Home Security and Automation System

Final-Year Project

System Design Document

Authors – [SAAD SAEED]

Project supervisor– [PROFESSOR JAMES SMITH]

Plan date: [August 8th, 2018]

Course: [EECS 4080]

Revision: [3]

REVISION HISTORY

Revision Number	Date	Comment
1.0	July 19th, 2018	Rough draft of the report started
1.5	July 23rd, 2018	Rough draft completed
2.0	July 29th, 2018	Individual report started based on the feedback from Professor Smith
3.0	August 8th, 2018	Implemented the changes based on Professor's comments and had Vivian Octave proofread the report for grammatical issues.

Table of Contents

DOCUMENT OVERVIEW

This document has been developed by Saad Saeed for Home Security and Automation System for EECS 4080. This project explores the possibility of building an inexpensive and effective basic home security system that only collects information essential to its functionality. The project was completed in a group of three people; Andres Hernandez,Tapan Desai and myself. , This document strictly focuses on the contributions made by myself in an individual and group-based setting.

BACKGROUND

Statistically speaking, Toronto is one of the safest cities in Canada, although it is perceived by its citizens to be unsafe [1]. As true as that may be, Toronto has had a moderately-high number of break-ins in the recent years. In fact, it was reported that the amount of break-ins reported between 2008-2013 was 8400 [2]. According to this information, one can draw the conclusion that home security is a concern in Toronto and an issue which needs to be addressed. Companies such as ADT and Honeywell have attempted to solve this problem by selling home security systems that offer intrusion and fire monitoring [3]. These systems are basically a network on IoT devices; sensors that collect information from the real world, such as the temperature of a stove in the kitchen, the information is then sent to a central device with more computing power for analysis. Based on this analysis, actions are taken to eliminate potentials threat (such as sounding an alarm, turning off the stove, etc). Therefore, security systems can be thought of as a complex set of interconnected IoT devices that aim to make people's lives safer.

As good as that sounds, the IoT industry has suffered from a set of issues since its infancy; most notably issues pertaining to the collection and dissemination of sensitive information. These issues stem from the fact that companies do not define clear-cut policies that state the type of data being collected by the product/service in question. Furthermore, these policies usually also do not explain-in a clear and concise manner-how the collected data will be used and how long will the manufacturers have access to it [4]. Additionally, most IoT device manufacturers fail to implement basic security safeguards (such as encryption) during the design phase of the device [5]. This can have devastating consequences for both the manufacturer and consumer. For example, if a criminal was able to exploit a security flaw within the system (such as the lack of encryption), they could simply eavesdrop in on the data being relayed by the sensor and analyze it to determine the best time to break into the house [4]. Such attacks are very easy to conduct in a smart home environment because of bad development practices and security flaws within the inherent framework itself (Samsung SmartThings platform, for example) [5].

It can be difficult for consumers to determine what information is collected by the device itself and whether the amount of collected information could be reduced without affecting its functionality [4]. IoT device manufacturers make consumers accept their Privacy Policies before allowing access to the device itself. These legal documents contain information about the type of data collected and the storage policies associated with it. Unfortunately, these documents are extremely long, and as a result, not a lot of people read what is written in them [6]. Additionally, the complexity with which these documents are written in do not really help this cause. A study conducted by Good et al [6] tested to see whether End User License Agreements (EULA) had any influence on users' decision to install a program on their computer. They presented test subjects with five programs bundled with spyware (not know to the user at the start of the study) and gave them the option to install them on the lab computers. Three forms of EULAs were experimented with: regular, a popup warning and a regular EULA, and a short notice (a document that only states the essential information from an EULA, such as the type of data collected from the program, etc). They found that participants were more informed about a program's intention when they were presented with the short-notice documents. Furthermore, majority of participants used these short-notice documents to compare similar product before installing the programs. A few participants actually read parts of the original EULAs carefully on one occasion, but eventually gave up on reading them due to lack of brevity [6]. Participants most notably wanted the original EULAs "shorter, easier to read and in very accessible language." [6]. This study also found that participants expressed less regret about their decision to install (or not) a program after reading the short-notice document associate with that program. As one can see, this study echoes the general concern that users do not really read the legal documents that are long and complex in nature. Also, users are more informed about a product's intention when the terms and conditions are provided in format that is short and simple to read (such as the short-notice documents). Unfortunately, this is an issue that can have massive repercussions and should be addressed by the manufacturers as quickly as possible. Businesses should try and adopt the findings of this study to come up with agreements that highlight the important clauses so that customers are not left hanging in the dark.

The issues mentioned above were the main focus in this EECS 4080 project. The overall goal of this project is to provide a solution to these problems by implementing a security system through a Privacy-by-Design approach. As the name suggests, protection of private information will be the most important success criterion. Success criteria assigned to this project include providing a system that is robust, relatively well-guarded, and only collects the information that is necessary to its function.

SYSTEM OVERVIEW

DESCRIPTION

The system as a whole is comprised of three parts, each responsible for monitoring one particular part of the security system: **(1) Home Environmental Control and Sensing, (2) Home Security Action, and (3) Home User Authentication and Privacy**.

This report strictly focuses on (2): Home Security Action. This part of the project is responsible for permitting actions to be taken in response to (a) fire risk in the kitchen, (b) entry risk at the door. More specifically, fire risk at the stove/oven is to be monitored and power to the unit is to be controlled at the breaker level; door lock actuation is to be controlled in response to authentication verification conducted by the Home User Authentication and Privacy part of the project.

Each group member was responsible for independently implementing one of the three components mentioned above. Additionally, each member was provided with access to hardware necessary for the completion of the particular individual component. Once the three subcomponents were fully implemented, it was time to connect them together so that each component could communicate with the other two, forming the security system as a whole. In order to do so, time went into forming a network that focuses on a master-slave configuration. Most security systems utilize this configuration, where a central control unit is the master device and the peripherals are slave devices. The master device is responsible for collecting and analyzing data that is reported by the slave devices. It is also responsible for making decisions based on the results of the analysis.

Details on the network infrastructure will be provided in the subsequent sections of this report.

SYSTEM ARCHITECTURE

The main reason for undertaking this project was to make an intellectual contribution towards privacy of information. Certain guidelines were developed with a focus on privacy and were used to guide the design decisions of the security system as a whole. More specifically, the focus was on the following points regarding the system architecture:

- Eliminating the use of the cloud and strictly using a master device (isolated from the internet) for decision-making and computation;
- using a wired network instead of wireless for communication so that the burglars have a hard time intercepting information or stopping communications altogether via interference;
- only collecting the absolute minimum information to make features work;
- using encryption while storing and transmitting data for the purposes of thwarting man-in-the-middle attacks

According to the Chaudhri and Cavoukian paper on the 3P framework on IoT devices [7], key data components associated with every IoT network can be broken down into five parts. These parts are the most vulnerable to security breaches and measures need to be in place to protect the users' information [7]. The four bullet points mentioned above address these five vulnerability points in the best way possible given the time and resource constraints associated with this project.

