

Smart Window Blinds
EE4830 Senior Design II
Aya Mistica & Rui Chen
Steven F. Barrett, Ph.D., P.E.
Electrical and Computer Engineering Department
College of Engineering
University of Wyoming
Laramie, WY 82071
amistic1@uwyo.edu & rchen2@uwyo.edu

Abstract

According to a survey conducted by Pew Research Center in January 2018, over 77% of American adults own or use a smartphone [1]. Due to the growth in smartphone users, wireless technologies have become more and more integrated into everyday objects and given rise to the Internet of Things (IoT) [2]. By 2025, it is estimated that there will be more than 55 billion IoT devices.

This project aims to create and design an independent IoT system allowing users to open and close their household vertical blinds with just a simple tap on their personal mobile phones or tablets. The project requires three parts involving the software, hardware, and mechanical developments. The software aspect involves a web application which allows the user to open or close the blinds as they wish. The web application communicates to the hardware unit via WiFi where the user data is received by the System On Chip (SoC), ESP8266. The ESP8266 can then send out PWM signals to the stepper motor which along with the mechanical design is able to physically move the window blinds. The web application is implemented using HTML and CSS styling uploaded to the ESP8266 web server. The ESP8266 is programmed using the Arduino IDE, and the mechanical design independently sketched and designed using Tinkercad.

Keywords

Smart Window Blinds, Wireless Home System, Internet of Things, IoT, Wireless Window Blinds, Smart Home System, ESP8266

Overview

Project Scope

The Smart Window Blinds project aims to create a self-contained system which will allow users full automated control of their household vertical blinds through a smartphone or tablet device. The system consists of two main components: a compact attachable unit that

provides motorized control to open and close the window blinds as well as a web application which will connect to the attachable unit over Wifi. For design and implementation purposes, the project is divided into three technical parts: software, hardware, and mechanical.

The objective of the software portion of the project is to create a simple web application which will allow the user to communicate to our WiFi module, the ESP8266. It should allow the user to choose what type movement they desire for the vertical blinds and successfully send the user data to the ESP8266. Upon choosing an action, the web application should also reflect what button was previously pressed and indicate a status for the blinds.

The hardware unit consists of the ESP8266, stepper motor driver and other electrical components to ensure proper operation and interfacing between the mobile device and stepper motor. Within the hardware unit, the ESP8266 needs connect to the mobile device via WiFi and properly decode the transmitted user data. Based on the user data, the ESP8266 should output a PWM signal to the proper output pins which are connected to the stepper motor driver. Upon sending control signals to the stepper motor driver, the stepper motor should activate.

The mechanical system will be at the heart of physically moving the window blinds. It will include a stepper motor, a block attachment between the stepper motor and blinds as well as a twist bar. The block attachment will allow the stepper motor to be mounted on the side of the window blinds while the twist bar models a small rod which fits the stepper motor on one end and the turning knob of the window blinds on the other.

Project Goals

For the software portion of our project, our goal is to implement a straightforward, easy-to-use and simple web application. There should be two buttons that users can choose from to move the window blinds in their desired position. Upon pressing the button, a status should be displayed on the web application indicating that the user data was successfully sent. The purpose of displaying a status when a button was pressed, besides for testing purposes, is also to communicate to the user that their action was successful.

Our project also aims to fulfill the needs of consumers to have a reliable, easy-to-use, minimum upkeep but affordable product for their smart home goals. Compared to other wireless window blinds, our design will offer the same functionalities, but at a more cost-effective price. With the installable module, our users will not need to replace their current window blinds, unlike other products that require their customers to purchase a whole new set of blinds. The attachable module will be designed for quick and easy installations, keeping in mind that in most cases, buyers have multiple blinds in their homes. The product will be sleek and compact in design, well-hidden to preserve the aesthetics of the home or business. In addition, users have their smartphone or tablet device to automatically control their blinds unlike using a remote control which can be easily lost or damaged. We also aim for our IoT system to be low-maintenance, only requiring a 9V battery supply. The battery should last at least 6 months so that the users will not be inconvenienced by frequent replacement of batteries.

Background

The Smart Window Blinds systems falls under the category of Internet of Things devices. Internet of Things can be summarized as integrating common objects like mobile phones, home appliances, furniture, window blinds, etc. with sensors or other electrical components so that these objects can communicate with each other [3]. Our project implements the communication between a mere vertical window blinds and a mobile device. The impact of IoT devices is great especially for individuals in their everyday life. It allows for the creation of “smart” objects within the home and in industry.

ESP8266 [4] is a self-contained SOC with integrated TCP/IP protocol stack that can give any microcontroller access to our WIFI network. It is capable of either hosting an application or offloading all WIFI networking functions from another application processor. ESP8266 does not require for high voltage, the voltage supply for VCC is 3.3V. The standby current is around 0.9 uA and the running current in average is 80mA. It contains two input and output pins so that it can send out the signals when it communicate the software module. The ESP8266 module is an extremely cost effective board with a huge, and ever growing, community.

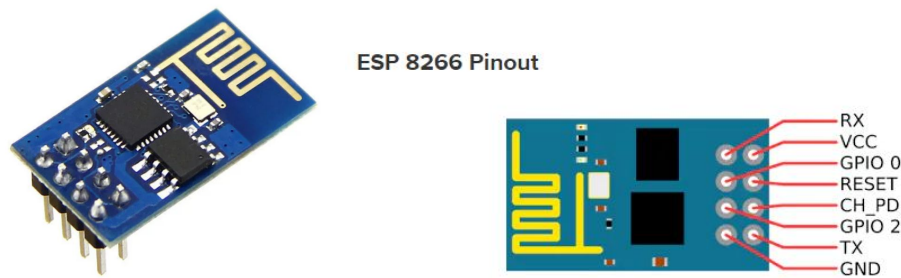


Figure 1. ESP8266 WIFI Module.

Technical Project Description

I. Functional Description

The block diagram in Figure 2 shows all the modules in the Smart Window Blinds project. The inputs of the system is the desired window blinds action input data from user to web application. The web application will develop the P2P WIFI protocol communication with WIFI module, the ESP8266. Then the ESP8266 can send out PWM signals to the stepper motor which will physically move the window blinds. Also the system includes a 9V Battery supply, voltage regulators and other circuitry components. The voltage regulators will reduce the 9V to 3.3V for ESP8266 and 5V for motor driver. The Table 1 - 3 below described in detail based on specific inputs, outputs and functionalities for each module.

