Greenhouse Monitoring and Automation

by

Kristian Doherty
Jevon Weisensel
Zhixuan Feng
Khanh Huynh
Braden Cross

Final report submitted in partial satisfaction of the requirements for the degree of

Bachelor of Science

in

Electrical and Computer Engineering

in the

Faculty of Engineering

of the

University of Manitoba

Faculty and/or Industry Supervisors:

Dr. Robert McLeod, Academic Advisor
Dr. Ahmed Byagowi, Co-Advisor
Dr. Carla Zelmer, Buller Greenhouse Manager

Winter 2016

© Kristian Doherty, Jevon Weisensel, Zhixuan Feng, Khanh Huynh, Braden Cross
Abstract

The Buller greenhouse was built in the 1960’s to house a variety of plants used for teaching and research purposes. Originally intended to be a temporary structure, all of the buildings temperature control systems are operated manually; while temperature and humidity is monitored using gauges that are read by staff inside the greenhouse. The aim of this project was to implement a scaled prototype monitoring and control system for one room of the greenhouse that could be accessed remotely via a web based server.

To achieve this goal, Arduino micro controllers were chosen to be used in a master/slave configuration. The slave controller is used to control a dc motor to open the window with provisions made for a relay system that could be used to operate additional temperature controls such as the swamp cooler and heater. It is also used to obtain readings from the temperature, humidity and light intensity sensors. These readings are relayed to the master controller using XBee wireless transceivers and then uploaded to the web server through an Ethernet shield. An automated control system was developed using the inputs from monitoring system to decide when to operate the temperature controls.

A prototype system was successfully developed that met the design constraints set out in the project proposal. This system logs temperature, humidity and light intensity readings to the web server every ten seconds. Through testing it was demonstrated that dc motor is capable of opening the windows in the greenhouse and that control signals could be sent remotely to operate the greenhouses temperature control systems. The secondary goal of developing software that utilizes the monitoring system data to operate the temperature controls was also achieved.
## Contributions

<table>
<thead>
<tr>
<th></th>
<th>Kristian Doherty</th>
<th>Jevon Weisensel</th>
<th>Zhixuan Feng</th>
<th>Khanh Huynh</th>
<th>Braden Cross</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitoring System</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>Control System</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>Wireless communication system</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Web server and Interface</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Hardware Testing</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Hardware Integration</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Final Report Content</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Final Report Compilation/Editing</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>Final Presentation</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

Legend:  ● Lead or Major Contributor  ○ Contributions
Acknowledgements

Throughout the course of our project we received advice and assistance from many members of staff at the University of Manitoba. We would like to thank and acknowledge the following people for their guidance:

Our academic supervisor, Dr. Robert McLeod, for his continual support throughout the school year. Significantly his help with the mechanical installation of the project saved us many frustrated hours.

Our Co-Advisor, Dr. Ahmed Byagowi, for proposing the project and giving technical advice on our design.

Our industry supervisor, Dr. Carla Zelmer, for allowing us to take over a room in her greenhouse and assisting us however she could.

The ECE Chief Technician, Glen Kolanksy, for helping us with budgeting and creating a realizable project scope. Always being available and ready to answer any questions we had through the year. His expertise was invaluable towards the completion of our project.

Electric Shop Technician, Sinisa Janjic, for helping us with part ordering and sourcing. Going out of his way to find the right place for the right part.

The Machine Shop Technicians, Cory Smit and Zoran Trajkoski, for helping make our project enclosure. We would’ve been at a loss without their assistance and the skills they provided.

Finally we would like to thank Dr. Derek Oliver, Aidan Topping, and Dan Card for providing expertise and guidance all year long. Providing valuable feedback allowing us to progress and improve our written and oral communication skills.
Table of Contents

Abstract .......................................................................................................................... i
Contributions ............................................................................................................... ii
Acknowledgements ..................................................................................................... iii
Table of Contents ......................................................................................................... iv
List of Figures .............................................................................................................. vii
List of Tables ............................................................................................................... viii
Nomenclature .............................................................................................................. ix
1 Introduction ............................................................................................................. 1
  1.1 Motivation ........................................................................................................... 1
  1.2 Design Approach ............................................................................................... 1
  1.3 System Overview ............................................................................................... 2
2 Monitoring System .................................................................................................. 4
  2.1 Hardware ........................................................................................................... 4
    2.1.1 Arduino UNO ............................................................................................... 4
    2.1.2 DHT11 Temperature and Humidity Sensor ............................................... 5
    2.1.3 DFRobot Light Sensor ............................................................................... 6
    2.1.4 DS18B20 Temperature Sensor .................................................................... 7
    2.1.5 Liquid Crystal Display .............................................................................. 8
  2.2 Software ........................................................................................................... 9
    2.2.1 DHT11 Temperature and Humidity Sensor ............................................... 9
    2.2.2 DFRobot Light Sensor ............................................................................... 10
    2.2.3 DS18B20 Digital Temperature Sensor ..................................................... 10
  2.3 Integration .......................................................................................................... 10
3 Control System ....................................................................................................... 12
  3.1 Hardware ........................................................................................................... 12
    3.1.1 MD10C Motor Driver ............................................................................... 12
    3.1.2 Sun Founder 2 channel Relay .................................................................... 13
    3.1.3 8507840 Linear Actuator ........................................................................... 13
    3.1.4 Denso 730557 7030 L Rotational Motor ................................................... 14
    3.1.5 Feedback Switches ..................................................................................... 14
Appendix C – Web Interface Code ........................................................................................................61
Appendix D – Website Login Code ......................................................................................................66
Appendix E - XBee Parameter Configuration ......................................................................................67
Appendix F - Budget .............................................................................................................................68
Curriculum Vitae ...................................................................................................................................69
List of Figures

Figure 1-1: The Buller greenhouse. ................................................................. 2
Figure 1-2: System overview. ............................................................... 3
Figure 2-1: Humidity sensor connections. ........................................... 6
Figure 2-2: Light sensor connections ............................................. 7
Figure 2-3: Temperature sensor connections. .................................. 8
Figure 2-4: LCD connections.............................................................. 9
Figure 2-5: Monitoring system enclosure. ........................................ 11
Figure 3-1: DC motor and belt............................................................. 14
Figure 3-2: Feedback Switches. (a) Window open switch. (b) Close window switch........ 15
Figure 3-3: Automated control flowchart. ...................................... 17
Figure 3-4: Control system enclosure............................................... 18
Figure 4-1: Connection between the master controller and XBee USB adaptor ........................................... 20
Figure 4-2: Setup for slave controller with XBee router......................... 20
Figure 4-3: XBee mesh network (C –Coordinator, R-Router, E-End Point). ................................................. 21
Figure 4-4: Arduino Mega with coordinator XBee. .................................... 22
Figure 4-5: XBee parameter configuration setup. .................................. 23
Figure 4-6: XTCU software interface. .................................................. 24
Figure 5-1: Web server flow chart...................................................... 29
Figure 6-1: The login page on different platforms. (a) Google Chrome running on a PC. (b) Safari running on an iPhone......................... 29
Figure 6-2: The interface page on different platforms. (a) Internet Explorer running on a PC. ...................... 31
Figure 6-3: Web interface flow chart.................................................. 33
List of Tables

Table 2-1: Arduino UNO specifications. ................................................................. 5
Table 2-2: Technical specifications of the humidity sensor. .................................... 5
Table 2-3: Technical specification of the light sensor ............................................. 6
Table 2-4: Technical specification of the temperature sensor. ................................. 8
Table 3-1: Motor controller logic table ................................................................. 13
Table 3-2: Automated Control Modes ................................................................. 16
Table 4-1: XBee S2 specifications [8] ...................................................................... 21
Table 4-2: Serial data look up table. ....................................................................... 25
Table 5-1: Arduino Mega specifications ................................................................... 27
Table 5-2: Valid requests look up table. ................................................................... 28
Nomenclature

**Arduino** – Open source microcontroller

**GUI** – Graphical User Interface

**Html** – Hypertext Markup Language

**HTTP** – Hypertext Transfer Protocol

**IP** – Internet Protocol

**MAC** – Media Access Control

**PC** – Personal Computer

**PWM** – Pulse Width Modulation

**RH** – Relative Humidity

**Shield** – peripheral boards that are designed to easily interface with Arduino microcontrollers

**URL** – Uniform Resource Locator

**WPA2** – Wi-Fi Protected Access II

**Xbee** – Low Power wireless transceiver
Chapter 1

1 Introduction

1.1 Motivation

The Buller greenhouse (seen in Figure 1-1) was built in the 1960’s to house a variety of plants used for teaching and research purposes. The building itself comprises of nine rooms that each have their own unique climate conditions. This allows some very unique plants to be kept in the greenhouse from a wide range of locations. The main concern for each of these rooms is that they stay within a proper temperature range. In the summer months this is primarily done by manually opening and closing windows. This requires staff to come in on the weekends if the outside temperature gets too high. The goal was to implement a system that allows remote monitoring and control to help reduce the amount of time staff needs to be present in the greenhouse.

The scope of the project was to produce a working prototype monitoring and control system for one room of the Buller greenhouse. This system can be accessed remotely using a web server on any internet connected device. This project could be easily expanded for use in all of the rooms in the greenhouse so that each room could be automated. This report will demonstrate that a fully automated greenhouse could be realized using our design and an expanded budget.