Software Architecture

In terms of a software architecture for this project, most of the software was written for the purposes of getting the sensors to communicate with the microprocessors. This meant writing code in a language that the Teensy boards could understand: C++ using Arduino IDE 1.8.5 and Teensyduino [8][9]. Furthermore, external open-source libraries had to be imported before the peripherals could report data back to the microprocessors. This can be attributed to the fact that some clickboards are not supported by the Teensy with out-of-the-box libraries (more information is provided in the Software Design section of this report).

Furthermore, writing code that adhere to good coding practice and style was essential to the project, considering the fact that a security system should have no obvious flaws-or any piece of software for that matter. Some of the practices employed included eliminating the use of magic numbers, using proper access modifiers for permissions, and commenting parts that were essential to the security of the program.

As mentioned previously in the System Architecture section, the use of encryption was necessary for the purposes of securing information while in storage and transit. For this part, an Advanced Encryption Standard (AES) encryption library [10] was imported into Arduino IDE. If there was a need to exchange data between two Teensy boards, the sender Teensy would utilize the aesEncrypt method from the library before sending the information, and the receiving Teensy would use the aesDecrypt method on the incoming data to decrypt the information. This ensured that the data was safe from man-in-the-middle attacks during transit. A slowdown in performance was noticed by 1-1.5 seconds due to the resources used during the encryption and decryption process. However, this slowdown is small price to pay for protecting private information.

Hardware Architectures

The underlying design principle associated with this project is as follows: make use of external sensors to collect information about the environment and make decisions based on this information regarding home security. All commercial security systems employ this-or a variant of this-technique. One needs to know that effort is not being put into trying to deviate from this practice, instead the focus is on trying to refine it by putting an emphasis on policies pertaining to information collection and user privacy. As mentioned in the System Architecture section of this report, the primary focus pertaining to hardware architecture is as follows:

- only collecting the absolute minimum information to make features work;
- using a wired network instead of wireless for communication so that the burglars have a hard time intercepting information or stopping communications altogether via interference;

In order to satisfy the first bullet point, the security system is being developed with the most basic hardware required to implement a certain feature. This ensures that the hardware will be utilized only for the tasks it was originally designed for and nothing else. The intent here is to eliminate the idea of a concept drift: utilizing a technology for purposes not initially known to its users [4]. Concept drift is a serious issue with new technologies because of a lack of awareness among users. Users should be made aware of all the potential uses of a particular technology so that they can make informed purchases and do not get surprised. Manufacturers should be transparent with the information they provide about their products. The IoT industry needs to get on board with this idea.

Furthermore, Personal Identifiable Information (PII) have been eliminated from the codebase for the purpose of anonymizing users identity. Additionally, hardware components have been assigned values strategically so that the devices within the security system are able to communicate with each other without any issue. If an adversary eavesdrops in on the network traffic, they will be unable to identify which device sent the information because these numbers would mean nothing to them. In order to distinguish between the communicating devices within the system itself, the RS485 hardware standard and the MODBUS software protocol have been utilized. More information on these technologies is provided further down the report. Finally, a fully-wired network has been used to prevent remote attacks. If an attack was to be carried out, the adversary would have to be on-site to do any harm to the system. This would force the criminal to think about the feasibility of launching an attack on-site and risk being seen by other people.

HARDWARE DESIGN

For the duration of the project, we were provided with different pieces of hardware, such as sensors, that we had to learn to connect to our microprocessors. Once this step was done, the sensors were polled by their respective microprocessors for information. Afterwards, actions would be taken based on the type of information collected. Some of these actions include turning off the power to an appliance and turning on a fan when the temperature increases significantly.

HARDWARE COMPONENTS

Note: component pictures and schematics provided in the Appendix section of this report

Computer Systems (hardware standards)

RS485: One thing that affected the outlook of the project was the choices we made when connecting the individual systems with each other. One of the key requirements for our network was that the system should be isolated from any outside sources, such as the internet. Furthermore, we wanted to establish a master-slave relationship between the devices, where all the slave devices send their data back to a central master device only after the master polls the slaves. This allows the master device to perform computation and make decisions based on the data it received back from the slave devices, such as sounding an alarm when the temperature sensor data shows a significant spike in its readings. We researched the most popular standards so that we could find one to integrate into our security system. We first looked into Ethernet as possible solution. However, the issue with Ethernet is that it is full-duplex in nature, meaning that devices can both send and receive data at the same time, and hence, it does not support the master-slave configuration that we were looking for [11]. Additionally, Ethernet does not support packet collision, meaning that data can be lost while in transit if two devices broadcast information over a network at the same time [11]. This would be very dangerous for a security system; if the central control unit does not receive critical information from its peripheral devices due to packet collision, it cannot determine if there is a threat on the premises. Standards such as WiFi, Bluetooth, and Zigbee were not chosen because they all operate over the airwaves (wireless) [12]. Utilizing such standards would mean that we would violate one of the key requirements:

- using a wired network instead of wireless for communication so that criminals have a hard time intercepting information or stopping communications altogether via interference;

We also looked into standards such as SPI as a possible solution. However, SPI supports communication over short distances (~ 3m) [13]. This was a problem because we were thinking about extensibility by trying to secure the whole house, which requires a lot more than a few meters of wiring.

Finally, we looked into the RS485 hardware standard as a solution. The RS485 is a wired hardware standard that only supports half-duplex communication [10]. However, this is perfect for the master-slave device relationship that we were looking for, where only one device can broadcast information throughout the whole network after being polled for information by the master device. Furthermore, since it is a wired hardware standard, we do not have to worry about violating the requirement mentioned above. RS485 also allows devices to send information over a long distance (1.2km) [10]. This would mean that we can have one security system give coverage to the whole house. Keeping these things in mind, it was then decided that we would use the RS485 standard to connect our individual deliverables together.

Project Specific hardware items (e.g. Sensors, Transducers, Mechatronics)

Listed below is the hardware used to implement the system as a whole:

Group-based

1) **Breadboard with power supply [14]** - To connect the external peripherals to the MCUs, we used breadboards with built-in power supply. The model of the breadboard is the PDS-500, which is manufactured by JDR Microdevices [14]. Colour-coated copper wires were used to connect the pins of the peripheral devices to the Teensy board. When a peripheral required more power than what the Teensy could supply, the power supply built into the breadboard was used to overcome this issue. The power supply provides three different voltage options: 5V constant, 10V varied, and 15V varied. For the varied voltage options, the user can use the dial on the breadboard to control the voltage being supplied by the power supply.
2) **RS485-Click 2 [15]** - This transceiver board is used for connection between the UART of an MCU and the RS485 communication bus. It offers half-duplex communication capability, low power consumption, and requires 3.3V or 5V to power on [15]. Its half-duplex communication capability allow for asynchronous communication with other devices on other breadboards. This characteristic allowed us to implement the master-slave relationship, where only one device can broadcast data throughout the network at any time.
3) **DTECH 6ft USB to RS485 RS422 Serial Converter Adapter Cable [16] -** This cable was used to connect a USB device (such as a laptop) to a Teensy board. Afterwards, MODBUS packets would be sent between devices for the purposes of debugging and extracting information.This cable was essential to our first deliverable where we were required to send data from our laptops to the Teensys and back.