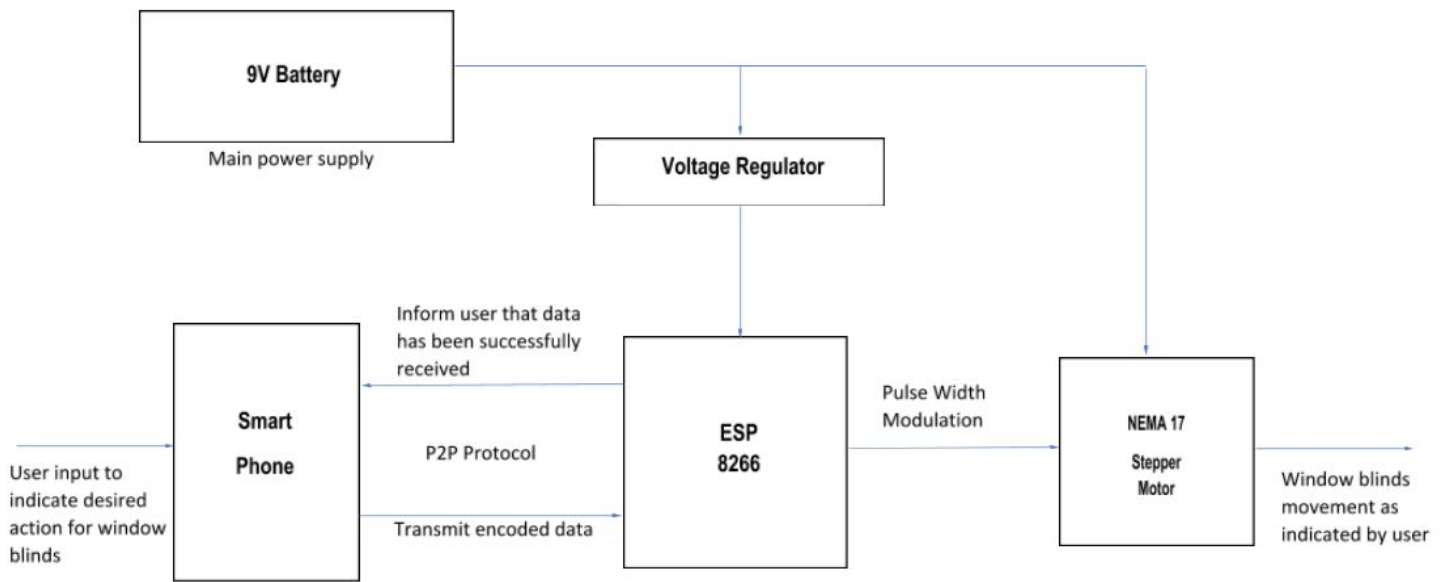


Figure 2. Block Diagram Smart Window Blinds

Table 1. Smartphone Module

Module	Smartphone
Inputs	User interacts with mobile application to indicate desired position of window blinds. When ESP8266 receives encoded data, send success message back to mobile application to inform user.
Outputs	Transmit encoded data: using P2P Wifi Protocol, send user data to ESP 8266
Functionality	Exchange data wirelessly between Wifi module and user

Table 2. ESP8266 Module

Module	ESP8266
Inputs	Transmit encoded data: Using Wifi, decode received user input from smartphone

	Power supply: 3.3V from voltage regulator
Outputs	When ESP8266 receives encoded data, send success message back to application to inform user. Send a pulse width modulation to DC motor determined by the decoded user data.
Functionality	Receive and decode user data to transmit appropriate pulse width modulation to DC motor.

Table 3. Stepper Motor

Module	NEMA 17
Inputs	Pulse width modulation from controller: moves the motor a specific angle based on the duty cycle of the signal. Power supply: 5V from voltage regulator
Outputs	Moves the window blinds as the user has requested
Functionality	Physically moves the window blinds based on user input

II. Software Module

To implement our web application, we used HTML and CSS styling. Both of these together allowed us to display our project name as a header so that users will know that they are in the correct webpage. In addition, we created two buttons in the web application (shown in Figure 3): left and right. These two buttons can be pressed to indicate the desired direction for the vertical window blinds to move. After pressing a button, the page reloads and displays the web application again but with a status as shown in Figure 4.

Since the ESP8266 can be used as a web server, we opted to put all of our HTML and CSS code in the ESP8266's web server. Not only was this free, but provided an easier way for us to manage the web application and ESP8266 communication. We found extensive tutorials of how to implement this in [5]. Using this resource, we learned to put our HTML and CSS code into header files to be used in the Arduino code which is then uploaded onto the ESP8266. Our

handleOpen() and handleClose() functions call the ESP8266 server to send our HTML and CSS code so that the web application can be displayed to client.

