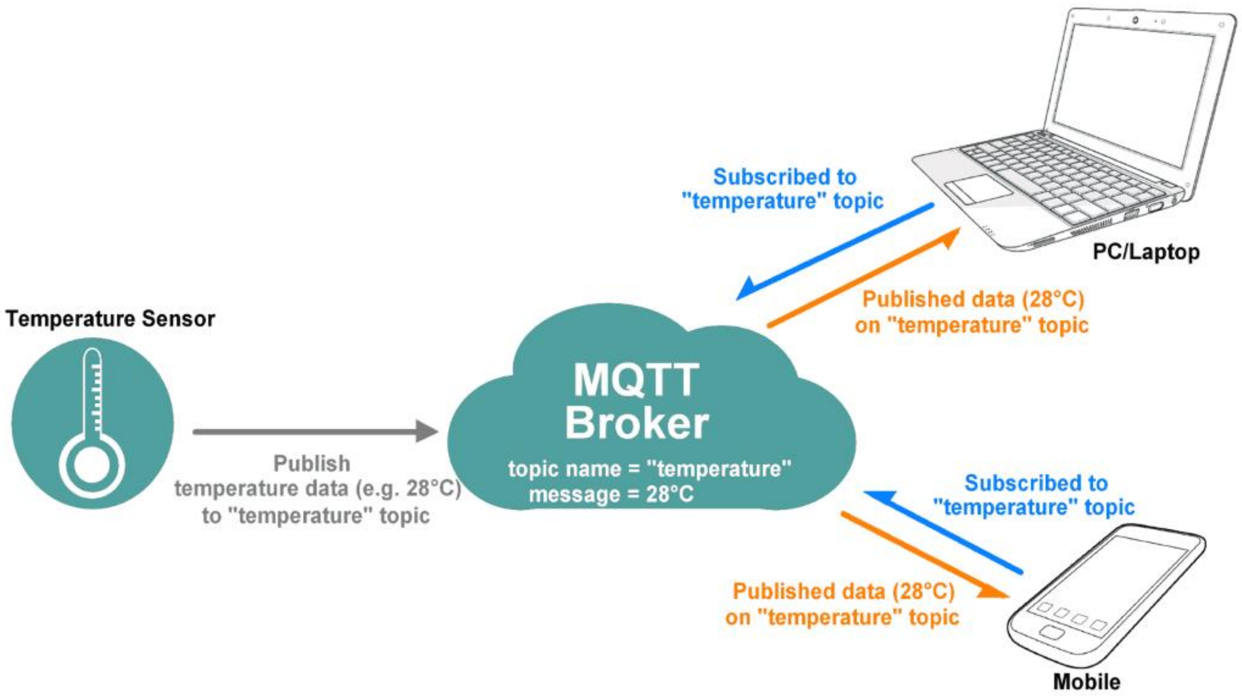

Subscribed to "temperature" topic
PC/Laptop
Published data (28°C) on "temperature" topic
Temperature Sensor
MQTT Broker
topic name = "temperature"
message = 28°C
Publish temperature data (e.g. 28°C) to "temperature" topic
Subscribed to "temperature" topic
Published data (28°C) on "temperature" topic
Mobile

Figure 8: MQTT publish-subscribe using Topics [25].

4.4.4 – Problems Encountered Using the MQTT Communication Protocol

Initializing the publish-subscribe pattern along with Home Assistant on the Raspberry Pi and NodeMCU can be a complicated process as it contains a lot of moving parts. Since MQTT is an Open Source technology, it goes under extensive updates regularly for the purposes of simplifying the configuration [23]. These frequent updates make certain plugins such as Mosquitto Broker become unreliable as they are unable to keep up with frequent changes. However, If the architecture of the MQTT protocol is not accurately set up, an update for MQTT can result in breaking of dependencies of the architecture. These breakages can result in some issues that are hard to solve such as WiFi disconnects, broker unavailability, and frequent restarts.