Individual-based

1) **Teensy 3.1 [17] -** This is the microprocessor I was assigned by the project supervisor, Professor James Smith. All of us were provided with a variant of a Teensy board because they are Arduino-based. This means that they are relatively easy to set up. Furthermore, the Arduino developer community is great at helping people solve issues with their boards; there are great resources and guides available on the internet for first-time developers due to a high number of people using Arduino boards or its variant. This particular board is developed by PJRC and comes with the most common pinouts to allow for compatibility with most types of hardware [18]. Furthermore, these boards don't have the serial port latency issues that persist on the ATMEGA-based Arduino boards [19]. This means that information is cached by the MCU until enough bytes of data are collected or until the latency timer runs out. This can be an issue where the board does not get the bytes of data fast enough to respond to the serial device before it times out [20]. Teensy boards eliminate this issue and are 18 times faster than an Arduino board [19]. Had we chosen a truly Arduino MCU, we would have run into this serial port latency issue because we are only passing a few bytes of data at a time, and this would have led to improper communication (synchronization issues) between communicating devices.
2) **IrThermo click 3.3V [21]** - The temperature-sensing and stove shut-off deliverable was made possible by using the IrThermo click board. This board is sold by MikroElektronika and houses a Melexis MLX90614ESF-BAA temperature sensor chip [21]. The temperature sensor is capable of providing measurements to a high accuracy over a broad range of temperatures (-40 ^{O}C to +125 ^{O}C for the chip itself and -70^{O}C to +380^{O} C for surrounding objects) [22].
3) **Relay click [23] -** This click board is also sold by MikroElectronika and has two G6D-1A-ASI DC5 relay switches from Omron that control the flow of electricity to the connected devices [23]. It requires 5V of power to operate, something that the Teensy 3.1 cannot provide [18]. Hence, an external power supply was used to make it operational.
4) **UHPPOTE Fail-safe strike plate [24] -** The strike plate was used for the door actuation deliverable, where current was passed by the relay click board to open it. It requires 12V of power, something that the Teensy cannot deliver on its own [18]. Hence, an external power supply from the breadboard in combination with the Relay Click were used to overcome this barrier. Furthermore, this deliverable relies on the keypad verification deliverable which was implemented by a fellow group member.
5) **LED and resistor -** An LED and a resistor were used for the stove shut-off deliverable. This part of the project was hooked up to the temperature sensor and the relay board, where a significant increase in temperature would activate the relay board and turn on the LED. This setup was used to simulate a stove being shut off.

HARDWARE INTEGRATION

Logical Design

With our security system, we elected to use a star topology as opposed to a bus topology. With a star topology, every peripheral device connects to one central device [25]. This is the basic setup for a master-slave relationship that we were looking for, where a master broadcasts its desire to receive information from a peripheral device, and that peripheral device sends data only to that master device. We also looked at the bus topology as a possible solution. However, the one thing that caught our attention was that the star topology ensures that every connecting device shares a common transmission medium. This can be an issue where a break in the transmission medium can render the whole system unusable [25]. We were hoping to avoid this issue by looking for a configuration that would allow for a dedicated path of communication between the slaves and master so that it becomes easier to troubleshoot a device without affecting the rest of the network. Additionally, it is very unsafe for a home security system to use the bus topology because all an adversary has to do is break the common transmission wire to render the whole system unusable. With a star topology, a break in the transmission medium for one peripheral means that the other devices remain operational.

Physical Design

Since this project serves as a proof of concept, we were not too worried about the aesthetic appeal of the project. As a result, we built the whole network on three large breadboards using coloured copper wires. We consulted the pinout diagram of the peripherals and the particular Teensy boards to get an understanding of the pin configuration. Based on this information, we connected the pair of devices via copper wires. The same approach was utilized when connecting our individual mini-projects into one big system.

SOFTWARE DESIGN

As mentioned, we used Arduino IDE version 1.8.5 [8] to write the pieces of code (sketches) to get our MCUs to perform the necessary tasks related to our project. Furthermore, we used TeensyDuino 1.4.2 SDK [9] (add-on for the Arduino IDE) to compile the code specific to the Teensy boards. As previously mentioned, one of the main reasons for using the Teensy boards is due to the latency issues associated with the Atmega Arduino boards. According to [9], Teensys include built-in flow control so that we will not lose incoming data if our sketch does not read the incoming data quickly. Due to this feature, we did not have to worry about synchronizing our polling calls to the sender device nor did we have to worry about losing information as a result of the built-in flow control. This made the code easier to write and understand.

SOFTWARE PACKAGES

Note: The code written for my deliverables can be found in the following GitHub repository: https://github.com/SaeedS28/EECS-4080.

{Deliverable 1: Temperature sensing with relay switch activation}

This deals with the stove shut-off deliverable. The sketch crucially depends on the Adafruit-MLX90614-Library [26] because this repository allows the IrThermo click board to communicate with the Teensy board and report back temperature values. As mentioned previously, this particular temperature sensor is not supported by the Teensy board out-of-the-box, hence the need to import this library.

Getting the relay board to turn on and off-based on the temperature readings-required no special library since all the Teensy board had to do was control the voltage of the outgoing signal: digital high to turn the particular relay on and a digital low to turn the relay off. Such instructions require no external library to execute.

{Deliverable 2: Door Actuation}

In order to simulate door actuation, a strike plate was connected to the relay board. The relay activation code was reused from the previous deliverable. The program developed for this deliverable relies on the AES encryption library [10]-mentioned previously-to verify incoming information. By this, I mean that I added a segment of code that looks for a MODBUS packet sent from another member's Teensy board. His Teensy board verifies a user's identity by checking the code entered via a keypad. If the user enters the right code, his MCU encrypts the data (AES algorithm) and encapsulates it in a MODBUS packet. It then sends the packet from its UART to my Teensy board via the RS485 clickboard. Within my program, I continuously poll for packets of information. When my Teensy board receives a packet, it check for two things straight-off-the-bat: the device ID and function ID associated with the group member's Teensy board. If these attributes match up with the expected values (see Appendix for a list of values that we chose to incorporate into a MODBUS packet), it decrypts the encrypted data with a dynamic key and checks to see if the decrypted data matches a predefined array of numbers. If all of these things are verified, my Teensy board sends a digital high signal to turn on the relay, and consequently, activate the strike plate for five seconds. Once the time expires, the relay switch turns off and deactivates the strike plate.