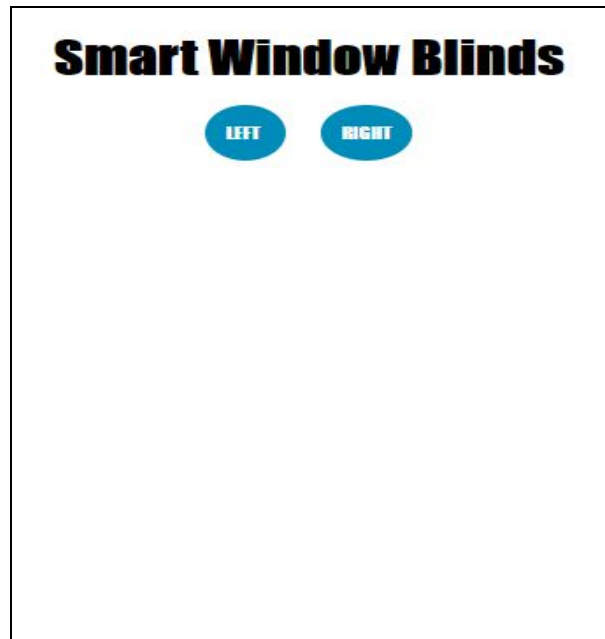


Figure 3. Smart Window Blinds Web Application

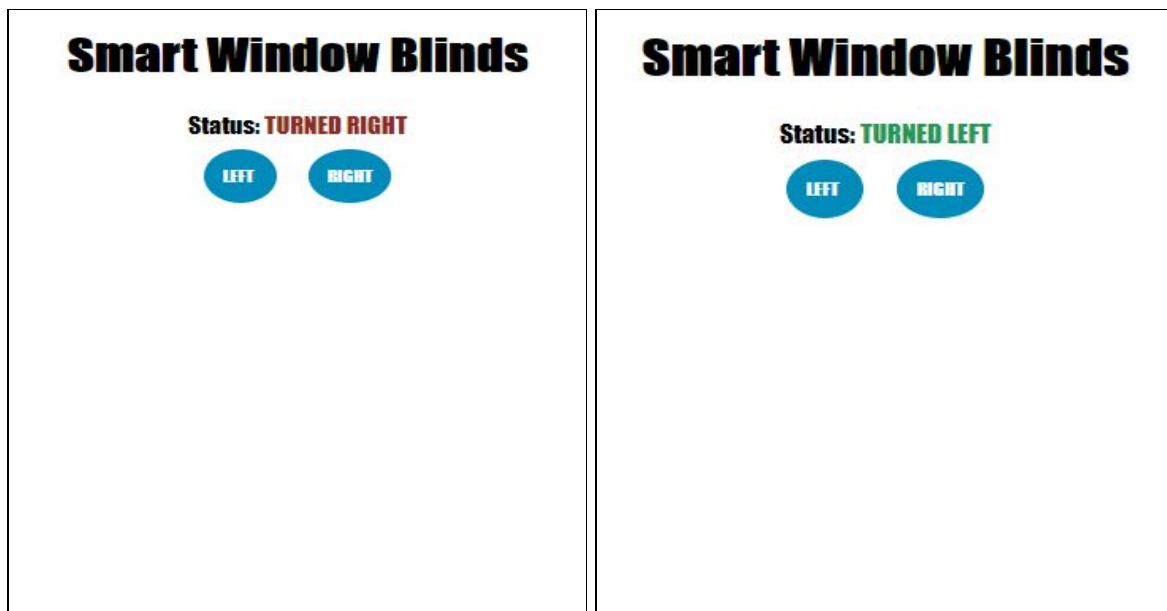


Figure 4. Web App with Status

In order to access the web application, we used the WifiManager library [6] from Github and included this in our Arduino libraries. The purpose of this was to prevent needing to hardcode WiFi SSID and password to the Arduino code when network credentials change. WiFi Manager provided a user interface to enter WiFi information every time the ESP8266 is powered on, or if it does not see any saved WiFi SSID and password (shown in Figure 5).

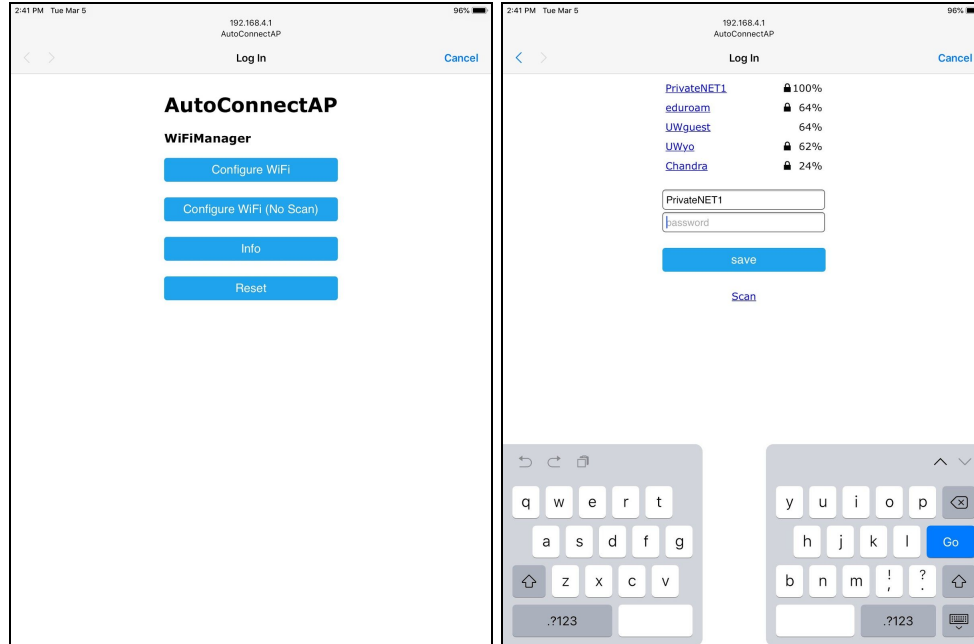


Figure 5. WiFi Manager

Pressing the “Configure WiFi” under AutoConnectAP would then show all of the available networks that can be connected to the ESP8266. Once the mobile device and ESP8266 are in the same network, the IP address of the ESP8266 can be entered into a browser which will direct the user to the web application. Initially, we knew the IP address of the ESP8266 from the Serial Monitor of the Arduino software. Later on, we found out that it is displayed on our laptop when we use it as a mobile hotspot.

III. Hardware Module

On the hardware side, we used the ESP8266 as a microcontroller unit. Using the ESP8266 by itself allowed us access to full WiFi connectivity, efficient reception of data from the mobile application as well as the capability of sending signals to our stepper motor driver, the DRV8834 motor driver carrier [7]. When a button is pressed on the web application, the ESP8266 receives this data and acts accordingly. We have defined two functions when a button is pressed in the web application: `handleOpen()` and `handleClose()`. The names reflect the initial names of the buttons when our buttons used to be “Open” and “Close”. Now, the buttons are named “Left” and “Right” to better convey which direction the blinds will move. If the “Left” or “Right” button is pressed, the ESP8266 sends a HIGH signal to the SLP pin of the motor driver to enable it. An `analogWrite()` is then sent to the GPIO0 of the ESP8266 to send a PWM signal with 50% duty cycle. This signal is then transmitted to the motor driver STEP input pin. This PWM signal corresponds to how fast the stepper motor should turn. Another signal is also sent from the ESP8266 which is just a simple HIGH or LOW signal to the DIR pin of the stepper motor driver. This signal indicates which direction the stepper motor will turn, whether clockwise or counterclockwise. After a 2.2 second software delay, the SLP pin is turned back to

LOW and the ESP8266 stops sending PWM signals to indicate that the stepper motor should now be stationary. Within our hardware module are various voltage regulators, a 3.3 V [8] and 5V [9] regulator. The 3.3V is provided for the ESP8266 and the 5V for the stepper motor driver. In addition, we have added some bypass capacitors to dampen any noise in the signals. These components can all be seen on our PCB board in Figure 6.

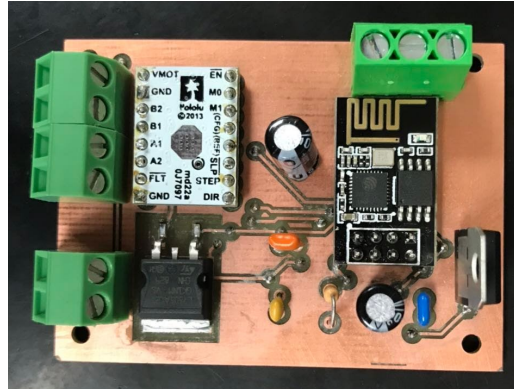


Figure 6. Hardware PCB Board

IV. Mechanical Module

For the mechanical module, we used a bipolar stepper motor, NEMA 17. It holds a high torque with 13 N-cm and takes 200 steps/rev. It draws 1A current per phase, which allows the stepper motor be able to drive it with the output current up to 2A per coil in 5V voltage supply. We chose this stepper motor on the premise of examining the motion involved in opening and closing a vertical window blind. A manual window blind has a rotation wand which rotates a 1.8cm diameter knob at the top of the blind. To open and close, it takes 5 full turns of this knob. Since we did not have a tool to measure the torque, we first estimated that a stepper motor with 6 N-cm torque is enough for our mechanical system. However, when we test our system with this stepper motor, it is not able to move the window blinds, then we changed to use the stepper motor with higher torque. This stepper motor has 4 input wires, we connected those 4 input to the 4 output pin on DRV8834 stepper motor driver carrier in a particular order, so that the stepper motor can send out the PWM signal and the current to drive the stepper motor to physically move the window blinds without vibrations.