1.2 Design Approach

Arduino microcontrollers were chosen due to familiarity with the C language, and the wide variety and availability of peripheral devices. By having access to components in a timely manner the work schedule set out in the proposal was adhered to. This left adequate time to react to challenges and make changes to the design while still meeting the project deadline. To allow for easier development of the project among group members a modular design approach was chosen. The project was broken down into 4 major subsystems: monitoring system, control system, wireless communications system and the web server based user interface. This allowed for easier debugging and helped make the integration phase of the project go smoothly.
1.3 System Overview

This project utilizes two Arduino microcontrollers in a master/slave configuration to obtain sensor readings and operate the greenhouse temperature controls. For the master controller an Arduino Mega 2560 was chosen. The master controller contains the code for the web server which is uploaded through an Ethernet shield. Additionally this controller transmits control signals and receives sensor data from the slave controller through a pair of XBee transceivers. For the slave controller an Arduino UNO R3 was chosen which is responsible for collecting and transmitting data. This includes temperature, humidity and light intensity readings as well as receiving and relaying control signal input by the user to the motor controller. The slave controller is also connected to an LCD screen which displays the readings from the monitoring system locally. A visualization of the system overview can be seen in Figure 1-2.

Figure 1-1: The Buller greenhouse.
Figure 1-2: System overview.
Chapter 2

2 Monitoring System

The main purpose of the monitoring system is to obtain temperature readings from the greenhouse to be relayed to a remote user. Additionally the system has been expanded to include humidity and light intensity monitoring as desired parameters. These parameters are fed back into our control system to allow for automated control. Currently the control system only has the ability to control temperature due to the limitations of the existing hardware in the greenhouse, but in future projects automated humidity and lighting control could be readily implemented. The design components of the monitoring system are broken down into two main sections: hardware and software.

2.1 Hardware

The monitoring system consists of four sensors that interface with the slave Arduino UNO microcontroller. These sensors include: a temperature sensor, humidity sensor, light intensity sensor and two contact switches that determine if the window is open or closed. Sensor data is relayed wirelessly to the Arduino Mega where it is logged and uploaded to the web server.

2.1.1 Arduino UNO

The Arduino UNO was chosen as the slave microcontroller. The UNO is a microcontroller board based off of the ATmega328P chip. The UNO provides 14 digital I/O pins along with six analog input pins. The monitoring system required a total of eight digital pins and three analog pins making the Arduino UNO a perfect candidate. The UNO has a clock speed of 16 MHz and a flash memory of 32 KB which was more than enough to run and process the monitoring system code. Below in Table 2-1 specifications of the Arduino UNO are summarized [1].
Table 2-1: Arduino UNO specifications.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Conditions</th>
<th>Minimum</th>
<th>Typical</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller</td>
<td>Atmega328P</td>
<td>5V</td>
<td>7-12V</td>
<td>20 mA</td>
</tr>
<tr>
<td>Operating Voltage</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supply Voltage</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC Current per I/O Pin</td>
<td></td>
<td>20 mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flash Memory</td>
<td></td>
<td>32 KB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clock Speed</td>
<td></td>
<td>16 MHz</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.1.2 DHT11 Temperature and Humidity Sensor

The DHT11 is a dual temperature and humidity sensor, meaning that it can read both temperature and humidity. Even though it has dual operation it is only used as a humidity sensor. A higher accuracy temperature sensor is implemented instead. This sensor uses serial communication that is reliable and has long term stability. The DHT11 requires a pull-up resistor, approximately 5 KΩ, to be placed between the data wire and voltage supply. This prevents the input pin from floating in an undefined state or going high if the sensor was disconnected. The technical specification below in Table 2-2 shows an overview of the DHT11 humidity sensor. Figure 2-1 shows the DHT11 connections with the Arduino UNO [2].

Table 2-2: Technical specifications of the humidity sensor.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Conditions</th>
<th>Minimum</th>
<th>Typical</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td></td>
<td>1% RH</td>
<td>1% RH</td>
<td>1% RH</td>
</tr>
<tr>
<td>Accuracy</td>
<td>25°C</td>
<td>±4% RH</td>
<td>±5% RH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0-50°C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Measurement Range</td>
<td>0°C</td>
<td>30% RH</td>
<td>90% RH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>25°C</td>
<td>20% RH</td>
<td>90% RH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>50°C</td>
<td>20% RH</td>
<td>80% RH</td>
<td></td>
</tr>
<tr>
<td>Response Time (Seconds)</td>
<td>1/e(63%) 25°C</td>
<td>6s</td>
<td>10s</td>
<td>15s</td>
</tr>
<tr>
<td></td>
<td>1m/s Air</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.1.3 DFRobot Light Sensor

The DFRobot light sensor uses a phototransistor that outputs a DC voltage depending on the level of ambient light. The Arduino reads this as an analog value from 0 to 1024. The maximum range of this sensor is 6000 Lux which translates to an output of 1024. These values are sent to the web server where the ambient light in the greenhouse can be monitored by a remote user. The light sensor specifications are shown below in Table 2-3. Figure 2-2 shows the light sensor connections with the Arduino UNO [3].

Table 2-3: Technical specification of the light sensor

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply Voltage</td>
<td>3.3V to 5V</td>
</tr>
<tr>
<td>Illumination Range</td>
<td>1 Lux to 6000 Lux</td>
</tr>
<tr>
<td>Response Time (Seconds)</td>
<td>15µs</td>
</tr>
<tr>
<td>Interface</td>
<td>Analog</td>
</tr>
<tr>
<td>Size</td>
<td>22x30mm(0.87”x1.18”)</td>
</tr>
</tbody>
</table>
2.1.4 DS18B20 Temperature Sensor

The DS18B20 digital thermometer provides 9-bit to 12-bit Celsius temperature measurements that communicates over a 1-Wire bus that requires only one data line to communicate with the Arduino UNO. Using 12-bit resolution takes longer to process on the DS18B20 but since speed is not an issue the higher resolution is worth it. The DS18B20 is consists of 4 main components that include a 64-bit ROM, an analog to digital converter, scratchpad memory, and the temperature sensor itself. The 64-bit ROM holds the unique address of the sensor. This allows for multiple DS18B20 sensors to all communicate on one digital pin. For the prototype a single DS18B20 was used however this is easily expandable to allow for temperature monitoring of multiple locations. The scratchpad holds the 2 byte temperature sensor value and updates when a read command is received. Like the DHT11 sensor a pull-up resistor is required. Table 2-4 shows the specifications for the DS18B20 temperature sensor[4]. Figure 2-3 shows the DS18B20 connections with the Arduino UNO.
Table 2-4: Technical specification of the temperature sensor.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Conditions</th>
<th>Minimum</th>
<th>Typical</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temperature Range</td>
<td>-55°C</td>
<td></td>
<td>+125°C</td>
<td></td>
</tr>
<tr>
<td>Supply Voltage</td>
<td>Local Power</td>
<td>+3V</td>
<td></td>
<td>+5.5V</td>
</tr>
<tr>
<td>Thermometer Error</td>
<td>-10°C to +85°C</td>
<td>±15°C to +30°C</td>
<td>±0.5°C</td>
<td>±2°C</td>
</tr>
<tr>
<td>Active Current</td>
<td>$V_{\text{supply}} = 5\text{V}$</td>
<td>1mA</td>
<td></td>
<td>1.5mA</td>
</tr>
</tbody>
</table>

Figure 2-3: Temperature sensor connections.

2.1.5 Liquid Crystal Display

This particular SainSmart 2004 LCD has its own on-board microcontroller. This means that it has hardware that converts serial control signals received by the microcontroller board into the right output signals for this LCD display. The on-board microcontroller communicates with the Arduino UNO through an I2C bus which stands for Inter-Integrated Circuit. There are special pins on the Arduino UNO that allow for I2C communication. There are two connector pins on the LCD which are labeled as SDA (System Data) and SCL (System Clock) that connect to analog pins A4 and A5 on the Arduino UNO. The other two pins on the LCD display are ground and the voltage that needs to be supplied by the microprocessor. By using a LCD with its own microcontroller the number of pins required is reduced on
the Arduino. Other features include a 4-line 20-character LCD module that has a contrast control nob and a backlight. The LCD is being used to display our temperature sensor, humidity sensor and our light sensor directly in the greenhouse room. LCD configuration with the Arduino UNO is shown below.

![Figure 2-4: LCD connections.](image)

### 2.2 Software

The monitoring software is written in the Arduino language and is implemented on the slave Arduino UNO. The three sensors are polled every 5 seconds and then the values are sent to the master Arduino Mega via Xbee as well as the LCD. The monitoring software can be found in Appendix A.

**2.2.1 DHT11 Temperature and Humidity Sensor**

The humidity sensor is read by the Arduino from digital pin 10. The DHT11 sensor comes with an Arduino ready library, this allows for easy integration with any system. The DHT sensor works on a 1 wire-2 way communication system. The Arduino will first send a high ready signal to the DHT, then the DHT will send back its data package. This data package is made up as follows: 8 bit integral humidity data + 8 bit decimal humidity data + 8 bit integral temperature data + 8 bit decimal temperature data + 8 bit check sum for a total of 40 bits. This process takes approximately 4 ms. The humidity value is then sent to the LCD and web server.
2.2.2 DFRobot Light Sensor

When the Arduino receives the light sensor value from analog pin 2 it is translated from 0-1024 to 0-100%. This gives the user a comprehensible number that can be used to judge the light intensity in the room. This value is displayed both on the LCD and Web server.