DELIVERABLE SUMMARY

Presented below is a detailed summary of all the deliverables that we implemented. They can be categorized as either being group-based or individual based:

Group-based

For the very first deliverable, we were asked of two things: flash the LED of our respective Teensy board and find a way to send a MODBUS packet between two laptops (master-slave configuration). Completing the first part was relatively easy due to the sample code that came with the Arduino IDE. As for the second part, it was more involved; this part required us to set up both the hardware and software in order for the communication to take place. On the hardware side of things, we had access to two DTECH 6ft USB to RS485 RS422 Serial Converter Adapter cables. In order to connect them together, we consulted the connection diagram that came with the cables [27]. After setting everything up according to the diagram-a pair of connections between the RX and the TX slots-it was time to test the connections. For a thorough testing experience, we utilized two applications: CoolTerm and SimplyModbus [28] [29].. CoolTerm is a program similar to the Linux Terminal and allows connected machines to communicate with each other [30]. For testing purposes, we created a connection between two machines and used our keyboards to send each other messages. Since the wiring between the two cables was correct, typing character on one machine became visible on the other. This proof-of-concept gave us the green light to proceed to the next step; adding the MODBUS software protocol on top of the RS485 hardware standard. MODBUS is a software protocol that allowed us to encapsulate the data into packets for transportation. A MODBUS packet is made up of four fields: device ID, function ID, data, and error checking [31]. The reason why we chose to work with MODBUS is because it is designed specifically for the master-slave configuration [31]. In order to run MODBUS over the RS485 standard, we had two options: find a MODBUS implementation on GitHub or find a commercial software. After exploring both options thoroughly, it was decided that we would choose a commercial solution. We found multiple MODBUS implementations written in Python and tried importing the libraries necessary to run them [32]. The issue we ran into was that we would always get import errors, and scouring the internet for a solution turned out to be a fruitless task. In the end, we gave up and switched priorities. Searching for commercial MODBUS programs was rather straightforward as there are only a few solutions to choose from. After reading the product reviews, we decided to use SimplyModbus. SimplyModbus is a program that sends MODBUS packets between devices over a serial port [33]. This program comes in two variants; Slave and Master. The sender and receiver needs to have the Master and Slave installed respectively before communication can take place [33] [34]. For operation instructions, we read the manual on their website [33] [34] and experimented with numerous settings on the master program. Some of these settings included setting the COM port number specific to the PC that the program is installed on (both master and slave devices have a different port number), setting the baud rate to 9600 (for proper communication, the master and slaves must communicate with the same rate), inputting custom data into the registers for transportation purposes, and setting the registers to a 16-bit size so each packet contains that much data [33] [34]. After modifying these settings, we were able to send MODBUS packets with user-defined data between the two machines successfully.

For the next group-based deliverable, we were asked to use the RS485 clickboards to send MODBUS packets from a PC to one of our Teensy boards. This proved to be a much harder task because of the amount of time we spent troubleshooting issues related to connections between the two RS485 clickboards. We were able to successfully transmit the MODBUS packet from the master PC but the Teensy board never received anything. It turns out we had to flip the RX and TX connections between the two click boards to get the communication working. This issue was spotted after consulting the click board documentation and the Teensy pinout diagram [18] [35]. The following steps were taken to troubleshoot the problem:

1. Flashing the LED to ensure the board was powered on.

2. Sending a MODBUS packet between two laptops by daisy chaining two RS485 click boards. Since this worked, we knew that there was something wrong with the connections between the RS485 click board and the Teensy MCU.
3. Changing the connections to match the documentation [18] [35] and uploading code that transmits a byte of data from the Teensy to the laptop via the RS485 click board. The RX light on the RS485 cable lit up, which meant the byte was received and the connections were sound.

We then used the SimplyModbus Master program to send MODBUS packet to the Teensy board. Additionally, we uploaded a simple program on the Teensy that would echo the contents of the MODBUS packets being set from the PC back onto the screen. This setup proved to be valuable as any form of device communication relies on it or its variant.

For the final group deliverable, we were asked to substitute a Teensy board for a master device instead of a laptop. Being a master in this configuration means that one of our Teensy board would perform two tasks: performing its own home security-specific task and polling slave devices for information and making decisions based on this information. At this point, it was time to move from the commercial software SimplyModbus to written code which the Teensy can process. Various solutions [36] offered by Professor James Smith were studied and modified, however after a two-week struggle, we decided to implement our own MODBUS code.

One issue we ran into was the inconsistency of current being supplied by the Teensy boards to the RS485 click boards when sending and receiving data. When the Teensys sent information to other Teensys through the RS485 click boards, we noticed the click boards could only relay this information if they were powered by a +5V coming from the Teensy. The more confusing thing was that they could only receive incoming information when they were powered by a +3V coming from the Teensy. This presented a huge issue as it meant our communication protocol could only send or receive information by manually connecting the power line specific to make one of the two functions work. We believe this issue occurred due to something being wrong with the wiring. However, after looking through our code, we realized that we were not explicitly telling the RS485 clickboard when to send information and when to receive information. Things returned back to normal when these changes were implemented through the code.

Individual-based

For the individual deliverables, I was required to (1) safely turn off a stove to prevent fire and (2) control the actuation of a door when someone approaches it. For the first part, I did not use a stove because it was dangerous modifying the electrical components within, and furthermore, very inconvenient to have access to one for demonstration purposes. It was then decided that something else would be used in its place for the purposes of simulating a stove being turned off. The solution was to use an LED in its place. This was a simplification of grand proportions, however the main idea is still there; lighting up an LED can simulate the stove being shut off. Furthermore, this simplification made the deliverable much safer and convenient for demonstration purposes. The LED would connect to the Relay 2 Click board, which in turn would be operated by the

Teensy board. The Teensy board would poll the IrThermo temperature sensor and activate the relay once the temperature went over a certain threshold. The relay board acts as a circuit breaker, controlling the flow of current to the LED. As for the temperature sensor, it is not natively supported by the Teensy board. To operate the sensor, an external library had to be imported to the Arduino IDE from GitHub [26]. Afterwards, everything worked as expected.

The next steps associated with this deliverable include moving this setup to a real-life setting and integrating it to an actual stove. This would require putting the temperature sensor over the stove top. A timing mechanism would have to be integrated into the code so that the stove does not shut off the instant the temperature readings go over a certain threshold. This would give the user more freedom while cooking food that requires high temperatures over a prolonged period of time. If the user were to go over the allotted time, the stove would shut off as a precautionary measure. In order to turn a stove off, a circuit-breaker would have to be controlled by the Teensy board. This would require looking at the documentation that came with the circuit-breaker itself on how to connect it to an MCU, however, the operating principles would be similar to operating the relay switch.