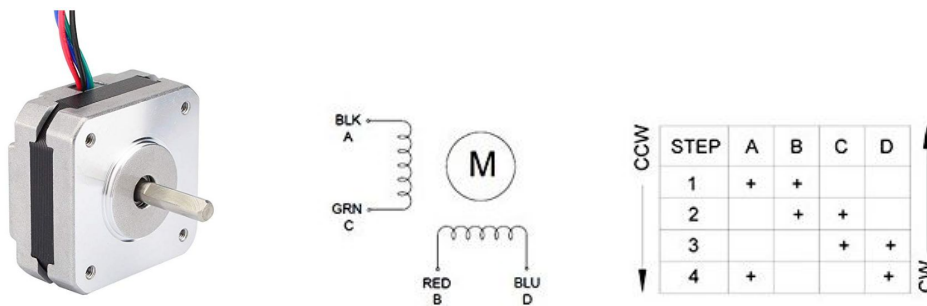


Figure 7. NEMA 17 bipolar stepper motor

Our Mechanical design consists two part: Twist Bar and Motor-Blind Attachment, shown in Figure 8 and Figure 9. The twist bar a small rotating fixture with different shapes on each side. This part will attach the metal rod of the stepper motor and the window blind. Then the window blind is able to move as the motor is turned on. The motor-blind attachment is a square block that attaches to stepper motor and window blinds, so that the stepper motor can be mounted on the side of window blinds. All the parts are designed by using Tinkercad and 3D printed by University of Wyoming IT Department.



Figure 8. Twist Bar

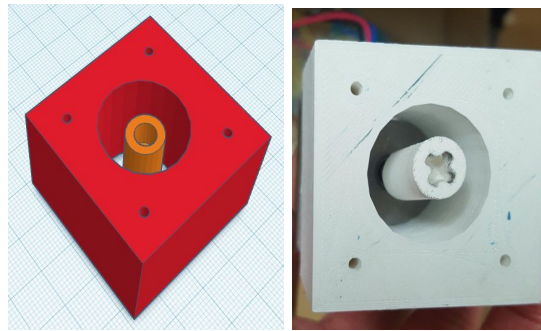
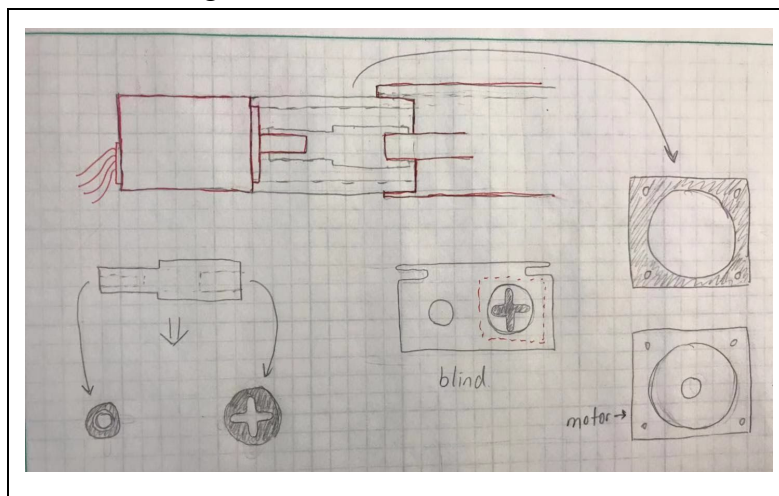


Figure 9. Motor-Blind Attachment



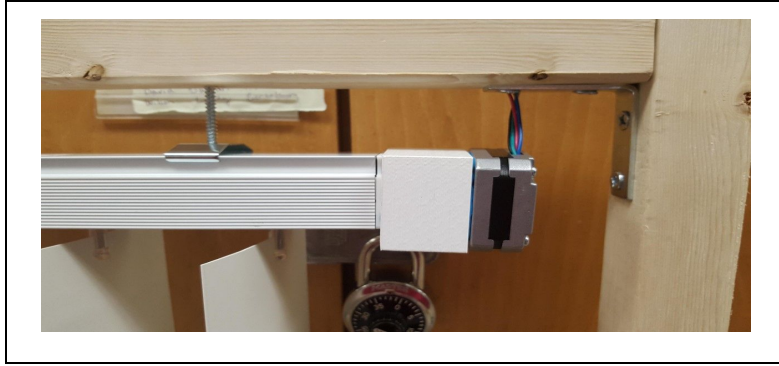


Figure 10. Mechanical Design

Testing

I. Software Testing

Initial testing for the web application heavily involved the Serial Monitor of the Arduino. We added print statements in certain functions within our Arduino code to make sure that when a button on the web application is pressed, we would be able to see that these functions were properly executing. In addition to this, if these print statements (shown in Figure 11) are displayed on the Serial Monitor, we can safely assume that the web application and the ESP8266 have established communication with each other. In addition, we also tested the WiFi Manager by entering the current network credentials we were using and seeing that the ESP8266 successfully connects to it which is also confirmed via the Serial Monitor.

One thing to note during this testing progress was when we found out that the ESP8266 cannot be connected to WiFi transmitting in the 5GHz frequency band. It can only connect to the 2.4GHz frequency band. We discovered this because when we would test our system in the lab, sometimes our mobile hotspot would switch frequency bands. When this happens, and the WiFi network is in the 5GHz frequency band, the WiFi Manager--which generally shows up in the Network list of the mobile device--is not listed. Once we switch our WiFi network back to 2.4GHz communication band, the WiFi Manager can be found again.