2.2.3 DS18B20 Digital Temperature Sensor

The temperature sensor data bus is implemented on digital pin 9. The digital sensor function first searches the 1-wire bus for all available devices, in our case there is just the one. If no devices are found or if a device with an improper address is found, it returns -1000. Next it writes to the rom of the sensor at the first address to initialize the analog to digital conversion. After that the function can send a read command to the scratchpad and receive a digital value in return. This value is then converted from binary to decimal and returned as a float. This process is then repeated for all other devices on the 1 wire bus. The temperature value is sent to the LCD and web server.

2.3 Integration

All the sensors are connected to the Arduino UNO via a solder-able prototyping board. In the future this could be changed to a PCB. The Arduino UNO and the prototyping board are held within a 221x150x60 mm plastic enclosure. The LCD is mounted to the front of the enclosure to give the user easy access to the greenhouse room conditions. The monitoring system is powered by a 9V 1A power adapter that plugs into a standard 120V wall outlet. A photo of the enclosure setup can be seen in Figure 2-5.
Figure 2-5: Monitoring system enclosure.
Chapter 3

3 Control System

The current method for temperature control in the greenhouse involves manually opening and closing the windows. The main purpose of the control system is to allow the windows to be controlled remotely using an electric motor to turn the existing manual crank. The secondary purpose is to automate this process where the windows are opened or closed without human interaction. This is achieved by using the temperature readings obtained by the monitoring system to determine which control mode is selected. The automated control mode can be overridden by user commands sent from the user via the web server.

3.1 Hardware

The control system consists of two major components that interface with the Arduino UNO microcontroller. These components are: the dc motor used to open and close the greenhouse windows, and the dc motor controller used to control it. Additionally a 250V ac relay that uses 5V control signals from the Arduino has been included which could be used to control additional temperature control devices such as swamp coolers or heaters.

3.1.1 MD10C Motor Driver

The MD10C motor driver takes input from the microcontroller to control the direction of current flow from the 12V power supply to the dc motor. This controls the direction of the motor and allows the window to be opened or closed. The driver is controlled by inputs to the DIR (direction) and PWM (pulse width modulation) pins [5].
<table>
<thead>
<tr>
<th>Input Pins</th>
<th>Window Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM</td>
<td>DIR</td>
</tr>
<tr>
<td>LOW</td>
<td>X</td>
</tr>
<tr>
<td>HIGH</td>
<td>LOW</td>
</tr>
<tr>
<td>HIGH</td>
<td>HIGH</td>
</tr>
</tbody>
</table>

The DIR pin controls the direction the motor spins and the PWM pin allows for speed control of the motor by taking input values from 0-255 from the microcontroller. The logic table can be seen above in Table 3-1.

3.1.2 Sun Founder 2 channel Relay

The Sun Founder 2 channel 5V relay shield consists of two SRD-05VDC-SL-C relays which can be used to switch devices up to 250V AC with up to 10A of current using 5V input signals from the Arduino microcontroller [6]. This active low device switches the relay on when it receives a 0V signal and off when it receives a non-zero signal. This device adds the capability to operate additional temperature control devices in the greenhouse such as swamp coolers or heaters.

3.1.3 8507840 Linear Actuator

The 8507840 linear actuator is the original component chosen to open and close the greenhouse windows. Using a 12V dc power supply the actuator can lift up to 270 lbs with a stroke length of 12 inches [7]. In the testing phase the approach for opening the windows was changed from using a linear actuator to using a rotational motor as it is easier to implement with the existing manually operated greenhouse controls. This allowed for less intrusive testing of the system in the greenhouse and minimized the mounting hardware required.
3.1.4 Denso 730557 7030 L Rotational Motor

The rotational motor was chosen as the replacement for the linear actuator. It is mounted to the greenhouse wall just underneath the existing manual window crank. The motor is connected to the existing manual window control by a belt and pulley which turns the crank that opens and closes the window. The 730557 7030 L Motor can be operated at different speeds which are set by the power supplied to it. The biggest advantage of a rotational motor over the linear actuator is that the existing manual control system can still be operated without having to remove the motor. The installed motor system can be seen below.

Figure 3-1: DC motor and belt.

3.1.5 Feedback Switches

To ensure that the motor doesn’t move the window too far in one direction we elected to use a pair of contact switches to determine the absolute position of the window. One switch is used to send a LOW signal when the window is fully opened and another switch is used to send a LOW signal when the window is fully closed. The switches are wired in the normally closed mode. This was done so that if a connection breaks or power is lost the motor will not run. When the microcontroller receives one of these
signals, it sends a control signal to the motor controller to cut power to motor that opens and closes the window. The contact switches require pull-down resistors to prevent the digital input from floating.

![Feedback Switches](image)

Figure 3-2: Feedback Switches. (a) Window open switch. (b) Close window switch.

### 3.2 Software

The control system software has two modes of operation: manual and automatic. In manual mode the control system takes user commands from the web server which are used to operate the temperature control devices. The automatic mode uses temperature input from the monitoring system to operate the temperature control devices when the temperature in the greenhouse falls into certain ranges. The control system code can be seen in Appendix A.

### 3.2.1 Manual Control Mode

While in the manual mode of operation, the control system takes inputs from the user via two push button and two toggle switches on the GUI. The two push buttons are used for opening the window and closing the window while the toggle switches are used for the swamp cooler and heat controls. The GUI can be accessed on the website by anyone with authorization allowing for the greenhouse staff to
operate the temperature controls while off-site. The commands are sent from the master controller to the slave controller via XBee transceivers.

### 3.2.2 Automated Control Logic

The automated control system has been programmed for four temperature ranges and their associated commands, they are shown below in Table 3-2.

<table>
<thead>
<tr>
<th>Temperature Range</th>
<th>Control Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Window</td>
</tr>
<tr>
<td>R1</td>
<td>Closed</td>
</tr>
<tr>
<td>R2</td>
<td>Closed</td>
</tr>
<tr>
<td>R3</td>
<td>Open</td>
</tr>
<tr>
<td>R4</td>
<td>Open</td>
</tr>
</tbody>
</table>

By reducing the number of possible operating modes to just four allowed for more efficient code development. The automatic control mode utilizes a polling method to continuously check which temperature range the greenhouse falls into. This is done using fall through switch case statements that are switched by using the current temperature reading obtained from the monitoring system. The logic structure for the automatic control mode can be seen in the flowchart in Figure 3-3.
After the temperature value is updated the program checks to see which range the temperature falls into. The window status is then checked by reading the feedback switches before any control actions are taken. Once the window is moved to the proper position, the motor controller is turned off by setting the parameter winControl LOW which is connected to the PWM pin of the motor controller; this ensures that the motor is disengaged. Finally the heater and fan controls signals are sent to the 2 channel relay by setting the parameters fanControl and heatControl to HIGH or LOW.

### 3.3 Integration

The DC motor controller is enclosed in a 167x107x53 mm plastic enclosure. The feedback switches and DC motor controller signal wires are connected to a solder-able prototyping board also contained in the plastic enclosure. These wires are connected to the monitoring system enclosure by a wire harness. The DC motor controller and motor are powered by a 12V 5A power adapter than can plug into a standard 120V wall outlet. Below is a photo of the control system enclosure setup in Figure 3-4.
Figure 3-4: Control system enclosure.
Chapter 4

4 Wireless Communication System

A wireless communication system was chosen to transmit data between the master and slave controllers to give the system the ability to be easily expanded. This makes it possible for the master controller to communicate with additional slave controllers which could be used to expand the system to additional rooms in the greenhouse. Another advantage of wireless communication is that it reduces the number of pins required on the master controller for data transfer. This type of communication requires only one serial communication channel on the master controller in order to communicate with multiple slave controllers in the greenhouse. Furthermore wireless communication eliminates the burden of having to run a complex wire system throughout the greenhouse.

4.1 Hardware

To implement wireless communication between the master and slave controllers XBee transceivers were chosen. XBees are easy to interface with Arduino due to the availability of Arduino shields which allow for the transceivers to be directly mounted to the Arduino boards. XBee also meets the range requirements of the project because a mesh network of XBees can be created where intermediary points act as routers between the Coordinator XBee and End Point XBee.

The coordinator XBee connects to the Mega Arduino master board through a USB Adaptor. The USB Adaptor utilizes the TX1 and RX1 pins for serial communication since TX0 and RX0 pins are used for Ethernet shield communication. This setup is shown below in Figure 4-1. The router XBee communicates with the Arduino UNO through an XBee shield. The shield connects with RX and TX pins of the Arduino UNO. The slave setup can be seen below in Figure 4-2. Currently the system does not have an end point XBee as only a single room is being monitored however this could easily be added to expand the capabilities of the system.
Diagram pins Connections:

- The TX1 pin on the Arduino connects to the XBee USB Adaptor RX (Data in)
- The RX1 pin on the Arduino connects to the XBee USB Adaptor TX (Data out)
- The 5V, GND and Reset pins of the Arduino are connected to the 5V, GND and Reset of the USB Adaptor respectively.
4.1.1 Xbee S2

Two XBees are required to meet the needs of the project. The first Xbee acts as a coordinator which receives data and sends control commands to the second Xbee. This second Xbee functions as a router which communicates with other end points or routers. Based on the size of the greenhouse, Xbee S2’s were chosen for wireless communication as they have a larger operating range than the lower power Xbee model.