For the deliverable pertaining to the door actuation, the UHPPOTE strike plate was substituted in for a door [24]. The strike plate requires 12V, something that the Teensy board cannot provide. Hence, the strike plate was connected to the power supply built into the breadboard. Furthermore, it was also connected to the relay board so that the flow of current can be controlled for the purposes of opening and closing the strike plate. As previously mentioned, one thing to note is that this part of the deliverable deals with the door actuation. The identity verification is the responsibility of a fellow group member. He uses a keypad to verify the code entered and then gets his Teensy to send a MODBUS packet over the RS485 clickboard to my Teensy. My Teensy board continuously polls for MODBUS packets and when the Teensy receives a packet, it performs a few checks before decrypting the message. Finally, after checking the message against a predefined array, it send a signal to the relay board to open the strike plate for a given amount of time before deactivating it.

As an aside, a new strike plate had to be ordered because the original one shorted out during testing purposes due to faulty connections. The strike plate has two connecting terminals, positive and negative. Since there was no wiring diagram that came with the strike plate, we experimented by plugging the strike plate thinking the negative terminal was deep into the door and the positive terminal was the one closest to the mounting screws. This was indeed not the case as our original setup ended up shorting the strike plate. Once the new strike plate came in, the connections were reversed, leading to it working without any issues.

NEXT STEPS

Time was of essence during the implementation of this project and if we had more of it on our hands, the quality of this project could have definitely been improved. During the development of the many functions needed for the deliverables, we came across many issues (mentioned throughout this report), especially those pertaining to the communication protocol used to connect the three individual prototypes. Configuring RS485 took a sizeable portion of our time (before this project, we had very little experience with hardware) and therefore any wiring issues could have been avoided easily by having an expert review all the connections at certain checkpoints throughout the development cycle.

On the software side of things, the development of the MODBUS brought up numerous issues: many of these were solved, however, some remain. The bare minimum our security system offers is the basic implementation of packet transmission which includes fields such as device ID, function ID, data, and error checking information. The error checking field was included in the MODBUS packet, however at this point, serves no purpose. We wish to integrate this field into the codebase of the project in the future.

The hardware provided at the beginning of our project was enough to prove concepts, but it would be ideal to obtain additional clickboards such as alarms and extra temperature sensors for adding additional functionality. At times we improvised with the provided hardware, shared information between prototypes, and responded accordingly. However, an ideal implementation would have seen more hardware and better MODBUS integration.

APPENDIX

COMPONENT PICTURES AND SCHEMATICS

PDS 500 BREADBOARD

Figure 1: Breadboard and its components.

Image source: http://www.dahonmarket.top/electronics-tweezers-c-630_1567_3101_3118/jdr-microdevices-pds500-powered-project-board-breadboard-new-p-20877.html

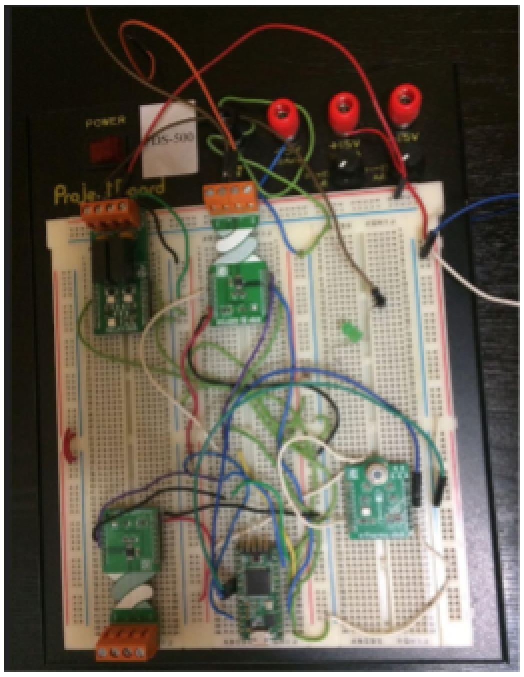

Figure 2: A picture of the breadboard with all the connected components.

RS485 2 CLICK BOARD

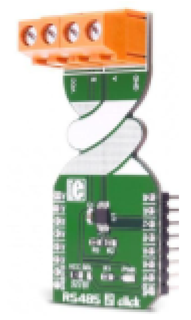

Figure 3: A picture of the RS485 clickboard

Image Source:
https://www.mikroe.com/rs485-2-click

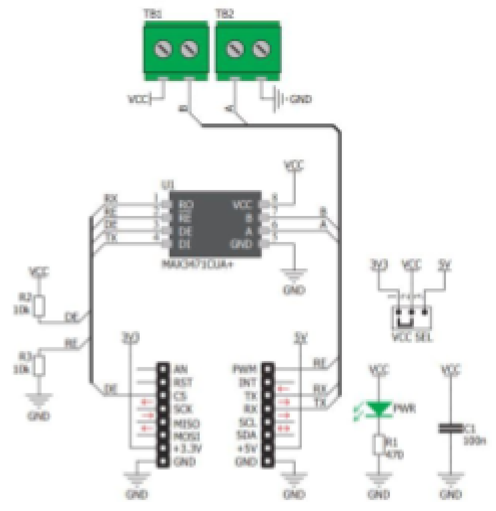

Figure 4: The schematic associated with the RS485 clickboard

Image source:
https://download.mikroe.com/documents/add-on-boards/click/rs485-2/rs485-2-click-schematic-v100.pdf

DTECH 6FT USB TO RS485 RS422 SERIAL CONVERTER ADAPTER CABLE

Figure 5: The adapter cable and its components.

Image source:
http://www.dtechelectronics.com/dtech-dt-5019-usb-to-rs485-422-cable_p118.html

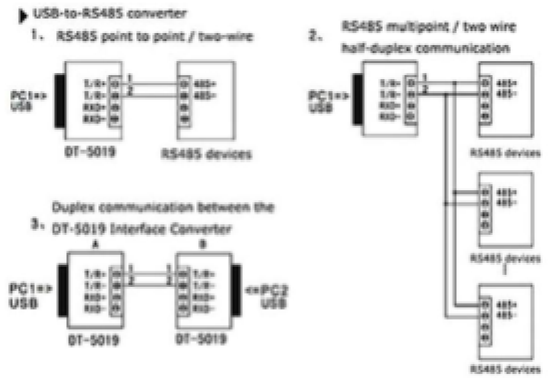
USB-to-RS485 converter
1. RS485 point to point / two-wire
2. RS485 multipoint / two wire half-duplex communication
3. Duplex communication between the DT-5019 Interface Converter
PC1=> USB
<=PC2 USB
DT-5019
RS485 devices

Figure 6: Connection schematic

Image source:
https://www.google.ca/search?q=dtech+usb+2.0+to+rs422/rs485+adapter+cable+schematic&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjHx8SUsdfcAhWkyoMKHa-GBIUQ_AUICigB&biw=1440&bih=833#imgrc=acX2rtFevdny7M:

TEENSY 3.1 MICROCONTROLLER

Figure 7: A picture of the Teensy 3.1 MCU

Image source: https://www.pjrc.com/teensy/card5a_rev7.pdf

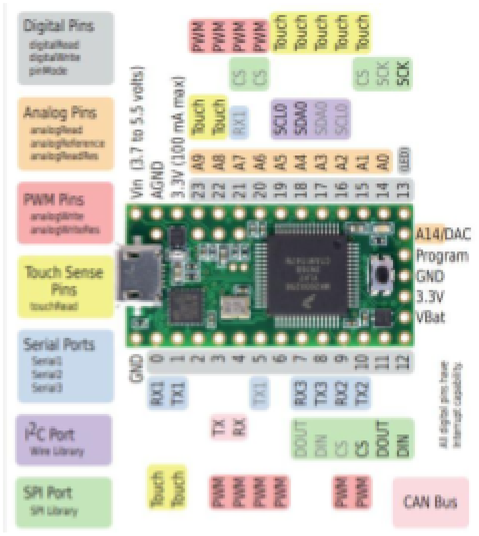

Figure 8: Pinout diagram for the Teensy 3.1

Image source: https://www.pjrc.com/teensy/card5a_rev7.pdf

IrThermo CLICK 3.3V TEMPERATURE SENSOR

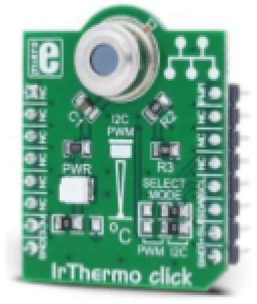

Figure 9: A picture of the IrThermo temperature sensor.

Image source: https://www.mikroe.com/irthermo-33v-click

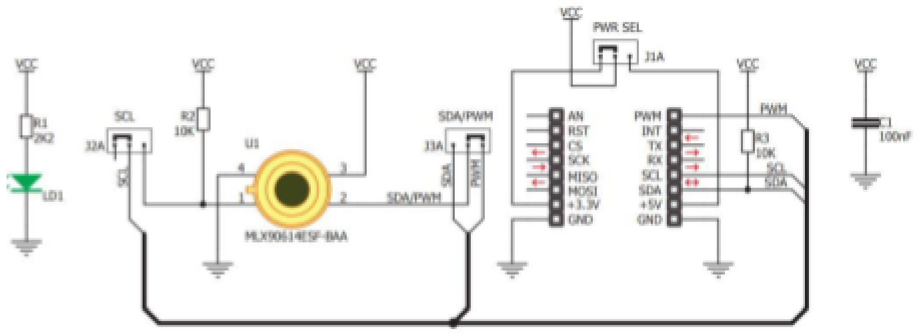

Figure 10: A schematic associated with the IrThermo temperature sensor.

Image source: https://download.mikroe.com/documents/add-on-boards/click/irthermo-33v/irthermo-33v-click-schematic-v100.pdf

RELAY CLICKBOARD

Figure 11: A picture of the relay boards

Image source: https://www.mikroe.com/relay-click

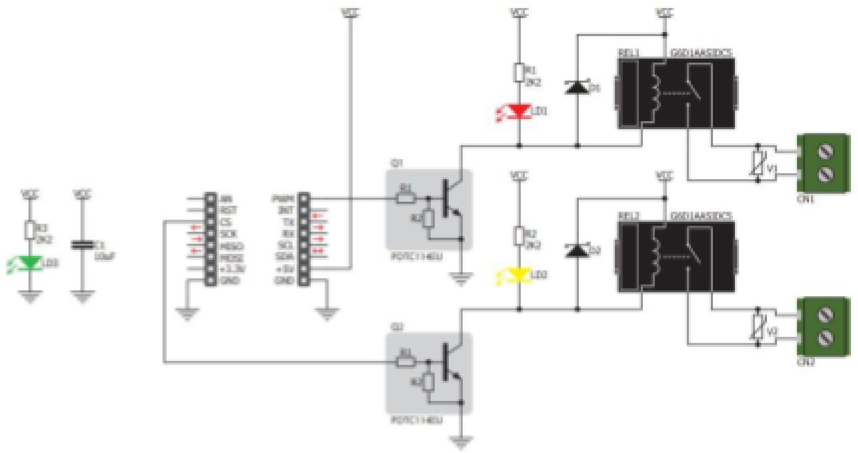

Figure 12: A schematic of the relay board

Image source: https://download.mikroe.com/documents/add-on-boards/click/relay/relay-click-schematic-v100-a.pdf

UHPPOTE FAIL-SAFE STRIKE PLATE

Figure 13: A picture of the strikeplate.

Image source: https://www.amazon.com/UHPPOTE-Electric-Strike-Secure-Control/dp/B00V45GWTI

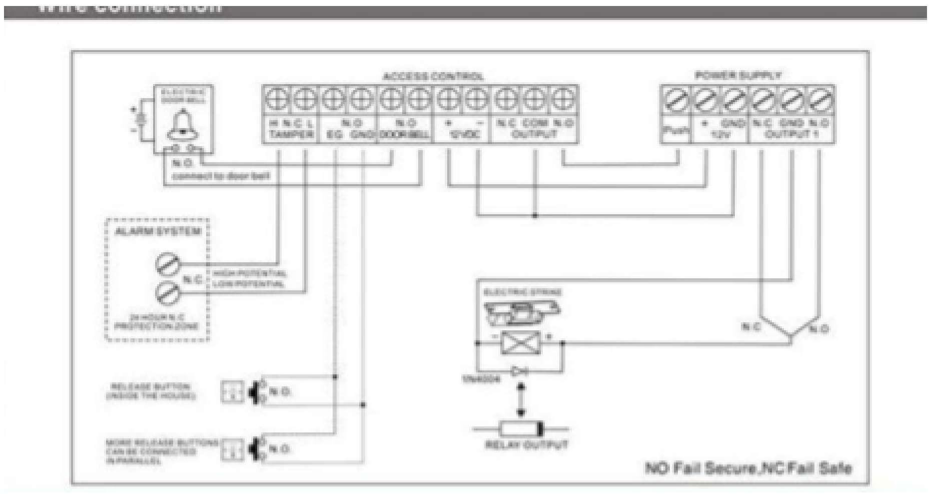

Figure 14: A Wire connection diagram associated with the strike plate

Image source: https://www.amazon.com/UHPPOTE-Electric-Strike-Secure-Control/dp/B00V45GWTI

MODBUS PACKET DISSECTION

Below is a table that explains what is encapsulated in each MODBUS packet:

Device ID (0-255)	Function ID	Message (Bytes)	CRC	Notes
0	-	-	-	Device ID 0 means it’s the master. Master just receives information from slaves. Andres’ board is the master. His device also acts as an activity logger, notifying the user what exactly is occuring within the premises at that time.
1	10	2	-	Saad sends Andres two MODBUS packets with 2 bytes of data each (temperature;) when the temperature readings go over a threshold and when the temperature returns below the threshold. Andres uses this information to turn a fan on or off and to notify the user of when the area is too hot and when it’s normal
1	20	2	-	Saad’s board sends Andres’ board 2 bytes of data when the strike plate opens. Andres takes this data and uses it to prompt the user that the door is now open.
2	10	17	-	Tapan’s device is the second slave. He sends Saad 17 bytes of data (16 bytes of encrypted data and 1 byte nRound value used to decrypt the information). This MODBUS packet is only sent when someone enters the correct code on the keypad. Saad tkes this packet and uses it to open a strike plate.