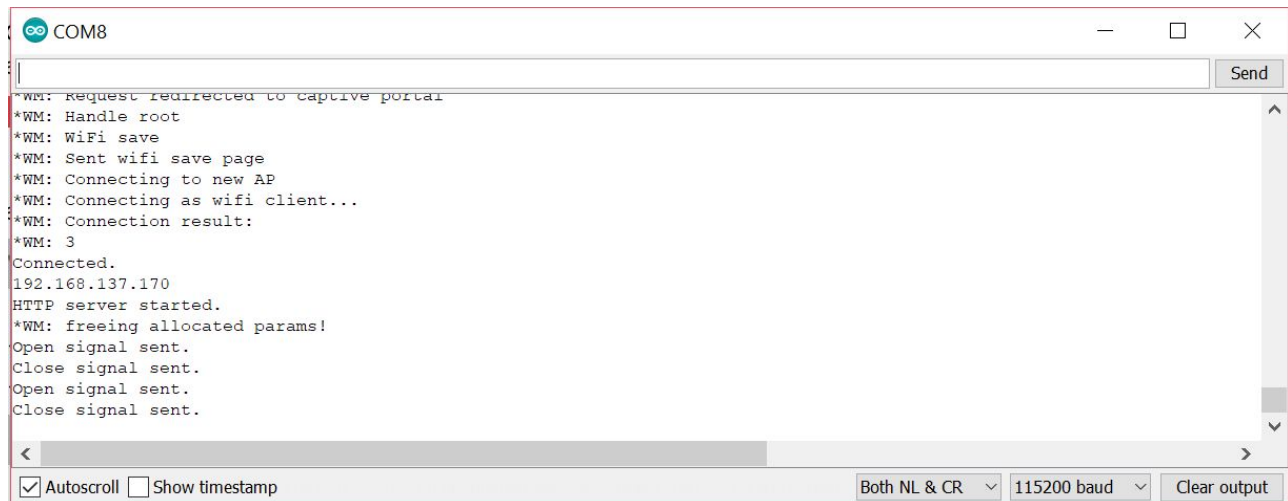


Figure 11. Serial Monitor Print Statements for Testing

II. Hardware Testing

Our hardware module was tested on a breadboard with a 9V power supply. We first started with connecting the output of voltage regulators to multimeter to check if the voltage was reduced from 9V to 3.3V and 5V. We secondly tested if the programmed WIFI module, ESP8266, is able to communicate with web application on the breadboard. During testing, the output pin of the ESP8266 was connected to the oscilloscope. When we press the open or close button on web application, the LED of ESP8266 turned on and an output signal was shown on the oscilloscope, which means the programmed ESP8266 is communicating with our web application. Then we tested the hardware module with our stepper motor driver. However, since we started with a motor driver chip without the heat sink and adapter, so the chip was broken by high temperature. Then we changed to use DRV8834 motor driver carrier with heat sink and adapter. We connected the input and output pins of motor driver carrier to the oscilloscope to check if there is any PWM signals go into and out from the motor driver carrier. We set 50% PWM signal since it is about the average speed of signal sending. The figure 13 shows the output signal of the motor driver carrier with 50% PWM.

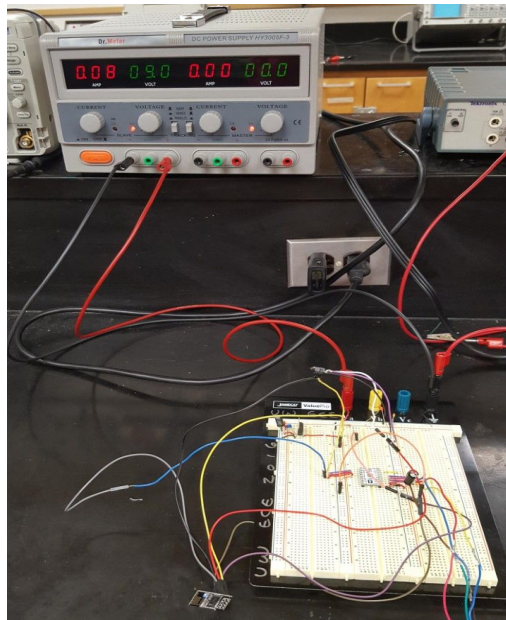


Figure 12. Hardware Testing.

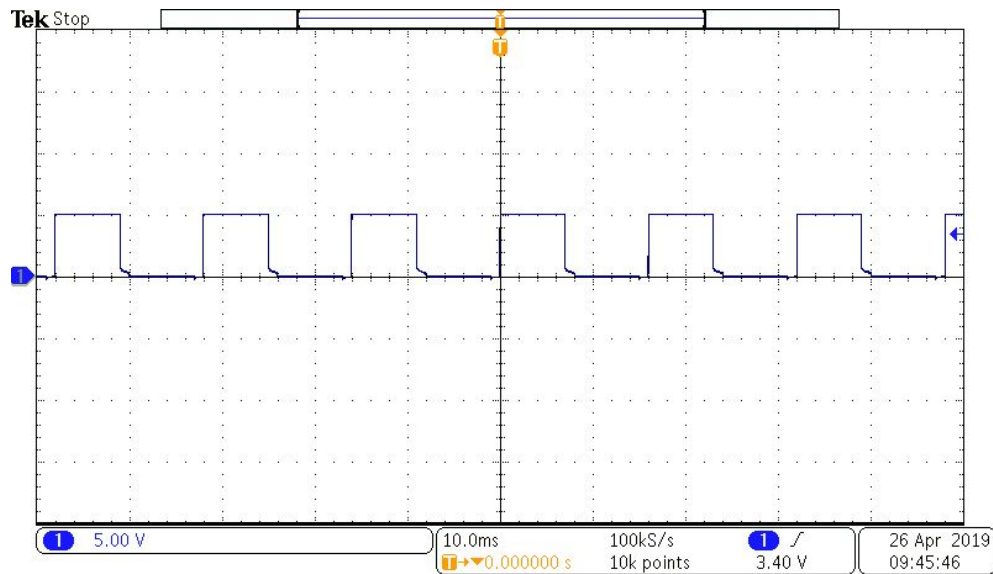


Figure 13. 50% PWM output signal of DRV8834 motor driver carrier

III. Overall Testing

The entire system was tested by connect the NEMA 17 stepper motor to the hardware system. At the initial testing, there were some vibrations occurred, and then we realized that it is because we randomly connected the four wires of stepper motor with the four output pin of the stepper motor driver carrier. To fix the vibrations, we followed the wiring diagram of DRV8834 stepper motor driver carrier (shown in Figure 14.) to rewire the stepper motor to the motor driver. We then mounted the stepper motor to the window blinds for the final testing. To prevent overturning the window blinds in one direction, we tested different time delay to keep the motor on by change the delay in the code. As result, we figured out that 2.2 seconds delay is the best time before turning off control signals and putting motor driver back to sleep mode. So that there is no overturning when the window blinds is turned on and off in one direction.