4.4.5 – Alternatives to the MQTT Communication Protocol

There are multiple other protocols that are Open Source and provide suitability for the similar environments used for tiny house based on asynchronous communication and can be used instead of MQTT. One example is Constrained Application Protocol (CoAP) for devices with limited resources. CoAP uses HTTPs to achieve data transfer. The packets used by CoAP are smaller in size compared to TCP packets and easier to generate and parse. The implementation of the CoAP is similar to MQTT where there is a client-server architecture where the clients request data from the server using a GET request. However, being a one-to-one device protocol, transferring state information between multiple devices will be difficult to achieve [26].

Additionally, there is Advanced Message Queuing Protocol (AMQP) which is another protocol that can be used to accommodate IoT communication. AMQP was created to be a reliable and interoperable communication protocol. AMQP gives the ability to store the transported messages while receiving and sending in the queues. AMQP uses the similar modularity of publishing and subscribing to topics in order to transport the data. AMQP is very flexible when it comes to routing and securing the transportation of the data. However, being recently developed, it goes under extensive updates, this provides backward compatibility but may break dependencies that version was using when it updates [27].

Chapter 5 – Moving Forward

5.1 – Recommendations

At the end of this project, we accomplished three goals that we had set out to. The first goal was to install Home Assistant on the Raspberry Pi, and to configure the user interface. The second goal was to use the find a way to send and receive data to non-IoT, which we achieved by using the NodeMCU. Out third and final goal was to figure out a way to get the data from the devices and display it on Home Assistant. We accomplished this using the MQTT communication protocol to turn our NodeMCUs into clients who publish their data to the Raspberry Pi, which is acting as the server. However, there were some goals that we were not able to accomplish during our work on the project.

Moving forward, we have some recommendations for the future of this project. Our first recommendation would be to purchase devices that are natively IoT. The biggest complication we ran into during this project was converting non-IoT devices into devices that we could read data from. For a smart house to work properly, we would strongly suggest buying IoT devices because they have better documentation, more support, and are easier to use out of the box. In a similar vein, all devices that require an internet connection, or proprietary software, should be stripped from the house. These devices are of no use as the project is meant to function without an internet connection.

The next big step would be to start collecting and storing mass amounts of data. We were unable to work on the Tiny House for most of year, and as a result we have no data from the house itself. Given that this is a research project on the practicality of a net zero house, it is important to have data that backs up this research. Building on top of this, the next team should focus on creating profiles for the occupants of the house based on the collected data. One of the long-term goals of this project is the ability to learn the behaviours of the occupants and to suggest different ways to reduce their energy consumption and improve their quality of life.

Finally, we strongly suggest that future groups have proper documentation for the work they do. This project has a lot of moving parts as it mixes software and hardware, and this means that it is difficult for a single person to know everything about the house. Proper documentation ensures that future teams will be able to build upon this project without having to figure everything out themselves, thus lowering the barrier for entry to this project.

Please feel free to fork the GitHub repository that we created for this project and continue documenting the project as it grows. The GitHub repository can be found at https://github.com/joshcampitelli/NorthernNomadHouse

References

[1] “Internet of Things,” Internet of Things Definition, 16-Jan-2015. [Online]. Available: https://techterms.com/definition/internet_of_things. [Accessed: 05-Mar-2019]

[2] “SMART,” SMART (Self-Monitoring Analysis And Reporting Technology) Definition. [Online]. Available: https://techterms.com/definition/smart. [Accessed: 10-Mar-2019].

[3] "Home Assistant," [Online]. Available: https://www.home-assistant.io/. [Accessed 10 March 2019].

[4] "Raspberry Pi | What is a Raspberry Pi?," [Online]. Available: https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/. [Accessed 5 March 2019].

[5] "Install Home Assistant," [Online]. Available: https://www.home-assistant.io/getting-started/. [Accessed 10 March 2019].

[6] "YAML Syntax | Grav Documentation," [Online]. Available: https://learn.getgrav.org/15/advanced/yaml. [Accessed 10 March 2019].