![Xbee mesh network diagram](image)

Figure 4-3: Xbee mesh network (C – Coordinator, R - Router, E - End Point).

Table 4-1: Xbee S2 specifications [8]

<table>
<thead>
<tr>
<th>Specification</th>
<th>XBee (S2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indoor/Urban Range</td>
<td>Up to 300 ft. (90 m), up to 200 ft (60 m)</td>
</tr>
<tr>
<td>Supply Voltage</td>
<td>3.0 - 3.4 V</td>
</tr>
<tr>
<td>Transmission Power Output</td>
<td>10mW (+10 dBm)</td>
</tr>
<tr>
<td>Operating Current (Transmission, max output power)</td>
<td>295mA (@3.3 V)</td>
</tr>
<tr>
<td></td>
<td>170mA (@3.3 V)</td>
</tr>
<tr>
<td>Operating Current (Receive)</td>
<td>45 mA (@3.3 V)</td>
</tr>
<tr>
<td>Operating Temperature</td>
<td>-40 to 85º C</td>
</tr>
</tbody>
</table>
4.1.2 XBee USB adaptor

The XBee adaptor is required for two functions, the first being as an adaptor for the configuration of the XBee parameters. This is done through the XCTU software explained in Section 4.2. The second function is connecting the coordinator XBee to the Arduino Mega. Using an adaptor instead of a shield for the coordinator XBee allows the XBee to connect to the TX1 and RX1 pins of the Arduino Mega. This prevents interference with the XBee and Arduino when compiling code onto the Arduino. Below in Figure 4-4 is a photo of the Arduino Mega and adaptor setup.

![Arduino Mega with coordinator XBee](image)

Figure 4-4: Arduino Mega with coordinator XBee.

4.1.3 XBee Shield

The shield is required to connect the router XBee to the Arduino UNO. This prevents the need for complicated wiring and leaves additional I/O pins available for use. The shield also displays the status of the XBee module on onboard LEDs which light up when the XBee is connected to the network [9]. This assists in testing and setup.
4.2 XTCU Software

XCTU software is used to configure the XBee. The USB adaptor is required in order to connect the XBee to a computer for configuration with the XCTU software. The XCTU software is required to configure multiple XBees to communicate with each other in a mesh network system. These XBees are set up to communicate using serial pins on Arduino boards. The parameter configurations can be found in Appendix E. The configuration setup and software can be seen below in Figure 4-5 and Figure 4-6, respectively [10].

Figure 4-5: XBee parameter configuration setup.
4.3 Arduino Software

In order to interface the XBee with the Arduino, a look-up table was designed to classify the data being transferred between the master and slave controllers. The look-up table can be seen below in Table 4-2. Before transferring any data to the other XBee, a unique data tag is assigned to that type of data and is sent first, followed by the data itself. The code numbers start at 101. For example, when the slave controller sends 30° C to the master controller, it will firstly send 101 to signal that temperature data will be sent over serial communication.
Table 4-2: Serial data look up table.

<table>
<thead>
<tr>
<th>Data Tag</th>
<th>Data types</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Temperature</td>
</tr>
<tr>
<td>102</td>
<td>Humidity</td>
</tr>
<tr>
<td>103</td>
<td>Light intensity</td>
</tr>
<tr>
<td>104</td>
<td>Window open</td>
</tr>
<tr>
<td>105</td>
<td>Window close</td>
</tr>
<tr>
<td>106</td>
<td>Control mode: Manual</td>
</tr>
<tr>
<td>107</td>
<td>Control mode: Auto</td>
</tr>
</tbody>
</table>

In the future, when there are more XBees in the system, our code will need to be modified to recognize the high and low source address of data being transferred to the master Arduino.
Chapter 5

5 Web Server

The web server is hosted on the Arduino Mega. It can be accessed by any internet connected device such as smart phones or PCs by using the server’s IP address as the URL. The web server is responsible for sending the received greenhouse sensor values as well as the web interface to the connected users’ browsers. It also receives commands from the users and responds accordingly.

5.1 Hardware

An Ethernet shield is used on the Arduino Mega to enable internet connectivity [11]. An 8GB micro SD card is plugged in the shield’s on-board SD card slot which is utilized for storing the html web page file. Ethernet was chosen over Wi-Fi because the University of Manitoba campus uses WPA2-Enterprise security protocols which are not supported by most Wi-Fi shields. Additionally, Ethernet provides a more reliable connection, and the existing Ethernet connection in the greenhouse uses wide area network IP address, which means users can access the IP address from any location on or off campus.

5.1.1 Arduino Mega

The Arduino Mega microcontroller board was chosen as the master controller to host the web server and coordinator XBee. The Arduino Mega runs on the ATmega1280 and has 54 digital I/O pins as well as 16 analog input pins. The main reason for selecting the Mega is due to its increased flash memory of 128 KB. This provides enough room to host the web server as well as the data logging software. Below in Table 5-1 is a summarized list of the Mega specifications [12].
Table 5-1: Arduino Mega specifications.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller</td>
<td>Atmega1280</td>
</tr>
<tr>
<td>Operating Voltage</td>
<td>5V</td>
</tr>
<tr>
<td>Supply Voltage</td>
<td>7-12V</td>
</tr>
<tr>
<td>DC Current per I/O Pin</td>
<td>40 mA</td>
</tr>
<tr>
<td>Flash Memory</td>
<td>128 KB</td>
</tr>
<tr>
<td>Clock Speed</td>
<td>16 MHz</td>
</tr>
</tbody>
</table>

5.2 Software

The web server is contained in the Arduino Mega program code. It utilizes the Ethernet library for connectivity as well as the SD library for accessing the SD card [13]. Once the program is initialized, it starts to monitor and process client connections continuously. The code for the web server can be found in Appendix B.

5.2.1 Initialization

The program first ensures that the html files and the password file can be found as the server cannot function without these files. The program then attempts to establish a connection to the campus network. Since the campus is using Dynamic Host Configuration Protocol (DHCP), the MAC address is the only information needed to connect. The Ethernet shield does not have a fixed MAC address. To establish a connection its MAC address must be set to one that is permitted to connect to the campus’s network. Finally, the program loads the password file and stores the password in memory for later use.

5.2.2 Polling

Polling method is used to listen to client connections. During each loop, the program tests whether or not there is a connection request. If a request is found, the program reads its header line and uses string processing methods on the line to find out the purpose of the request. The server only accepts certain requests and all other requests will be ignored. Descriptions of valid requests, their header lines and the corresponding server responses are listed in Table 5-2.
Table 5-2: Valid requests look up table.

<table>
<thead>
<tr>
<th>Header</th>
<th>Request Description</th>
<th>Server Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET / HTTP/1.1</td>
<td>A new client connects.</td>
<td>The pass.htm is sent to the client.</td>
</tr>
<tr>
<td>GET /password_xxxxx</td>
<td>The client submits a password of “xxxxx”.</td>
<td>If the password is correct, send client index.htm; if not, redirect client back to pass.htm.</td>
</tr>
<tr>
<td>GET /para</td>
<td>The client is requesting for temperature, humidity and light intensity values as well as control mode, window, fan and heat status.</td>
<td>An array of corresponding values is sent to the client.</td>
</tr>
<tr>
<td>GET /auto</td>
<td>The client is trying to set the control system to automatic mode.</td>
<td>Signal is sent to set slave Arduino to automatic mode.</td>
</tr>
<tr>
<td>GET /manual</td>
<td>The client is trying to set the control system to manual mode.</td>
<td>Signal is sent to set slave Arduino to manual mode.</td>
</tr>
<tr>
<td>GET /openWin</td>
<td>The client is requesting the window to be open.</td>
<td>Signal is sent to slave Arduino to open window.</td>
</tr>
<tr>
<td>GET /closeWin</td>
<td>The client is requesting the window to be closed.</td>
<td>Signal is sent to slave Arduino to close window.</td>
</tr>
<tr>
<td>GET /fanOn</td>
<td>The client is requesting the fan to be turned on.</td>
<td>Signal is sent to slave Arduino to turn on fan.</td>
</tr>
<tr>
<td>GET /fanOff</td>
<td>The client is requesting the fan to be turned off.</td>
<td>Signal is sent to slave Arduino to turn off fan.</td>
</tr>
<tr>
<td>GET /heatOn</td>
<td>The client is requesting the heat to be turned on.</td>
<td>Signal is sent to slave Arduino to turn on heat.</td>
</tr>
<tr>
<td>GET /heatOff</td>
<td>The client is requesting the heat to be turned off.</td>
<td>Signal is sent to slave Arduino to turn off heat.</td>
</tr>
</tbody>
</table>

The flow chart of the web server program is shown below in Figure 5-1. The left part represents the setup phase and the right part represents the polling phase.
Figure 5-1: Web server flow chart.
Chapter 6 Web Interface

6 Web Interface

The GUI is located on a web page hosted by the master Arduino which allows users to view and adjust the greenhouse environment on their browsers. The main advantage of using a web page based user interface is that it is compatible with most devices compared to an application based interface where different versions of the application must be created for different platforms (e.g. iOS, Android, etc.). Another advantage of using a web page based interface is that there is no installation required and the same piece of code performs the same function the same on different browsers and devices. The URL of the web interface is the IP address of the master Arduino. Upon accessing the web site the user is prompted to enter the correct password before they can access the interface page. The main web interface code can be found in Appendix C.