TEST REPORT

The following tables serve as a checklist for all the functions pertaining to my deliverables. The video demonstration of the whole system can be found on my Google Drive.

Requirement 1: Can the system process the information being relayed by the sensors about its surroundings?

Part being tested	Successful/Unsuccessful	Next Steps	Notes
Connections between the temperature sensor and MCU	Successful	-	The system was connected to only one sensor: temperature sensor. A program was loaded onto the Teensy board that would poll the sensor every 0.5 seconds and display the information on the screen. Hovering a hot object over the sensor (e.g. a hand) showed an increase in the readings (~ 28 ^{O}C) and moving a cold object over the sensor (e.g. ice cube) showed a decrease in the readings (~14 ^{O}C). The room temperature was 25.5 ^{O}C.

Requirement 2: Can the Teensy board receive information from other members' deliverables

Part being tested	Successful/Unsuccessful	Next Steps	Notes
Connections between the Teensy 3.1 and Teensy 3.6 (via the RS485 clickboard)	Successful	-	This test was conducted to see if my board would receive a MODBUS packet from Tapan's board. I uploaded tester code where I poll his board for information. He sends me the packet only when someone enters the right code on his keypad. My board was able to receive the packet every time he entered the right code. I also displayed the contents of the packet. Such print statements have been commented out from my code because this part has

			been thoroughly tested. This test also verifies that the connections between the two boards are correct.

Requirement 3: Can the system perform the deliverable-specific task based on the information being sent to the MCU?

Part being tested	Successful/Unsuccessful	Next Steps	Notes
Relay board and LED (simulate a stove being shut off)	Successful	The next step associated with this deliverable is to incorporate an actual stove into this setup. As mentioned previously in the Deliverable Summary, this would mean getting the MCU to control the circuit-breaker that the stove is connected to.	Based on the readings from the temperature sensor, the Teensy board controls the relay switch. If the temperature readings go beyond a threshold (27 ^{O}C), the MCU send a signal to activate Relay 1. One can see it work by hearing the clicking noise and noticing the LED on the clickboard turn on. Furthermore, the LED connected to Relay 1 turns on for the duration of time the temperature stays over the threshold mark. This test also verifies the fact that the connections pertaining to this deliverable are correct. See video regarding the first deliverable for a test run.
Relay board and the strike plate.	First time, unsuccessful (shorted out the plate) Second time, successful	-	Based on the MODBUS packet sent by Tapan's board, the message is decrypted and compared with an array. Afterwards, my board sends a signal to Relay 2 which is connected to the strike plate. The strike plate unlocks for a few seconds before locking

			up again. This part can be verified by the fact that when the strike plate unlocks or locks, it makes a clicking noise. One thing to note is that during testing, we shorted out the strike plate because of wrong connections (see the Deliverable Summaries for more detail). A new strike plate was ordered and configured with the right connections. See the video on full system integration for more details.

Requirement 4: Does the system notify the master of any abnormal activity it is designed to monitor?

Part being tested	Successful/Unsuccessful	Next Steps	Notes
Connections between the Teensy 3.1 and Teensy LC (temperature sensor)	Successful	-	My Teensy board sends a MODBUS packet to the master device whenever the temperature goes over a threshold. To start off, I uploaded code that made my Teensy board send test packets every 0.5 seconds. The packet was dissected by the master devices and its contents were displayed on the screen. Once this was successful, it was time to integrate this setup into the project. As soon as the temperature goes below the threshold, my board sends a final MODBUS packet letting the master know that things have returned back to normal. The master takes in these packets and notifies the user on the screen when there is danger and when

			things have returned to normal. Furthermore, the master turns on a fan for as long as the temperature stays above the threshold. When it receives the MODBUS packet that states the temperature has returned back to normal, it turns the fan off. This test also verifies that the connections between the two boards are correct. See the See the video on full system integration for more details.
Connections between the Teensy 3.1 and Teensy LC (unlocking of the strike plate)	Successful		Similar to above, my Teensy board sends a different MODBUS packet letting the master device know that someone just unlocked the strike plate. It then uses this MODBUS packet to notify the user on the screen that someone just unlocked the door. See the video on full system integration for more details.

REFERENCES

[1] C. Wilson, “Poll: Despite low crime rate, Canadians still believe Toronto is unsafe,” CP24, 23-Aug-2017. [Online]. Available: https://www.cp24.com/news/poll-despite-low-crime-rate-canadians-still-believe-toronto-is-unsafe-1.3557695. [Accessed: 14-Jul-2018].

[2] “A Look at Burglary Statistics in Toronto,” Citywide Locksmith, 24-Aug-2017. [Online]. Available: https://www.citywidelocksmith.ca/blog/look-burglary-statistics-toronto/. [Accessed: 14-Jul-2018].

[3] “Reliable Home Alarms And Security Systems | ADT,” ADT Home Security - Current Offer. [Online]. Available: https://www.adt.ca/en/home-security/products-solutions/home-security-systems. [Accessed: 14-Jul-2018].

[4] A. Jacobsson, M. Boldt, and B. Carlsson, “A risk analysis of a smart home automation system,” Future Generation Computer Systems, vol. 56, pp. 719–733, 2016.

[5] E. Fernandes, J. Jung, and A. Prakash, “Security Analysis of Emerging Smart Home Applications,” 2016 IEEE Symposium on Security and Privacy (SP), 2016.

[6] N. Good, R. Dhamija, J. Grossklags, D. Thaw, S. Aronowitz, D. Mulligan, and J. Konstan, “Stopping Spyware at the Gate: A User Study of Privacy, Notice and Spyware ,” Proceedings of the 2005 symposium on Usable privacy and security - SOUPS 05, 2005.

[7] A. Chaudhuri and A. Cavoukian, “The Proactive and Preventive Privacy (3P) Framework for IoT Privacy by Design,” Edpacs, vol. 57, no. 1, pp. 1–16, Feb. 2018.

[8] “Donload the Arduino IDE,” Arduino-Software. [Online]. Available: https://www.arduino.cc/en/Main/Software. [Accessed: 06-Aug-2018].