[7] home-assistant. (2018). https://github.com/home-assistant/home-assistant: home-assistant.

[8] P. Schoutsen, "Report the temperature with ESP8266 to MQTT | Home Assistant," 11 October 2015. [Online]. Available: https://www.home-assistant.io/blog/2015/10/11/measure-temperature-with-esp8266-and-report-to-mqtt/. [Accessed 20 March 2019].

[9] nodemcu-devkit. (2014). https://github.com/nodemcu/nodemcu-devkit-v1.0.

[10] I. Grokhotkov, "ESP8266WiFi library - ESP8266 Arduino Core 2.5.0 documentation," 2017. [Online]. Available: https://arduino-esp8266.readthedocs.io/en/2.5.0/esp8266wifi/readme.html. [Accessed 2nd March 2019].

[11] Arduino. (2014). https://github.com/esp8266/Arduino: esp8266.

[12] "Overview - NodeMCU Documentation," [Online]. Available: https://nodemcu.readthedocs.io/en/master/. [Accessed 2 March 2019].

[13] Nodemcu, “nodemcu/nodemcu-firmware,” GitHub. [Online]. Available: https://github.com/nodemcu/nodemcu-firmware/blob/master/docs/index.md. [Accessed: 10-Apr-2019].

[14] T. V. Eicken, "Computing stuff tied to the physical world | JeeLabs," [Online]. Available: https://jeelabs.org/book/1526f/. [Accessed 3 March 2019].

[15] "HC-SR04," [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf. [Accessed 7 March 2019].

[16] "LDR - Light Dependent Resistor 12mm | ProtoSupplies," [Online]. Available: https://protosupplies.com/product/ldr-light-dependent-resistor-12mm/. [Accessed 12 March 2019].

[17] “NodeMcu and Arduino IDE which is based on the ESP-12 module,” IoT. [Online]. Available: https://devzone.iot-ignite.com/knowledge-base/nodemcu-and-arduino-ide/. [Accessed: 10-Mar-2019].

[18] "FAQ - Frequently Asked Questions | MQTT," [Online]. Available: https://mqtt.org/faq. [Accessed 20 March 2019].

[19] S. Cope, "How MQTT Works -Client Connections | Steves Internet Guide," 13 March 2019. [Online]. Available: http://www.steves-internet-guide.com/mqtt-works/. [Accessed 26 April 2019].

[20] S. Cope, "MQTT Publish and Subscribe Beginners Guide," 10 October 2018. [Online]. Available: http://www.steves-internet-guide.com/mqtt-publish-subscribe/. [Accessed 20 March 2019].

[21] "MQTT Brokers | Home Assistant," [Online]. Available: https://www.home-assistant.io/docs/mqtt/broker/. [Accessed 26th March 2019].

[22] J. Blom, "Exploring the Protocols of IoT | sparkfun," 5 January 2015. [Online]. Available: https://www.sparkfun.com/news/1705. [Accessed 5 March 2019].

[23] Nodemcu, “MQTT re-connect issue · Issue #2197 · nodemcu/nodemcu-firmware,” GitHub. [Online]. Available: https://github.com/nodemcu/nodemcu-firmware/issues/2197. [Accessed: 15-Mar-2019].

[24] "Publish and subscribe messaging | IBM Knowledge Center," 21 October 2008. [Online]. Available:https://www.ibm.com/support/knowledgecenter/SSVHEW_6.2.0/com.ibm.rcp.tools.doc.appdev/simplepubsub.jpg. [Accessed 5 March 2019].