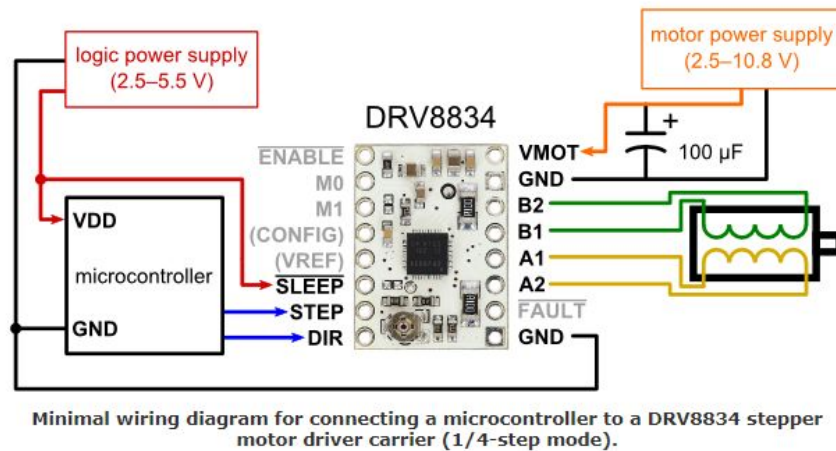


Figure 14. Wiring Diagram for DRV8834

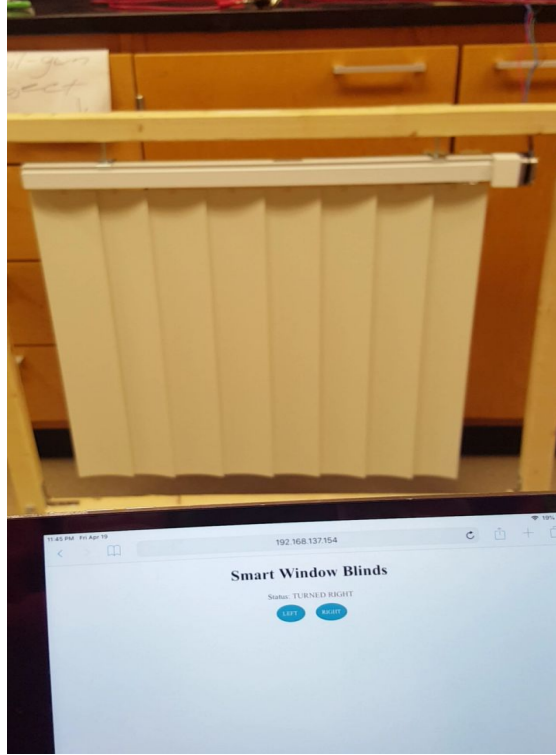


Figure 15. Overall Testing

Packaging

Our packaging contains a battery compartment and rocket switch placement. The battery compartment will be attached on the top of the packaging box, so that there is enough space underneath the battery compartment that allow us to mount the PCB board under the battery compartment. A 9V battery case will be place in the battery compartment allowing the users to replace the 9V battery if needed. The wires of the battery case will be connected to the PCB board by passing through the hole in the back of the battery compartment. The stepper motor will be mounted on the back side of the packaging box. All the wires of the stepper motor will pass through the hole in the back of the packaging box to connect to the motor driver on the PCB board. After we mounted all the components, a lid will be screwed on the packaging box. All parts are designed using Tinkercad and 3D printed by University of Wyoming IT Department.



Figure 16. 9V battery case.

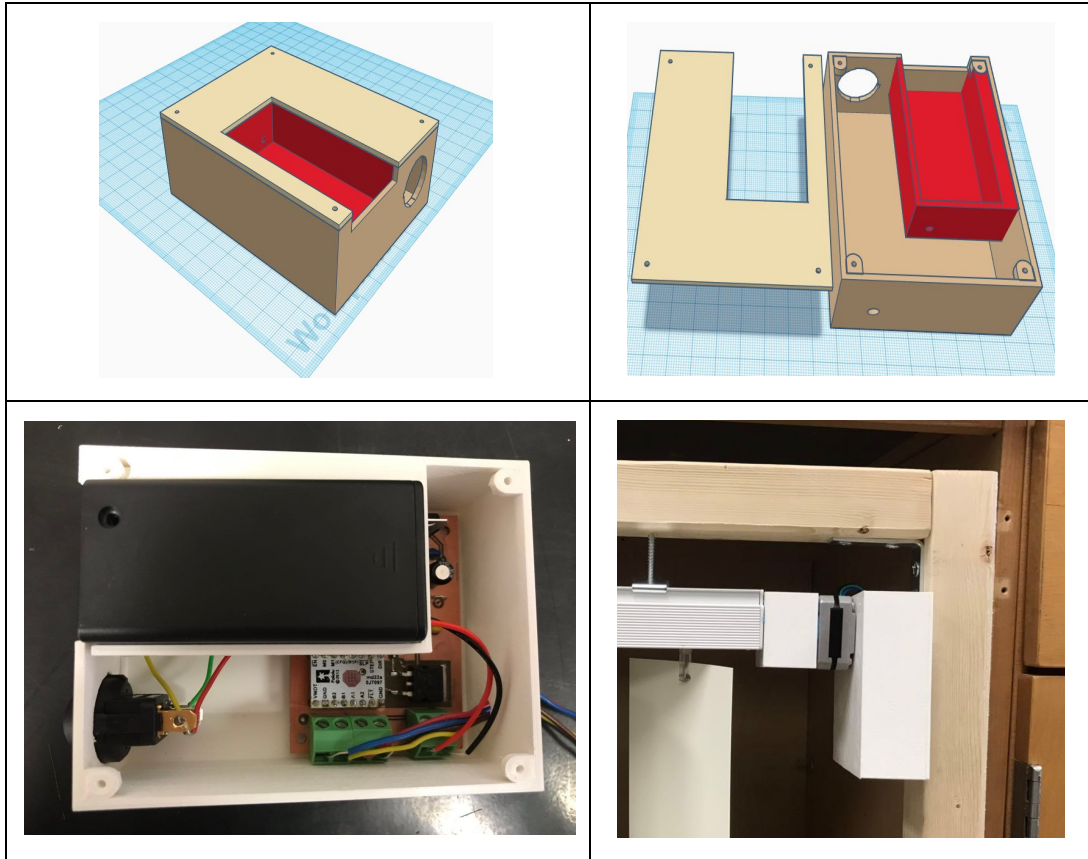


Figure 17. Packaging

Other Consideration

I. Project Cost

Table 4. Project Costs

Item	Quantity	Cost	Total
ESP 8266-01	3	\$6.95	\$20.85
USB to TTL Adapter (CP2102)	2	\$9.00	\$18.00
9V Battery	1 (2-pack)	\$7.80	\$7.80
Voltage Regulator (3.3V)	3	\$0.54	\$1.62
Voltage Regulator (5V)	3	\$0.78	\$2.34
DRV8834 Motor Driver Carrier	2	\$5.95	\$11.90
NEMA 17 Stepper Motor	2	\$13.5	\$27
3-D Printed Casing & Mechanical Parts	1	\$21	\$21
Vertical Blinds Set	1	\$27.51	\$27.51

			\$138.02
--	--	--	----------

II. Other consideration

Ethical

Our design appropriately follows the guidelines of the IEEE Code of Ethics. It strictly follows section 1 under the IEEE Code of Ethics “to hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices ...” as discussed later under Environmental/Sustainability, Health and Safety. If we find any other possible dangers to our design, we strive to promptly disclose this to the public so that they are fully aware. In addition, we also aim to improve our design by accepting feedback and criticisms from others.

