**COS 491 Senior Project I**

**Automated Blinds with Arduino**

**Rosen Stratkov, ID# 100097089**

**Author:**   **, Date:**

*signature*

**Supervisor:**   **, Date:**

*signature*

**Blagoevgrad, 2017**

**Title**: Automated Blinds with Arduino

**Author**: Rosen Stratkov

**Abstract:** The project is about blinds which are to be automated using Arduino Uno. The blinds will open and close depending on the light inside and outside of the room. The blinds themselves will be controlled with an electromotor. The Arduino Uno board will be programmed to do the aforementioned actions. The blinds can be also controlled by physical buttons on the board or by a mobile application which is developed specifically for the project. The connection between the application and the Arduino Uno board is done through a Wi-Fi connection using aESP8266 Wi-Fi module. The user is asked to provide the IP address of the Wi-Fi module and the port number. This data can be retrieved from the Serial Monitor window through the Arduino IDE when the program is initially compiled.

**Declaration of authorship:**

“The Senior Project/Bulgarian Diploma Thesis presented here is the work of the author solely, without any external help, under the supervision of Professor Volin Karagiozov. All sources, used in development, are cited in the text and in the Reference section.”

Author:

1. Introduction:

The idea about this project came to me during while I was contemplating how I can combine by software and my hardware skills in one. I decided to do something that would be useful for the everyday life and that is when I discovered the concept Internet of Things (IoT). I was looking at my blinds which I have just bought and thought to myself whether I can make them move in a non-traditional way. That was when I stopped wondering what my senior project would be. I decided that I was going to make something to move the blinds from open to close.

I am interested about this project because I believe that we can make our everyday actions easier and more enjoyable. The automated blinds are my first step towards automating other home appliances. I imagine this as turning my boring actions into interesting ones where I can have fun while doing a very mundane task. The blinds are one thing that I personally love having in my house but I was always fed up with the traditional way of operating them. I hope that I manage to integrate this project into my home and spend the days admiring it.

I had two options about how to make this happen. One was to make the blinds move using light sensors. This option made it unnecessary for any human touch for the blinds to operate. All that was needed was lights inside and outside and darkness inside and outside. When the light is turned on inside, the blinds would close so that nothing is seen from the outside. When it is daytime and it is light outside and dark inside, the blinds would open. I realized that this solution was not enough to make the best of the blinds functionality.

I decided that I will make the blinds controlled remotely according to the user’s preference. To do this I created an Android application. The user would use the application to communicate with a Wi-Fi module connected to the blinds. The user has the options to open and close the blinds.

The technologies used in this project are Arduino Uno board, light sensors, stepper motor, driver for the stepper motor, Wi-Fi module and a smartphone. Other hardware things that are required for the project are jumper cables, buttons, electrical tape, resistors, capacitor, batteries, USB Type A/B cable for the Arduino Uno and a USB cable for the smartphone. Some things that I used to help me do the project are soldering gun (to solder the pins of the stepper motor driver), voltmeter (to measure the voltage current input to the ESP8266 Wi-Fi module since it is very delicate device), my personal laptop (to implement the code for the project) and my audio speaker (to keep me in good mood while working on the project). The software needed for the project includes Ardino IDE, Android Studio IDE and driver for the Wi-Fi module. The programming languages that I used are C programming language – for the code uploaded on the Arduino Uno board and Java – for the Android mobile application.

I came through a lot of difficulties during the development of this project and I managed to keep my enthusiasm until the end. I learned many new things about mobile connection, Wi-Fi, stepper motors, sensors and more.

2. Specification and Analysis of the Software and Hardware Requirements

The hardware requirements for my project include two light sensors (photo resistors). Using the light sensors I was able to tell the Arduino Uno what to do with the stepper motor connected to the blinds. The light sensors work by detecting how much light is emitted towards them. They work analogously and have 1023 levels for detection. This means that the less light there is, the higher the value the light sensor will be. The more light there is, the lower the value will be. Since I do not need to be that specific for my project, I value the level of the light sensors by being either high or low. This helps me tell the program whether there is light and whether there is not. The sensors are integrated in a small module with three legs – one for Voltage (5 volts), another for Ground and a third one for the signal sent to the board.

Another hardware requirement for my project is a stepper motor accompanied by a driver for it. In the beginning I was using a stepper motor of size NEMA 8 which turned out to be weak for the purposes of the project. I had it connected perfectly with the blinds I am using, but during the tests I did it turned out that it did not have enough power to spin the blinds. That is why I had to find a bigger and more powerful stepper motor and I got the stepper motor of size NEMA 13.