6.1 JavaScript Libraries

To achieve the goals of this project Basic HTML and JavaScript are not sufficient. In order to access higher functionality, additional JavaScript libraries are required. They can be fetched from their hosting server for free. JQuery and AccuWeather were the two additional libraries required. JQuery is the most widely used JavaScript library in the world [14]. It supports asynchronous JavaScript and XML (Ajax) which allows part of web page to be updated without having to refresh the whole page. This is essential to the project because it updates the monitoring data and command fields without refreshing the interface. JQuery is also has cross-platform functionality, so it is supported by both PCs and mobile phones. The AccuWeather widget library is required in order to use the weather widget from accuweather.com. The widget can be seen in Figure 6-4.
6.2 Login Page

Figure 6-1: The login page on different platforms. (a) Google Chrome running on a PC. (b) Safari running on an iPhone.

A login page is necessary security feature to ensure the interface can only be accessed by authorized persons. This is important because improper use of the interface (such as opening a window in the winter) could result in damage to the plants. When a user connects to the web server, a pop up window appears prompting the user to enter a password to continue. If the entered password is correct, the user will be redirected to the user interface. The password is stored inside “password.txt” file in the SD card on the server-side. The password can be set by the administrator by simply changing the “password.txt” file. The login page code can be found in Appendix D. The login page is shown on different interfaces above in Figure 6-1.
6.3 Interface Page

The interface page is where users can view and control the greenhouse environment. The page is programmed so that in every 5 seconds, the page sends a GET /para request to the web server, the web server responds by sending back a string containing the temperature, humidity and light intensity values as well as the control mode, window, fan and heat status. The interface page then extracts and displays the values. Buttons are used to input user commands including the control mode setting as well as the window, fan and heat controls. Each time a button is pressed, a block of program code is executed and a corresponding HTTP GET request is sent to the server. The request headers are shown in Table 5-2. The program flow can be visualized below in Figure 6-3.
At the bottom half of the interface page, a weather widget provided by accuweather.com is included. It would be convenient because when the user is using window control to adjust room temperature, the user might want to know the weather outside the greenhouse. This widget also shows the weather in the next two days and the next five hours which the user might be interested in. accuweather.com is one of the best websites that provides free reliable and accurate weather widget.
6.4 Data Logging

Every 5 seconds data is pulled from the sensor variables and converted to strings. These strings are concatenated into CSV format and written to a text file on the 8GB micro SD card. The user can then remove the SD card and easily transfer the data to be viewed in Excel. Every five seconds approximately 25 characters are written to the text file which is equivalent to 25 bytes of data. At this rate you could save 50 years of data before filling the SD card capacity.
Chapter 7

7 Design Changes

Throughout the course of the project there were many changes made to both the hardware and software in order to achieve a fully functional design. These changes were made in response to problems discovered in the design, prototyping, integration and testing phases.

7.1 Window Position

In the original design the window position was going to be determined by using sending the motor control signals for a period of time and knowing the rate at which the linear actuator opens and closes. This method was decided against as there was no form of feedback of the absolute window position to the microcontroller. This left open the possibility of the actuator extending too far and damaging the window.

7.1.1 Ultrasonic Sensor

To obtain accurate feedback on the position of the window an ultrasonic sensor that can be used to determine how far an object is away from the sensor was chosen. However, after bench testing it was determined that the ultrasonic sensor might not be accurate enough on its own to determine how far the window has opened. To ensure that the window wouldn’t open too far it was decided that an additional sensor to detect when the window had opened to a certain point to supplement the ultrasonic sensor which would be used to determine if the window is closed.

7.1.2 Reed Switch

To supplement the ultrasonic sensor a reed switch was chosen which sends a HIGH signal when the window has fully opened. After making changes to the motor used to open the window in the testing phase and receiving feedback from the advisors the combination of reed switch and ultrasonic sensor to detect the window position was decided against.
7.1.3 Contact Switches

In the final design two contact switches are used to determine if the window is fully open or fully closed. These switches were easily mounted near the rotating bar and tripped when a tab attached to that is attached to the existing manual window controls. This removed the need to have any sensors outside the building where they would be exposed to the elements. By using these switches the possibility in error of detecting the window position that could arise from using the other methods was greatly reduced. This reduced the risk of damaging the greenhouse windows by opening them too far.

7.2 Window Operation

One major design change made to the hardware in this project was changing the method used to open the greenhouse windows. Originally a linear actuator was chosen however after attempting to install it in the greenhouse and consulting with the project advisor it was decided that a rotational motor would be a better solution.

7.2.1 Rotational Motor

To take advantage of the existing window control hardware in the greenhouse a rotational motor that is compatible with the existing motor controller was installed. This “drop in” solution integrated easily with the original design and allowed product testing to continue without significant delay.

7.3 Web Server Connection

In the design phase, we planned to take advantage of the Wi-Fi connection in the greenhouse. However, after some research we found that the Wi-Fi connection security protocol is WPA2-Enterprise which is not supported by most of the Wi-Fi shields for Arduino. We had to switch to Ethernet for a reliable connection in the cost of limiting the physical location of the Arduino to be near the Ethernet port.
Chapter 8

8 Conclusion

The purpose of this project was to design and build a working prototype monitoring and control system for one room of the Buller greenhouse. This system allows a user to obtain temperature, humidity and light intensity readings as well as send temperature control commands remotely. This was achieved using Arduino microcontrollers and XBee wireless transceivers. The processes of hardware design, software design, integration, and testing of the prototype system have been described in this report. The project was broken down into four main sections: monitoring, control, wireless communication and the web server/GUI. Each individual section was unit tested before it was integrated into the final design. A fully integrated prototype was built and bench tested to ensure proper functionality before installation into the greenhouse. The system performed as designed and was able to remotely monitor and control the greenhouses temperature. Through the successful testing of the prototype it has been demonstrated that this system could be expanded to cover all rooms in the greenhouse.

8.1 Future Considerations

Although the design was successful there are improvements that could be made in future adaptations of this project. One of these improvements would be the addition of time stamping to the data logging system to make the data more useful. Another possible improvement would be the addition of humidity control devices to the greenhouse which currently do not exist. Finally lighting controls could be added to the system using the same by expanding the number of relays used.
References


# Appendix A - Slave Arduino Code

```c
/** Libraries */
#include "DHT.h" //DHT library
#include <OneWire.h> // ds18b20 library
#include <Wire.h>
#include <LCD.h>
#include <LiquidCrystal_I2C.h>

/** Control System Digital Pin Names */
#define dirControl 2   // window direction control
#define winControl 3   // window motor controller on/off
#define fanControl 4   // active high
#define heatControl 5   // active high
#define tempAlarm 7     // active high
#define winSensorOpenSwitch 12 // Low = Open
#define winSensorClosedSwitch 13// Low = Closed

/** Sensor Definitions and Setup */
#define DHTPIN 10 // using digital pin 10
#define DSPIN 9 // using digital pin 9
#define LIGHTPIN 2 // using analog pin 2
#define DHTTYPE DHT11 // select model of DHT

// LCD
#define I2C_ADDR 0x27 // <<- Add your address here.
#define Rs_pin 0
#define Rw_pin 1
#define En_pin 2
#define BACKLIGHT_PIN 3
#define D4_pin 4
#define D5_pin 5
#define D6_pin 6
```
#define D7_pin 7

LiquidCrystal_I2C lcd(I2C_ADDR, En_pin, Rw_pin, Rs_pin, D4_pin, D5_pin, D6_pin, D7_pin); //LCD setup
DHT dht(DHTPIN, DHTTYPE); //creates the dht object
OneWire ds(DSPIN); //onewire bus on PIN 9

//This section of code maps the temperature ranges for the switch case statements and temp alarm
int R0_LOW = -20; //Low value of temp range 0
int R0_HIGH = 15; //high value of temp range 0
int R1_LOW = 15; //Low value of temp range 1
int R1_HIGH = 25; //high value of temp range 1
int R2_LOW = 25; //Low value of temp range 2
int R2_HIGH = 30; //high value of temp range 2
int R3_LOW = 30; //Low value of temp range 3
int R3_HIGH = 35; //high value of temp range 3
int range; //variable used for switch case statements
float temp = 0; //temperature value read from sensor
int tempLow; //low threshold temperature for alarm
int tempHigh; //high threshold temperature for alarm

//control buttons from website
int manualOverride = LOW; //signal coming from web server when user wants to operate controls manually
int fanButton = HIGH; //web server fan control button
int heatButton = HIGH; //web server heat control button
int openWindowButton = LOW; //web server open window control button
int closeWindowButton = LOW; //web server close window control button