Social and Political

A political issue that may arise from creating and manufacturing our product is security attacks on the supply chain. For example, imported components that we use to create our system may have been tampered with for malicious purposes. These components may have the quiet capability to collect and track data as the components travel to the country and used in residential homes.

Another political issue we may face in the future if we decided to mass produce this design is deciding whether to have manufacturing facilities within the US or in foreign countries. If the latter, we must be able to provide fair wage and working conditions to workers in factories and manufacturing facilities within those countries.

A social issue from using the Smart Window Blinds is that it is an internet-connected device. It belongs under IoT devices which in the past--and still some today--garnered a reputation for having low-security. The general public will be concerned that our system does a constant and effective job of protecting their data and we need to ensure that cybersecurity becomes a priority in further developments of our design.

Health & Safety

To consider the health and safety of our users, we first need to consider security challenges involving our system as well as certain safety conditions within a typical household. First, our system may be vulnerable to cyber attacks where the data of our users can easily fall into the wrong hands. We need to implement a more secure system to ensure that our users and their information is protected. This may be by using secure communication protocols between the ESP8266 web server and the web application so that all packet headers and data are encrypted. In addition, we also need to consider when our system may cause harm within the home. These situations may be when children are able to reach the attachable unit and use it for unintended purposes.

Environmental/Sustainability

Our project affects the environment by using its limited natural resources--from the electrical components we used to our packaging--in order to build our system. With this in mind, we want to provide 9V rechargeable batteries with the Smart Window Blinds system to reduce batteries (which contain toxic chemicals) from going back out into the environment once they are maxed out. People who use our system will not need repeatedly purchase batteries, but can simply recharge the ones they already have and reuse. In addition to this, we can reduce waste even more by using lesser amounts for our packaging.

Aesthetics

One of our goals in our design is to ensure that it does not disrupt the overall ambience of the room, regardless of where it is placed. We wanted to make the system compact in size and can be well-hidden if need be. In addition, we chose the color white for our packaging since it is a very neutral color and easily blends with the surroundings.

Manufacturability

The Smart Window Blinds system has been designed to optimize manufacturing functions like assuring best cost to quality, fabrication, testing, and other manufacturability factors. Our design uses minimum and low-cost materials so that it is easier for mass production. Our design is compact in size, as seen on Figure 6 of our PCB board, and can be easily repaired if malfunctions occur. Testing our design also does not take a long time because problems that occur during the process can be easily pinpointed to a specific module in the project: software, hardware or mechanical.

Lessons Learned

Throughout the design process, we have learned many things. One of the biggest is to compartmentalize tasks and certain parts of our project. Looking at the project in its entirety seems overwhelming, but when we were able to segment each part into software, hardware and mechanical modules, we were able to visualize and comprehend more clearly specific tasks we needed to do. We've come to realize that in engineering and in doing large-scale projects, it's vital to compartmentalize parts of the project. In this way, we can focus on one thing at a time whether the software part or the mechanical portion of the project. Another critical thing we learned is how important the online community is to troubleshoot and design certain aspects of our system. In particular, this applied to our software module integrating with our ESP8266. We are glad that we chose to use the ESP8266 for this project because there were so many tutorials and helpful documents online that allowed us to successfully implement our design. We never would have found out about the WiFi Manager library or using ESP8266 as a web server if it weren't for hobbyists and passionate individuals who have extensively documented their experience with the ESP8266. For the future, we know to opt for components, programming practices, softwares and other technological factors in the project process that have extensive documentation and resources available. Another skill that we picked up is strictly following schedules.

Doing this project has been good practice for us in maintaining and following our schedule for the past 3 months. Dr. Barrett was totally right when he said that we should not stress out about all the parts of the project and what we needed to do but just focus on the current thing on our schedule. We know, for certain, that in the future, when we handle other engineering projects that following and maintaining an appropriate project deadline and schedule will be paramount.












References

- [1] “Mobile Fact Sheet.” *Pew Research Center Internet & Technology*, Pew Research Center, 5 Feb. 2018, www.pewinternet.org/fact-sheet/mobile/.
- [2] Newman, Peter. “IoT Report: How Internet of Things Technology Growth Is Reaching Mainstream Companies and Consumers.” *Business Insider*, 27 July 2018, www.businessinsider.com/internet-of-things-report.
- [3] Gubbi, Jayavardhana, et al. “Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions.” *Future Generation Computer Systems*, vol. 29, no. 7, Sept. 2013, pp. 1645–1660.
- [4] Sparkfun. ESP8266 Module. Datasheet: <https://cdn.sparkfun.com/datasheets/Wireless/WiFi/ESP8266ModuleV1.pdf>
- [5] tzapu. WifiManager: ESP8266 Wifi Connection Manager. 2018. Github repository: <https://github.com/tzapu/WiFiManager>
- [6] Pieter, P. ESP8266 First Web Server. 2017. Github page: <https://tttapa.github.io/ESP8266/Chap10%20-%20Simple%20Web%20Server.html>
- [7] Texas Instruments. DRV8834 Dual-Bridge Stepper or DC Motor Driver. Datasheet: <https://www.ti.com/lit/ds/symlink/drv8834.pdf>
- [8] Mouser. LD1117AV33 Datasheet. 3.3V voltage regulator: <https://www.mouser.com/datasheet/2/389/Ld1117a-974076.pdf>
- [9] Mouser. L7805ACD2T-TR Datasheet. 5V voltage regulator: <https://www.mouser.com/datasheet/2/389/l78-974043.pdf>

Appendices

Appendix I. Schedule

Table 5. Project Schedule

ID		Task Mode	Task Name	Duration	Start	Finish
1			Finish App Design and ESP Interfacing (Deploy WebServer, ESP8266 Code, Serial Monitor to App)	8 days	Fri 2/8/19	Tue 2/19/19
2			Test ESP and Mobile App Connection	4 days	Tue 2/19/19	Fri 2/22/19
3			Finalize mechanical design	19 days	Fri 2/22/19	Wed 3/20/19
4			Testing	10 days	Thu 3/14/19	Wed 3/27/19
5			PCB Layout	4 days	Fri 3/22/19	Wed 3/27/19
6			Design Product Packaging	6 days	Thu 3/28/19	Thu 4/4/19
7			Finish and Assemble Final System Packaging	6 days	Fri 4/5/19	Fri 4/12/19
8			Final Design Improvements	9 days	Tue 4/9/19	Fri 4/19/19
9			Prepare for Final Presentations	6 days	Sat 4/20/19	Fri 4/26/19
10			Finish Final Project Report	16 days	Sun 4/28/19	Fri 5/17/19