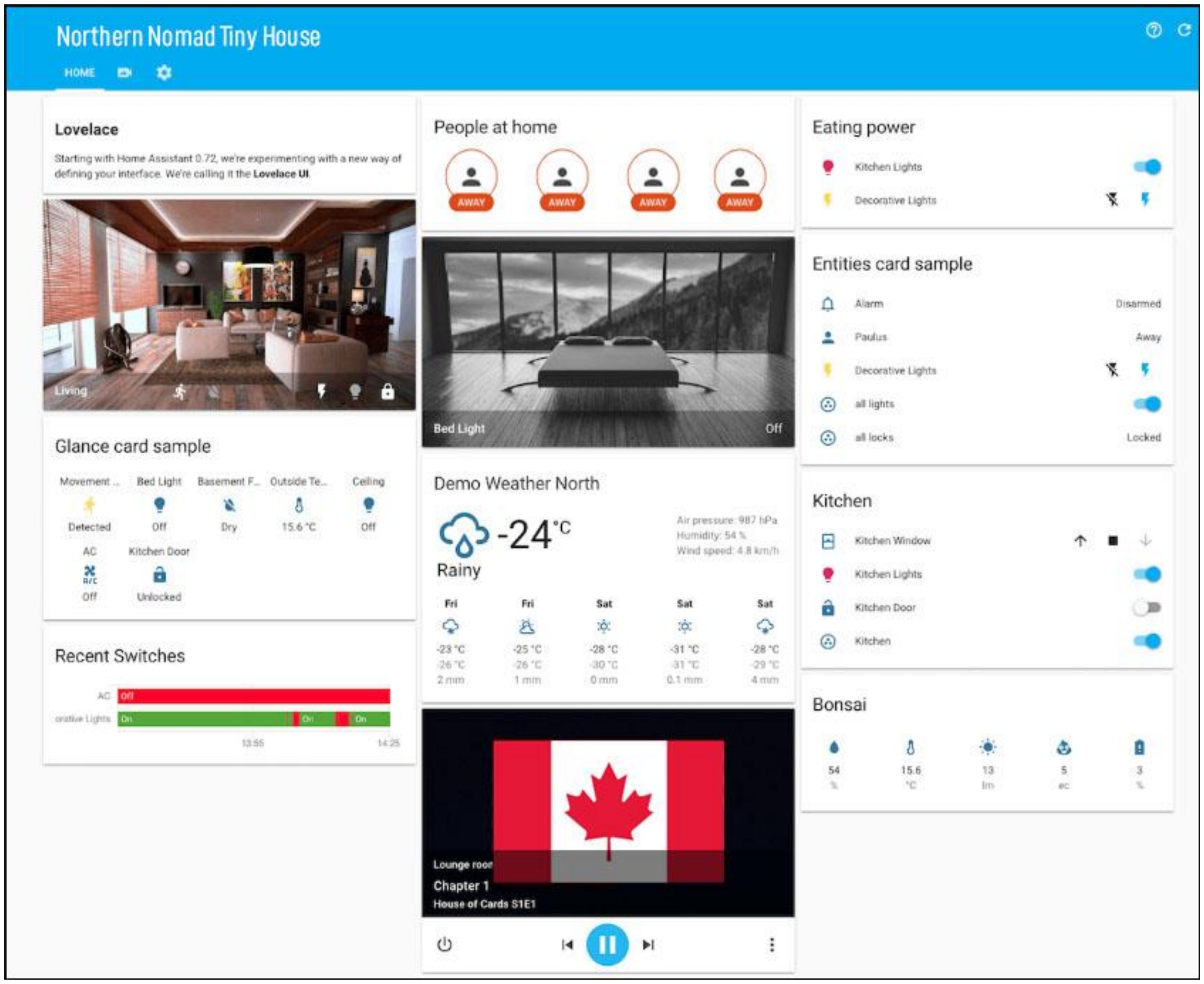
[25] "NodeMCU MQTT Client with Arduino IDE | ElectronicWings," [Online]. Available: https://www.electronicwings.com/nodemcu/nodemcu-mqtt-client-with-arduino-ide. [Accessed 9 March 2019].

[26] T. Jaffey, "MQTT and CoAP, IoT Protocols | The Eclipse Foundation," 2014. [Online]. Available: https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php. [Accessed 29 March 2019].

[27] A. Piper, "Choosing Your Messaging Protocol: AMQP, MQTT, or STOMP | VMware vFabric Blog," 19 February 2013. [Online]. Available: https://blogs.vmware.com/vfabric/2013/02/choosing-your-messaging-protocol-amqp-mqtt-or-stomp.html. [Accessed 29 March 2019].