[9] “Teensyduino,” PJRC. [Online]. Available: https://www.pjrc.com/teensy/teensyduino.html. [Accessed: 06-Aug-2018].

[10] Rweather, “rweather/arduinolibs,” GitHub, 15-May-2018. [Online]. Available: https://github.com/rweather/arduinolibs/tree/master/libraries/Crypto. [Accessed: 07-Jul-2018].

[11] S. Bélanger, “RS485 vs Ethernet: Which One is Most Used in Industry?,” Robot-Enabled Factories: 5 Powerful Statistics for 2017, 25-May-2015. [Online]. Available: https://blog.robotiq.com/what-is-rs485-communication-protocol. [Accessed: 03-Aug-2018].

[12] M. Ali, “The Differences Between Bluetooth, ZigBee and WiFi,” LinkedIn SlideShare, 19-Sep-2015. [Online]. Available: https://www.slideshare.net/MostafaAli39/understanding-differences-between-bluetooth-zigbee-and-wifi. [Accessed: 03-Aug-2018].

[13] J. Axelson, Serial port complete: COM ports, USB virtual COM ports, and ports for embedded systems, 2nd ed. Madison, WI: Lakeview Research, 2007.

[14] “JDR Microdevices PDS-500 Powered Project Board Breadboard NEW,” DAHONMARKET. [Online]. Available: http://www.dahonmarket.top/electronics-tweezers-c-630_1567_3101_3118/jdr-microdevices-pds500-powered-project-board-breadboard-new-p-20877.html. [Accessed: 04-Aug-2018].

[15] “RS485 2 click,” MikroElektronika. [Online]. Available: https://www.mikroe.com/rs485-2-click. [Accessed: 02-Aug-2018].

[16] “DTECH DT-5019 USB TO RS485/422 Cable,USB To RS232 Cable,” Guangzhou Dtech Electronics Technology Co.,Ltd. [Online]. Available: http://www.dtechelectronics.com/dtech-dt-5019-usb-to-rs485-422-cable_p118.html. [Accessed: 02-Aug-2018].

[17] “Teensy 3.2 & 3.1 - New Features,” PJRC. [Online]. Available: https://www.pjrc.com/teensy/teensy31.html. [Accessed: 02-Aug-2018].

[18] “Teensy 3.1 Pins,” PJRC. [Online]. Available: https://www.pjrc.com/teensy/pinout.html. [Accessed: 02-Aug-2018].

[19] michu, “Serial Latency: Teensy vs. Arduino,” neophob.com, 16-May-2011. [Online]. Available: http://neophob.com/2011/04/serial-latency-teensy-vs-arduino/. [Accessed: 05-Aug-2018].

[20] “RS485/RS232 - How do I reduce latency on com ports,” Chipkin Automation Systems. [Online]. Available: https://store.chipkin.com/articles/rs232-how-do-i-reduce-latency-on-com-ports. [Accessed: 05-Aug-2018].

[21] “IrThermo click 3.3V,” MikroElektronika. [Online]. Available: https://www.mikroe.com/irthermo-33v-click. [Accessed: 02-Aug-2018].

[22] “MLX90614 family Single and Dual Zone Infra Red Thermometer in TO-39,” Melexis Microelectronic Integrated Systems. [Online]. Available: https://download.mikroe.com/documents/datasheets/MLX90614ESF.pdf. [Accessed: 03-Aug-2018].

[23] “Relay click,” MikroElektronika. [Online]. Available: https://www.mikroe.com/relay-click. [Accessed: 02-Aug-2018].

[24] “UHPPOTE Electric Strike Fail Secure NO Mode Lock a Part For Access Control Wood Metal Door -,” Amazon. [Online]. Available: https://www.amazon.com/UHPPOTE-Electric-Strike-Secure-Control/dp/B00V45GWTI. [Accessed: 02-Aug-2018].

[25] “Selecting correct cabling topology - Ring vs Star vs Bus,” State-of-art networking.net, 14-Dec-2015. [Online]. Available: https://www.ad-net.com.tw/selecting-correct-cabling-topology-ring-vs-star-vs-bus/. [Accessed: 02-Aug-2018].

[26] Adafruit, “adafruit/Adafruit-MLX90614-Library,” GitHub, 18-Jul-2018. [Online]. Available: https://github.com/adafruit/Adafruit-MLX90614-Library. [Accessed: 03-Aug-2018].

[27] FastTech.com, “$20.97 Dtech DT-5019 USB to RS485/RS422 Serial Port Adapter Cable (120cm) - plug & play at FastTech - Worldwide Free Shipping,” FastTech - Cool Gadgets and Electronics. [Online]. Available: https://www.fasttech.com/product/9636290-dtech-dt-5019-usb-to-rs485-rs422-serial-port. [Accessed: 03-Aug-2018].

[28] R. Meier, “Very useful stuff-Coolterm,” Roger Meier's Freeware. [Online]. Available: http://freeware.the-meiers.org/. [Accessed: 03-Aug-2018].

[29] “Downloads,” Simply Modbus. [Online]. Available: http://www.simplymodbus.ca/download.htm. [Accessed: 03-Aug-2018].

[30] Joel_E_B and Jimbo_Maittu_This, “Serial Terminal Basics,” Sparkfun. [Online]. Available: https://learn.sparkfun.com/tutorials/terminal-basics/coolterm-windows-mac-linux. [Accessed: 03-Aug-2018].

[31] "Modbus Tutorial for Arduino, Raspberry Pi and Intel Galileo," Cooking Hacks by Libelium. [Online]. Available: https://www.cooking-hacks.com/documentation/tutorials/modbus-module-shield-tutorial-for-arduino-raspberry-pi-intel-galileo/. [Accessed: 03-Aug-2018].

[32] Riptideio, "riptideio/pymodbus," GitHub, 13-May-2018. [Online]. Available: https://github.com/riptideio/pymodbus. [Accessed: 03-Aug-2018].

[33] "Simply Modbus - Modbus RTU/ASCII Master Test Software - Manual," Simply Modbus. [Online]. Available: http://www.simplymodbus.ca/MasterManual8.htm. [Accessed: 04-Aug-2018].

[34] "Simply Modbus - Modbus RTU/ASCII Slave Test Software - Manual," Simply Modbus. [Online]. Available: http://www.simplymodbus.ca/RTUSlaveManual8.htm. [Accessed: 04-Aug-2018].

[35] "RS485 2 click schematic v100," MikroElektronika. [Online]. Available: https://download.mikroe.com/documents/add-on-boards/click/rs485-2/rs485-2-click-schematic-v100.pdf. [Accessed: 04-Aug-2018].

[36] PaulStoffregen, "PaulStoffregen/Modbus-Master-Slave-for-Arduino," GitHub, 21-Jan-2018. [Online]. Available: https://github.com/PaulStoffregen/Modbus-Master-Slave-for-Arduino. [Accessed: 04-Aug-2018].