Appendix II. Parts Lists

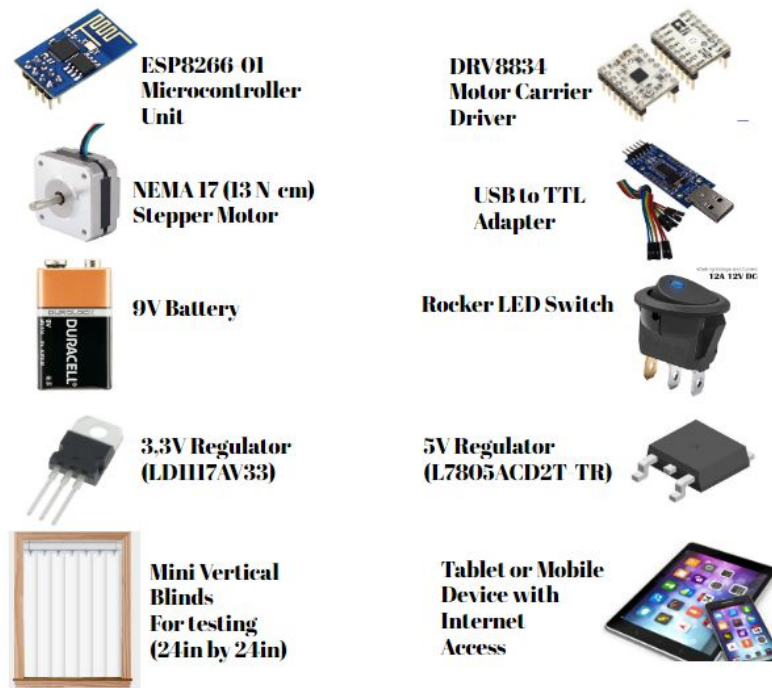


Figure 18. Part lists

Appendix III. PCB Board & Schematic

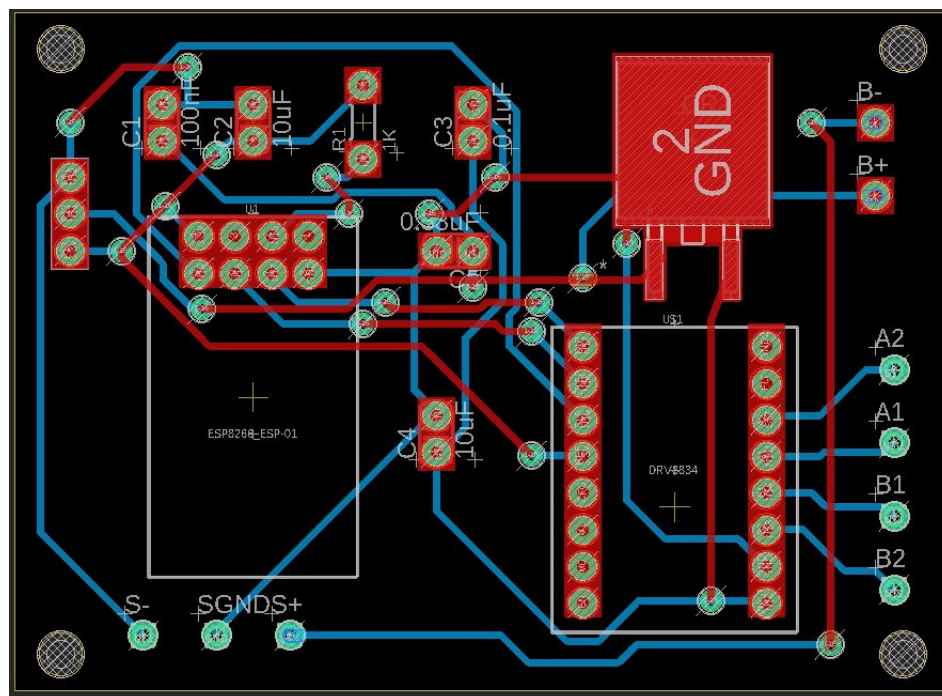


Figure 19. PCB Board

Appendix IV. Arduino Code

```
#include <ESP8266WiFi.h>
#include <WiFiManager.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#import "index.h"
#import "open.h"
#import "close.h"

const int dirPin = 2;          //GPIO2
const int stepPin = 0;        //GPIO0
const int sleepPin = 3;       //Using RXD as GPIO during normal mode operation
WiFiServer server(1337);
ESP8266WebServer ESPserver(80);
//String header;
WiFiClient client = server.available();

void printWiFiStatus();
void handleRoot();
void handleOpen();
void handleClose();

void setup(void) {
  Serial.begin(115200);

  WiFi.disconnect();
  Serial.println("Wifi Credentials Reset");

  WiFiManager wifiManager;
  wifiManager.autoConnect("AutoConnectAP");
  Serial.println("Connected.");

  pinMode(dirPin, OUTPUT);
  pinMode(stepPin, OUTPUT);
  pinMode(sleepPin, OUTPUT);

  Serial.println(WiFi.localIP());
```

```

// Start TCP server.
server.begin();

ESPserver.on("/", handleRoot); //Which routine to handle at root location
ESPserver.on("/open", handleOpen);
ESPserver.on("/close", handleClose);

ESPserver.begin();
Serial.println("HTTP server started.");
}

void loop(void) {
  // Check if module is still connected to WiFi.
  if (WiFi.status() != WL_CONNECTED) {
    while (WiFi.status() != WL_CONNECTED) {
      delay(500);
    }
    // Print the new IP to Serial.
    printWiFiStatus();
  }

  ESPserver.handleClient();
  WiFiClient client = server.available();

  if (client) {
    Serial.println("Client connected.");

    if (!client.connected()) {
      Serial.println("Client disconnected.");
      client.stop();
    }
  }
}

void printWiFiStatus() {
  Serial.println("");
  Serial.print("Connected to ");

```

```

    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
}

void handleRoot() {
    String htmlStr = main_page; //define html file in string
    ESPserver.send(200, "text/html", htmlStr);

}

void handleOpen() {
    String htmlStr = open_page; //define html file in string
    ESPserver.send(200, "text/html", htmlStr);
    digitalWrite(sleepPin, HIGH);    //enables motor driver, turns off sleep mode
    digitalWrite(dirPin, HIGH);
    analogWrite(stepPin, 127);    //set PWM to 80% duty cycle for now, IO0
    Serial.println("Open signal sent.");
    delay(2200);
    analogWrite(stepPin, 0);
    digitalWrite(sleepPin, LOW);
}

void handleClose() {
    String htmlStr = close_page; //define html file in string
    ESPserver.send(200, "text/html", htmlStr);
    digitalWrite(sleepPin, HIGH);    //enables motor driver, turns off sleep mode
    digitalWrite(dirPin, LOW);
    analogWrite(stepPin, 127);    //set PWM to 80% duty cycle for now
    Serial.println("Close signal sent.");
    delay(2200);
    analogWrite(stepPin, 0);
    digitalWrite(sleepPin, LOW);
}

```