//variables
bool openning = false;
unsigned long timeToStopOpenning = 0;

int mode = 0;
int winCommand = 0;

int lightSensor = 0;
int type = 0;
int windows = 0;

boolean winSensorOpen = false; //Low when open
boolean winSensorClosed = false; //low when closed

const long interval = 2000; //update interval for LCD
unsigned long prevMillis = 0;

void setup() {
    Serial.begin(9600);
    // define inputs and outputs
    pinMode(winControl, OUTPUT);
    pinMode(dirControl, OUTPUT);
    pinMode(fanControl, OUTPUT);
    pinMode(heatControl, OUTPUT);
    pinMode(winSensorOpenSwitch, INPUT);
    pinMode(winSensorClosedSwitch, INPUT);
    digitalWrite(fanControl, HIGH);
    digitalWrite(heatControl, HIGH);
    pinMode(tempAlarm, OUTPUT);
    dht.begin();
    lcd.begin (20, 4); // <-- our LCD is a 20x4
    // LCD Backlight ON
    lcd.setBacklightPin(BACKLIGHT_PIN, POSITIVE);
    lcd.setBacklight(HIGH);
    lcd.home (); // go home on LCD
    lcd.print("Greenhouse");
}

void loop() {
    // your code here
}

/*********************END CONTROL SYSTEM SETUP*************************

//********************CONTROL SYSTEM FUNCTIONS*****
//********OPEN WINDOW FUNCTION**************
int openWindow(boolean winSensorOpen) {
    while (winSensorOpen == true) { //false when open
        digitalWrite(winControl, HIGH);
        digitalWrite(dirControl, LOW);   //Open
        break;
    }
    if (winSensorOpen == false) { //fully open
        digitalWrite(winControl, LOW);
    }
}
//********END OPEN WINDOW**************

//********CLOSE WINDOW FUNCTION**************
int closeWindow(boolean winSensorClosed) {
    while (winSensorClosed == HIGH) { //low when closed
        digitalWrite(winControl, HIGH);
        digitalWrite(dirControl, HIGH);   //Close
        Serial.println("Window Closing");
        break;
    }
    if (winSensorClosed == LOW) { //fully closed
        Serial.println("Window is closed");
        digitalWrite(winControl, LOW);
    }
}
//********END CLOSE WINDOW**************

//***********************AUTOMATIC CONTROL FUNCTION***************************
int autoControl(boolean winSensorOpen, boolean winSensorClosed) {
    if (temp < R0_HIGH) {
        range = 0;
    }
    else if (temp < R1_HIGH) {
        range = 1;
    }
    else if (temp < R2_HIGH) {

range = 2;
}
else if (temp < R3_HIGH) {
    range = 3;
}
switch (range) {
    case 0: //Too cold
        if (temp < tempLow) { //check if temperature is critically low
            digitalWrite(tempAlarm, HIGH);
        }
        else {
            digitalWrite(tempAlarm, LOW);
        }

        if (winSensorClosed == true) { //window is open
            closeWindow(winSensorClosed);
        }
        if (winSensorClosed == false) { //window is closed
            digitalWrite(winControl, LOW);
        }
        digitalWrite(fanControl, HIGH);
        digitalWrite(heatControl, LOW);
        break;

    case 1: //Just right
        if (winSensorClosed == true) { //window is open
            closeWindow(winSensorClosed);
        }
        if (winSensorClosed == false) { //window is closed
            digitalWrite(winControl, LOW);
        }
        digitalWrite(fanControl, HIGH);
        digitalWrite(heatControl, HIGH);
        break;
case 2: //too warm
    if (winSensorOpen == true) { //window is closed
        openWindow(winSensorOpen);
    }
    else if (winSensorOpen == false) { //fully open
        digitalWrite(winControl, LOW);
    }
    digitalWrite(fanControl, LOW);
    digitalWrite(heatControl, HIGH);
    break;

    case 3: //too hot
    if (winSensorOpen == true) { //window is closed
        openWindow(winSensorOpen);
    }
    else if (winSensorOpen == false) { //fully open
        digitalWrite(winControl, LOW);
    }
    digitalWrite(fanControl, LOW);
    digitalWrite(heatControl, HIGH);
    break;

} //**************MANUAL CONTROL FUNCTION**************
int manualControl(boolean winSensorOpen, boolean winSensorClosed) {
    if (heatButton == HIGH) {
        digitalWrite(heatControl, HIGH);
    }
    if (fanButton == HIGH) {
        digitalWrite(fanControl, HIGH);
    }
    if (openWindowButton == HIGH) {
        openWindow(winSensorOpen);
    }
if (closeWindowButton == HIGH) {
closeWindow(winSensorClosed);
}

//***********END MANUAL CONTROL***********************

//************DS18B20 Digital Temp Sensor************

//returns the temperature from one DS18S20 in DEG Celsius
float getTemp() {
byte data[12];
byte addr[8];
if ( !ds.search(addr)) {
//no more sensors on chain, reset search
  ds.reset_search();
  return -1000;
}
if ( OneWire::crc8( addr, 7) != addr[7]) {
//Serial.println("CRC is not valid!");
  return -1000;
}
if ( addr[0] != 0x10 && addr[0] != 0x28) {
//Serial.print("Device is not recognized");
  return -1000;
}
ds.reset();
ds.select(addr);
if ( addr[0] != 0x10 && addr[0] != 0x28) {
//Serial.print("Device is not recognized");
  return -1000;
}
ds.write(0x44, 1); // start conversion, with parasite power on at the end
for (int i = 0; i < 9; i++) { // we need 9 bytes
  data[i] = ds.read();
}
ds.reset_search();
byte MSB = data[1];
byte LSB = data[0];
float tempRead = ((MSB << 8) | LSB); //using two's compliment
float TemperatureSum = tempRead / 16;
return TemperatureSum;
}

//**************************END CONTROL SYSTEM FUNCTIONS**************************

void loop() {
  lightSensor = analogRead(LIGHTPIN);
  int hum = dht.readHumidity(); //reads humidity
  float dhtTemp = dht.readTemperature(); //reads temp in celsius
  float tempDS = getTemp();
  temp = tempDS;
  winSensorOpen = digitalRead(winSensorOpenSwitch);
  winSensorClosed = digitalRead(winSensorClosedSwitch);
  unsigned long curMillis = millis();
  Serial.println(lightSensor);
  Serial.println(lightSensor * 100 / 1023);
  int lightMapped = map(lightSensor, 0, 1023, 0, 100);
  Serial.println("101");
  Serial.println(tempDS);
  Serial.println("102");
  Serial.println(hum);
  Serial.println("103");
  Serial.println(lightMapped);
  Serial.println("104");
  if (winSensorOpen == true) {
    Serial.println(1); //window is open
  }
  else if (winSensorClosed == false) {
    Serial.println(2); //window is closed
  }
  else {
  
}
Serial.println(3); //window is moving
}
Serial.println("105");
if (mode == 1) {
    Serial.println(1);//manual mode selected
} else if (mode == 0) {
    Serial.println(0); //auto mode selected
}

//Controls from Xbee via Web server
if (Serial.available()) {
    type = Serial.parseInt();
    if (type == 104) { //Open Window Command from Xbee
        winCommand = 1;
        Serial.println("Window open");
    }
    if (type == 105) { //Close Window Command from Xbee
        winCommand = 0;
        Serial.println("Window close");
    }
    if (type == 106) {
        mode = 1; //Manual Mode
    }
    if (type == 107) {
        mode = 0; //Auto Mode
    }
}
if (mode == 1) {
    manualOverride = HIGH;
    Serial.println("Manual Mode");
}
if (mode == 0) {
    manualOverride = LOW;
    Serial.println("Auto Mode");
if (winCommand == 1) {
    openWindowButton = HIGH;  // Call the open function
    closeWindowButton = LOW;
}
if (winCommand == 0) {
    closeWindowButton = HIGH;  // Call the close function
    openWindowButton = LOW;
}

// LCD Printing
if (curMillis - prevMillis >= interval) {  // Updates every 2 seconds
    prevMillis = curMillis;
    lcd.setCursor(0, 1);  // Go to start of 2nd line
    lcd.print("Temp: ");
    lcd.print(temp);
    lcd.print(" C");
    lcd.setCursor(0, 2);  // Go to start of 4th line
    lcd.print("Humidity: ");
    lcd.print(hum);
    lcd.print(" %");
    lcd.setCursor(0, 3);
    lcd.print("Light: ");
    lcd.print(lightMapped);
    lcd.print(" %");
}

// Manual or Auto Mode
if (manualOverride == HIGH) {
    manualControl(winSensorOpen, winSensorClosed);
}
else if (manualOverride == LOW) {
    autoControl(winSensorOpen, winSensorClosed);
}
Appendix B - Master Arduino Code

```cpp
#include <SPI.h>
#include <Ethernet.h>
#include <SD.h>
#include <Client.h>

// size of buffer used to capture HTTP requests
#define REQ_BUF_SZ 25

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
//byte mac[] = {0x34, 0x17, 0xEB, 0xC7, 0xF5, 0x1F};
byte mac[] = {0xB8, 0xCA, 0x3A, 0xB4, 0xD7, 0x0A};
#define __USE_DHCP__

IPAddress ip(192, 168, 0, 107);
IPAddress gateway(192, 168, 0, 1);
IPAddress subnet(255, 255, 255, 0);
// fill in your Domain Name Server address here:
IPAddress myDns(8, 8, 8, 8); // google public dns

// Initialize the Ethernet server library
// with the IP address and port you want to use
// (port 80 is default for HTTP):
EthernetServer server(80);

File webFile;
char HTTP_req[REQ_BUF_SZ] = {0}; // buffered HTTP request stored as null terminated string
char req_index = 0; // index into HTTP_req buffer
float temp = 0;
float humi = 0;
int mode = 0; // 0 - auto, 1 - manual
int window = 0;
```

50
int fan = 0; //0 - off, 1 - on
int type;
int heat = 0; //0 - off, 1 - on
int winstatus = 0;
float light = 0;
char password[REQ_BUF_SZ - 14] = {0};