Appendix

Appendix A



Lovelace

Starting with Home Assistant 0.72, we're experimenting with a new way of defining your interface. We're calling it the **Lovelace UI**.

Glance card sample

Movement ...	Bed Light	Basement F...	Outside Te...	Ceiling
Detected	Off	Dry	15.6 °C	Off
AC	Kitchen Door			
Off	Unlocked			

Recent Switches

People at home

Demo Weather North

-24 °C
Rainy

Air pressure: 987 hPa
Humidity: 54 %
Wind speed: 4.8 km/h

Fri	Fri	Sat	Sat	Sat
-23 °C	-25 °C	-28 °C	-31 °C	-28 °C
-26 °C	-26 °C	-30 °C	-31 °C	-29 °C
2 mm	1 mm	0 mm	0.1 mm	4 mm

Eating power

Entities card sample

Alarm	Disarmed
Paulus	Away
Decorative Lights	
all lights	
all locks	Locked

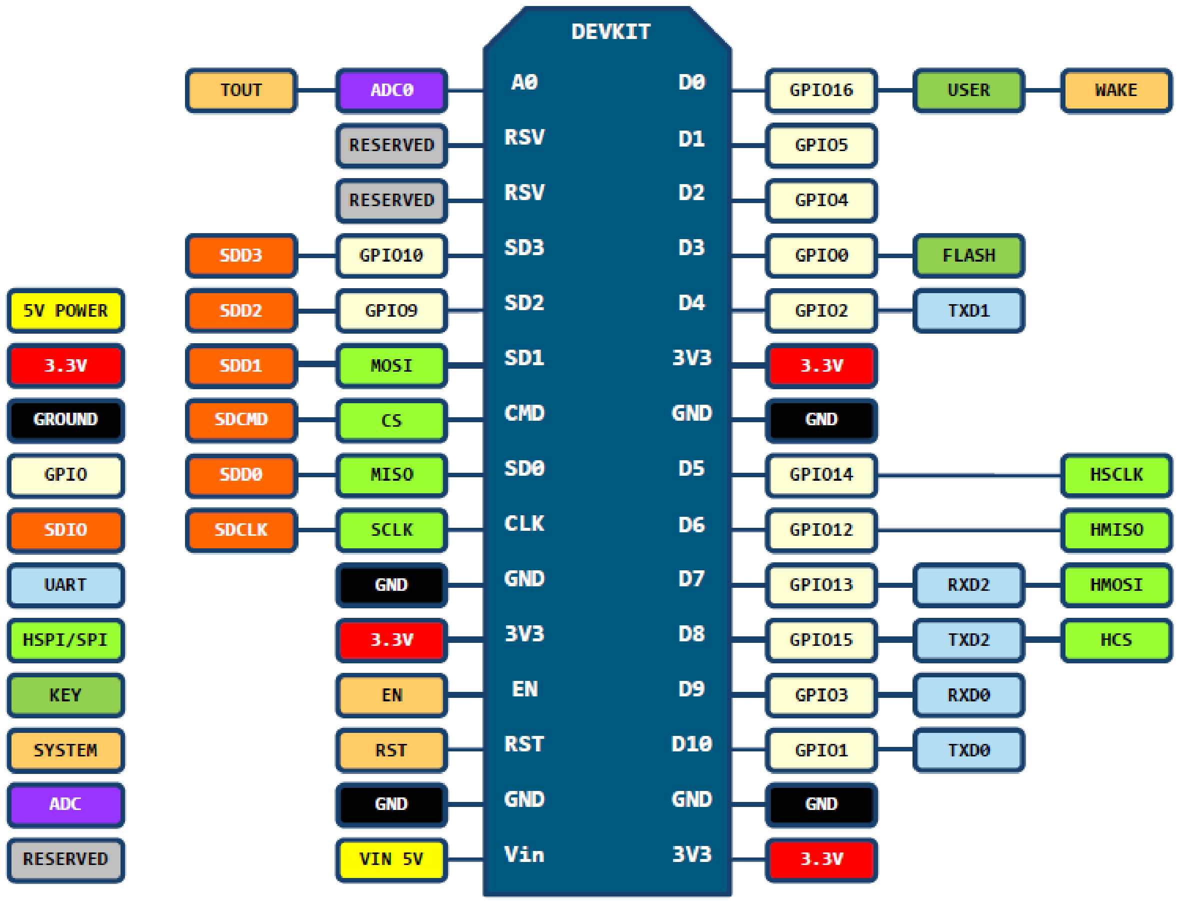
Kitchen

Bonsai

54	15.6	13	5	3
%	°C	lm	ec	%

Appendix B

PIN DEFINITION

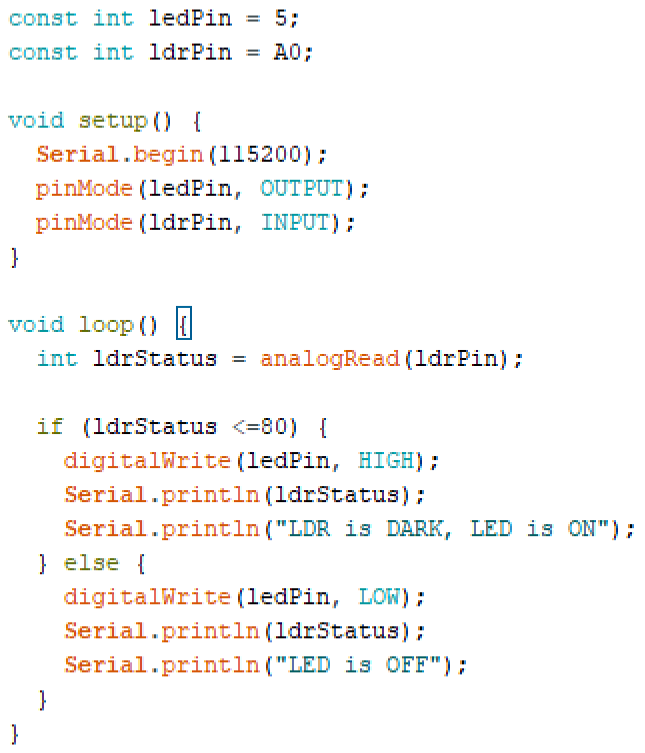


D0(GPIO16) can only be used as gpio read/write, no interrupt supported, no pwm/i2c/ow supported.

Appendix C

```
const int trigPin = 5;
const int echoPin = 4;
long duration;
int distance;

void setup() {
 pinMode(trigPin, OUTPUT);
 pinMode(echoPin, INPUT);
 pinMode(0, OUTPUT);
 Serial.begin(115200);
}

void loop() {
 digitalWrite(trigPin, LOW);
 delayMicroseconds(2);
 digitalWrite(trigPin, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPin, LOW);

 //Read in data from the sensor
 duration = pulseIn(echoPin, HIGH);

 //Distance is CM
 distance = duration * 0.034 / 2;

 //Display the distance in the output terminal
 Serial.println(distance);

 //Checking if the sensor detected motion within a certain distance
 if (distance < 35) {
    digitalWrite(0, HIGH);
    delay(5000);
  } else {
    digitalWrite(0, LOW);
    delay(500);
  }
}
```

Appendix D



```
const int ledPin = 5;
const int ldrPin = A0;

void setup() {
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);
  pinMode(ldrPin, INPUT);
}

void loop() {
  int ldrStatus = analogRead(ldrPin);

  if (ldrStatus <=80) {
    digitalWrite(ledPin, HIGH);
    Serial.println(ldrStatus);
    Serial.println("LDR is DARK, LED is ON");
  } else {
    digitalWrite(ledPin, LOW);
    Serial.println(ldrStatus);
    Serial.println("LED is OFF");
  }
}
```