//Delay variables
unsigned long previousMillis = 0;
unsigned long readMillis = 0;
const long interval = 10000;

/********************************************
 * setup   *
********************************************/
void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  Serial1.begin(9600);

  // ******** initialize SD card ********
  Serial.println("Initializing SD card...");
  if (!SD.begin(4)) {
    Serial.println("ERROR - SD card initialization failed!");
    return; // init failed
  }
  Serial.println("SUCCESS - SD card initialized.");
  // check for index.htm file
  if (!SD.exists("pass.htm")) {
    Serial.println("ERROR - Can't find pass.htm file!");
    return; // can't find index file
  }
  if (!SD.exists("password.txt")) {
    Serial.println("ERROR - Can't find password.txt file!");
    return; // can't find index file
  }
if (!SD.exists("index.htm")) {
    Serial.println("ERROR - Can't find index.htm file!");
    return; // can't find index file
}
Serial.println("SUCCESS - Found index.htm file.");
Serial.println("SUCCESS - Found pass.htm file.");
Serial.println("SUCCESS - Found password.txt file.");

// ******** initialize the ethernet device ********
#if defined __USE_DHCP__
    Ethernet.begin(mac);
#else
    Ethernet.begin(mac, ip, myDns, gateway, subnet);
#endif

// start the Ethernet connection and the server:
server.begin();
Serial.print("server is at ");
Serial.println(Ethernet.localIP());
#if defined __USE_DHCP__
ip = Ethernet.localIP();
#endif

int i = 0;
File passFile = SD.open("password.txt");
while (passFile.available()) {
    password[i] = passFile.read();
i++;
}
webFile.close();
Serial.print("Password is: ");
Serial.println(password);
void loop() {
    // datalogging variables
    unsigned long currentMillis = millis(); // used for delays throughout the code
    int sensors[3]; // array that holds the sensor data received from the xbee
    String currentTime = "1/15/2016 10:51,;"; // temporary place holder for current time, will be later pulled from NTP

    if (Serial1.available())
    {
        type = (Serial1.parseInt());
        if (type == 101) {
            temp = (Serial1.parseFloat());
        }
        if (type == 102) {
            humi = (Serial1.parseFloat());
        }
        if (type == 103) {
            light = (Serial1.parseFloat());
        }
        if (type == 104) {
        }
        if (type == 105) {
        }
        if (winstatus == 1) {
            Serial1.println(104); // Open
        }
        if (winstatus == 0) {
            Serial1.println(105); // Close
        }
        if (mode == 1) {
            Serial1.println(106); // mode=1 is manual
        }
    }
}
if (mode == 0) {
    Serial1.println(107); // mode=0 is auto
}

// Put sensor data into an array for datalogging
sensors[0] = (int)light;
sensors[1] = (int)humi;
sensors[2] = (int)temp;

// make a string for assembling the data to log:
String dataString = "";
dataString += currentTime; // Time stamp the data

// Add sensor data to a string
dataString = sensorsToString(sensors, dataString);

// Call Write Fcn with a delay
if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    sdWrite(dataString);
}

// listen for incoming clients
EthernetClient client = server.available();
if (client) {
    // Serial.println("new client");
    // an http request ends with a blank line
    boolean currentLineIsBlank = true;
    while (client.connected()) {
        if (client.available()) { // client data available to read
            char c = client.read(); // read 1 byte (character) from client
            // buffer first part of HTTP request in HTTP_req array (string)
            // leave last element in array as 0 to null terminate string (REQ_BUF_SZ - 1)
if (req_index < (REQ_BUF_SZ - 1)) {
    HTTP_req[req_index] = c;  // save HTTP request character
    req_index++;
}
// print HTTP request character to serial monitor
//Serial.print(c);
// last line of client request is blank and ends with \n
// respond to client only after last line received
if (c == '\n' && currentLineIsBlank) {
    Serial.println(HTTP_req);
    // open requested web page file
    if (StrContains(HTTP_req, "GET / HTTP/1", 0)) {
        client.println("HTTP/1.1 200 OK");
        client.println("Content-Type: text/html");
        client.println("Connection: close");
        client.println();
        webFile = SD.open("pass.htm");  // open password web page file
    }
    else if (StrContains(HTTP_req, "GET /password", 0)) {
        if (StrContains(HTTP_req, password, 13)) {
            client.println("interface");
        }
        else {
            client.println("");
        }
    }
    else if (StrContains(HTTP_req, "GET /interface", 0)) {
        client.println("HTTP/1.1 200 OK");
        client.println("Content-Type: text/html");
        client.println("Connection: close");
        client.println();
        webFile = SD.open("index.htm");    // open web page file
    }
    else if (StrContains(HTTP_req, "GET /para", 0)) {
        if (!((temp < 100) && (temp > -50))) temp = -99;
if (!(humi < 100) && (humi > 0)) humi = -99;
if (!(light < 1000) && (light > 0)) light = -99;
sendHeader_text(client);

String paraString = String(temp) + " " + String(humi) + " " + String(light) + " " + String(mode) + " " + String(winstatus) + " " + String(fan) + " " + String(heat);

client.println(paraString);
}
else if (StrContains(HTTP_req, "GET /Auto", 0)) {
    mode = 0;
    Serial.println("Auto mode");
    if (client.connected())
    {
        sendHeader_text(client);
        client.println(mode);
    }
}
else if (StrContains(HTTP_req, "GET /Manual", 0)) {
    mode = 1;
    Serial.println("Manual mode");
    if (client.connected())
    {
        sendHeader_text(client);
        client.println(mode);
    }
}
else if (StrContains(HTTP_req, "GET /openWin", 0)) {
    winstatus = 1;
    Serial.println("Window open");
    if (client.connected())
    {
        sendHeader_text(client);
        client.println(window);
    }
}
else if (StrContains(HTTP_req, "GET /closeWin", 0)) {
    winstatus = 0;
}
Serial.println("Window close");
if (client.connected())
{
    sendHeader_text(client);
    client.println(window);
}
else if (StrContains(HTTP_req, "GET /fanOn", 0)) {
    if (client.connected())
    {
        sendHeader_text(client);
        client.println(fan);
    }
}
else if (StrContains(HTTP_req, "GET /fanOff", 0)) {
    if (client.connected())
    {
        sendHeader_text(client);
        client.println(fan);
    }
}
else if (StrContains(HTTP_req, "GET /heatOn", 0)) {
    if (client.connected())
    {
        sendHeader_text(client);
        client.println(heat);
    }
}
else if (StrContains(HTTP_req, "GET /heatOff", 0)) {
    if (client.connected())
    {
        sendHeader_text(client);
        client.println(heat);
    }
}
if (webFile) {
    while (webFile.available()) {
        client.write(webFile.read()); // send web page to client
    }
    webFile.close();
    client.stop();
}
// reset buffer index and all buffer elements to 0
req_index = 0;
StrClear(HTTP_req, REQ_BUF_SZ);
break;
}
// every line of text received from the client ends with \n
if (c == '\n') {
    // last character on line of received text
    // starting new line with next character read
    currentLineIsBlank = true;
}
else if (c != '\r') {
    // a text character was received from client
    currentLineIsBlank = false;
}

} // end if (client.available())

} // end while (client.connected())
// give the web browser time to receive the data
// close the connection:
client.stop();
//Serial.println("client disconnected");
}

void sendHeader_text(EthernetClient client) {
    client.println("HTTP/1.1 200 OK");
    client.println("text/plain; charset=us-ascii");
client.println("Connection: close");
client.println();
}

/****************************************************
* string processing functions                       *
****************************************************/

// sets every element of str to 0 (clears array)
void StrClear(char *str, char length)
{
    for (int i = 0; i < length; i++) {
        str[i] = 0;
    }
}

// searches for the string sfind in the string str
// returns 1 if string found
// returns 0 if string not found
char StrContains(char *str, char *sfind, int start)
{
    char found = 0;
    char index = start;
    char len;
    len = strlen(str);

    if (strlen(sfind) > len) {
        return 0;
    }

    while (index < len) {
        if (str[index] == sfind[found]) {
            found++;
            if (strlen(sfind) == found) {
                return 1;
            }
        }
    }
    else {

found = 0;
}
index++;
}

return 0;
}

//Write to SD Card
void sdWrite(String inputString) {
  File dataFile = SD.open("datalog.txt", FILE_WRITE);
  // Write Function
  // if the file is available, write to it:
  if (dataFile) {
    dataFile.println();
    dataFile.print(inputString);
    // this prints three sensor values separated by commas(###,###,###) (csv format)
    dataFile.close();
  }
  // if the file isn't open, pop up an error:
  else {
    Serial.println("error opening datalog.txt");
  }
}

// read 3 sensors and add it to dataString/
String sensorsToString(int sensorArray[], String dString) {
  for (int i = 0; i < 3; i++) {
    int sensor = sensorArray[i];
    dString += String(sensor);
    if (i < 2) {
      dString += ",";
    }
  }
  return dString;
}
Appendix C – Web Interface Code

<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Greenhouse Web Interface</title>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.2.0/jquery.min.js"></script>
  <script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
</head>
<body>
  <h1>Greenhouse Web Interface</h1>
  <p id="tempNum"> temperature = </p>
  <p id="humiNum"> humidity = </p>
  <p id="lightNum"> light intensity = </p>

  <p id="controlMode">Current Control Mode: Manual</p>
  <button type="button" onclick="controlModeButtonClicked()" id="controlModeButton">Switch to Auto</button>

  <p id="windowStat">window status: closed</p>
  <button type="button" onclick="windowButtonClicked()" id="windowButton">open window</button>

  <p id="fanStat">fan status: off</p>
  <button type="button" onclick="fanButtonClicked()" id="fanButton">turn on fan</button>

  <p id="heatStat">heat status: off</p>
  <button type="button" onclick="heatButtonClicked()" id="heatButton">turn on heat</button>

  <div id="chart_div"></div>

</body>
</html>
<script>
google.charts.load('current', {packages: ['corechart', 'line']});
var timeInterval = 5; //update parameters every x seconds
var time = 0;
var array = [];
var temp = 0.0;
var humi = 0;
var light = 0;

setInterval(function() {
    var setStatus = function(data) {
        if (data.localeCompare("updating") == 0) {
            document.getElementById("tempNum").innerHTML = "temperature = " + data;
            document.getElementById("humiNum").innerHTML = "humidity = " + data;
            document.getElementById("lightNum").innerHTML = "light intensity = " + data;
        }
        else {
            var para = data.split(" ");
            "temperature = " + para[0] + "ºC";
            if (parseInt(para[0]) != -99) document.getElementById("tempNum").innerHTML = "temperature = " + para[0] + "ºC";
            "humidity = " + para[1];
            if (parseInt(para[1]) != -99) document.getElementById("humiNum").innerHTML = "humidity = " + para[1];
            "light = " + para[2];
            if (parseInt(para[2]) != -99) document.getElementById("lightNum").innerHTML = "light intensity = " + para[2];
            if (parseInt(para[3]) == 0) {
                document.getElementById("controlMode").innerHTML = "Current Control Mode: Auto";
                document.getElementById("controlModeButton").firstChild.data = "Switch to Manual";
            }
            else if (parseInt(para[3]) == 1) {
                document.getElementById("controlMode").innerHTML = "Current Control Mode: Manual";
                document.getElementById("controlModeButton").firstChild.data = "Switch to Auto";
            }
            if (parseInt(para[4]) == 0) {
                document.getElementById("windowStat").innerHTML = "window status: closed";
            }
        }
    }
    setStatus(data);
}, timeInterval);
</script>
document.getElementById("windowButton").firstChild.data = "open window";
}
else if (parseInt(para[4]) == 1) {
    document.getElementById("windowStat").innerHTML = "window status:open";
    document.getElementById("windowButton").firstChild.data = "close window";
}
if (parseInt(para[5]) == 0) {
    document.getElementById("fanStat").innerHTML = "fan status:off";
    document.getElementById("fanButton").firstChild.data = "turn on fan";
}
else if (parseInt(para[5]) == 1) {
    document.getElementById("fanStat").innerHTML = "fan status:on";
    document.getElementById("fanButton").firstChild.data = "turn off fan";
}
if (parseInt(para[6]) == 0) {
    document.getElementById("heatStat").innerHTML = "heat status:off";
    document.getElementById("heatButton").firstChild.data = "turn on heat";
}
else if (parseInt(para[6]) == 1) {
    document.getElementById("heatStat").innerHTML = "heat status:on";
    document.getElementById("heatButton").firstChild.data = "turn off heat";
}
if(!isNaN(parseFloat(para[0]))) temp = parseFloat(para[0]);
array.push([time, temp]);
if(array.length > 20)
    array.shift();
drawBasic(array);
time += timeInterval;
}
$('#my_div').sendRq("para", setStatus);
}, timeInterval*1000);

(function( $( ){

$.fn.sendRq = function(rq, func) {

63
$.ajax(rq).done(function(data) { func(data); }).fail(function() { func("updating"); });

return this;

})(jQuery);

function reservedForLater(data) {
}

function controlModeButtonClicked() {

    if (document.getElementById("controlModeButton").firstChild.data.localeCompare("Switch to Auto") == 0) {

        $('#my_div').sendRq("Auto", reservedForLater(1));
    }
    else {

        $('#my_div').sendRq("Manual", reservedForLater(1));
    }

}

function windowButtonClicked() {

    if (document.getElementById("windowButton").firstChild.data.localeCompare("press to open") == 0) {

        $('#my_div').sendRq("windowOpen", reservedForLater(1));
    }
    else {

        $('#my_div').sendRq("windowClose", reservedForLater(1));
    }

}

function fanButtonClicked() {

    if (document.getElementById("fanButton").firstChild.data.localeCompare("turn on") == 0) {

        $('#my_div').sendRq("fanOn", reservedForLater(1));
    }
    else {

        $('#my_div').sendRq("fanOff", reservedForLater(1));
    }

}

function heatButtonClicked() {

    if (document.getElementById("heatButton").firstChild.data.localeCompare("turn on") == 0) {

        $('#my_div').sendRq("heatOn", reservedForLater(1));
    }
else {
    $("#my_div").sendRq("heatOff", reservedForLater(1));
}

function drawBasic() {

    var data = new google.visualization.DataTable();
    data.addColumn('number', 'seconds');
    data.addColumn('number', '°C');
    data.addRows(array);

    var options = {
        hAxis: {
            title: 'Time'
        },
        vAxis: {
            title: 'Temperature'
        }
    };

    var chart = new google.visualization.LineChart(document.getElementById('chart_div'));
    chart.draw(data, options);
}
Appendix D – Website Login Code

<!DOCTYPE html>
<html>
<body onload="passPrompt()">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.2.0/jquery.min.js"></script>
<link rel="stylesheet" href="https://ajax.googleapis.com/ajax/libs/jquerymobile/1.4.5/jquery.mobile.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquerymobile/1.4.5/jquery.mobile.min.js"></script>
<script>
function passPrompt() {
    var pass = prompt("To access greenhouse web interface, please enter password and press OK.");
    if (pass != null) {
        $('#my_div').sendRq("password_" + pass);
    }
}
(function( $ ){
    $.fn.sendRq = function(rq) {
        $.ajax(rq).done(function(data){location.replace(data);});
        return this;
    }
})( jQuery );
</script>
</body>
</html>
Appendix E - XBee Parameter Configuration

XBee on the master controller:

**Function set:** ZigBee Coordinator AT.

**ID PAN ID:** 1234

**DH** Destination Address High: 0

**DL** Destination Address Low: FFFF (Broadcast address)

XBee on the slave controller:

**Function set:** ZigBee Router AT.

**ID PAN ID:** 1234

**JV** Channel Verification: Enable 1

**DH** Destination Address High: 13A200

**DL** Destination Address Low: 40E79F0
## Appendix F - Budget

<table>
<thead>
<tr>
<th>Component</th>
<th>Supplier</th>
<th>Quantity</th>
<th>Cost</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Monitoring System</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arduino UNO R3</td>
<td>Supplied by team</td>
<td>4</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>Temperature Sensor</td>
<td>Robotshop</td>
<td>1</td>
<td>$10.24</td>
<td>$10.24</td>
</tr>
<tr>
<td>Humidity Sensor</td>
<td>Robotshop</td>
<td>1</td>
<td>$6.67</td>
<td>$6.67</td>
</tr>
<tr>
<td>Analog Light Sensor</td>
<td>Robotshop</td>
<td>1</td>
<td>$4.79</td>
<td>$4.79</td>
</tr>
<tr>
<td>9V 1A Power Adaptor</td>
<td>Supplied by team</td>
<td>1</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>LCD</td>
<td>Supplied by team</td>
<td>1</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td><strong>Control System</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear Actuator</td>
<td>Princess Auto</td>
<td>1</td>
<td>$74.99</td>
<td>$79.99</td>
</tr>
<tr>
<td>Mounting Bracket</td>
<td>Princess Auto</td>
<td>1</td>
<td>$19.99</td>
<td>$35.75</td>
</tr>
<tr>
<td>Rotational Motor</td>
<td>Bob McLeod</td>
<td>1</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>Feedback Switches</td>
<td>ECE Tech Shop</td>
<td>2</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>Enclosures</td>
<td>Supplied by team</td>
<td>2</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>12V 5A Power Adaptor</td>
<td>Robotshop</td>
<td>1</td>
<td>$31.25</td>
<td>$31.25</td>
</tr>
<tr>
<td><strong>Wireless Communication</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XBee S2</td>
<td>Robotshop</td>
<td>2</td>
<td>$29.42</td>
<td>$58.84</td>
</tr>
<tr>
<td>XBee Arduino Shield</td>
<td>Robotshop</td>
<td>2</td>
<td>$7.40</td>
<td>$14.80</td>
</tr>
<tr>
<td><strong>Web Server</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arduino Mega</td>
<td>Supplied by team</td>
<td>1</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>Ethernet Shield</td>
<td>Robotshop</td>
<td>1</td>
<td>$18.70</td>
<td>$18.70</td>
</tr>
<tr>
<td>Wifi Shield</td>
<td>Robotshop</td>
<td>1</td>
<td>$51.22</td>
<td>$51.22</td>
</tr>
<tr>
<td><strong>Miscellaneous</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wires</td>
<td>ECE Tech Shop</td>
<td>1</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>Circuit Components</td>
<td>ECE Tech Shop</td>
<td>1</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>Prototyping Board</td>
<td>ECE Tech Shop</td>
<td>2</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
</tbody>
</table>

Subtotal: $312.25  
Shipping, Duty, and Taxes: $34.78  
Total: $347.03