The stepper motor has 4 cables connected to it. These cables are then connected to the driver. The driver is a Pololu A4988. The driver has 16 pins that help to operate the motor. The first pin is to enable the driver; the next three pins are to specify the step size of the motor. They enable the motor to do smaller or larger steps depending on what is needed.

|  |  |  |  |
| --- | --- | --- | --- |
| MS1 | MS2 | MS3 | Microstep Resolution |
| LOW | LOW | LOW | Full step |
| HIGH | LOW | LOW | Half step |
| LOW | HIGH | LOW | Quarter step |
| HIGH | HIGH | LOW | Eight step |
| HIGH | HIGH | HIGH | Sixteenth step |

**The steps of the stepper motor are set as shown (MS1, MS2 and MS3 and the three pins’ names of the stepper motor driver Pololu A4988):**

For my project I use the half step option. This means that I have connected pins MS1 and MS3 to Ground and pin MS2 to 5 volts.

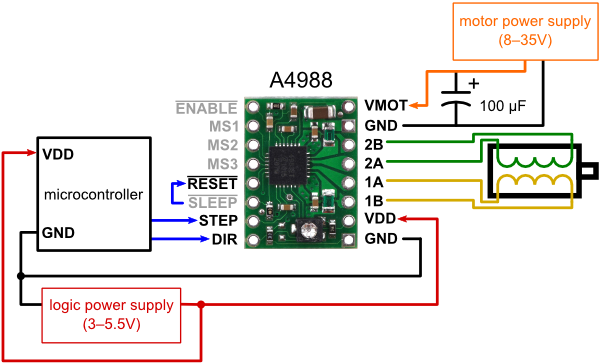
The next two pins are the reset and sleep pins.

The next two pins of the driver are to determine the direction and to make stepper motor spin. If the direction pin is set to LOW, the stepper motor spins counterclockwise. If the direction pin is set to HIGH, the stepper motor spins clockwise. If the direction pin is set to HIGH, which means 5 volts, the driver makes the motor spin to the direction according to the direction pin setting.

The next two pins as for the motor power supply. Stepper motors need higher voltage in order to operate, especially the one I am using which is of size NEMA 13. The driver can take voltage in the range of 8 volts and 35 volts. The plus and minus of the supply must be connected with a capacitor. In my case I am using three 9 volt batteries, the sum of which is 27 volts. In order to make the input level stable I am putting a 500 uf capacitor which has a maximum voltage limit of 50 volts. I had problems in the beginning since I did not have a capacitor and my stepper motor was behaving awkwardly – it was working for about 20 seconds and then it started only to vibrate. I was trying to figure out why was it doing this and then I figured out that it could be because of the voltage current not being stable enough. Then I bought a capacitor and when I tried operating the stepper motor with it, it was working flawlessly.

The next four pins of the Pololu A4988 stepper motor driver are for the two coils inside of the stepper motor. They partner each other two by two so that they get electrified and operate the stepper motor.

The last two pins are for the 5 volts for the driver itself and for the Ground of the board.



**A picture with every pin described of the stepper motor driver Pololu A4988 driver for the stepper motor**

Continuing with the hardware requirements, I am using two LED diodes – one green and one red. I use them to indicate when either of the modes of the blinds (open and close) is in action.

I use buttons which enable the user to operate the stepper motor opening and closing the blinds. I use this functionality in case of fault in the mobile application.

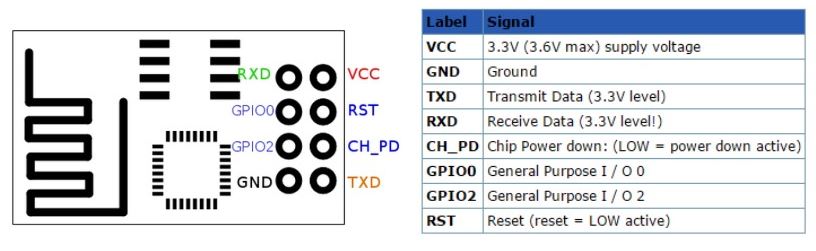
I am using an original Arduino Uno Board. When I started doing this project I had a Chinese replica of the original Arduino Uno Board. It is almost identical with the only major difference being that is has a different microcontroller chip. The chip on the Chinese replica of the Arduino Uno is a CH340G. The driver for this chip is also Chinese and I had to find it online and install it to be able to use the Chinese replica board. I did this, but unfortunately the driver was unsigned and Windows does not tolerate unsigned drivers. To be able to use this driver I disabled the signed checking from the BIOS of my computer. The Chinese replica board was working fine until one day when Windows just stopped recognizing it. This is when I was forced to get an original Arduino Uno board in order to not have difficulties in the future. The chip on the original Arduino Uno is ATMEGA328P. It is also easy to change by hand if something bad happens to it. The CH340G, which is on the Chinese replica, is soldered to the board and if it blows up, the whole board goes in the trash. The chip ATMEGA328P does not require any special software. The Arduino IDE comes with a driver for the ATMEGA328P.

Other hardware requirements include resistors I used in order to operate with the LED diodes safely. 5 volts can burn a regular LED diode.

I am using three 9 volt batteries connected together to get a voltage of 27 volts. This is the cheapest solution to providing voltage of around 27 volts. Such voltage is needed by the stepper motor to make it spin.

I am using a laptop to provide 5 volts to the Arduino Uno board. I get the 5 volts from the USB cable.

I am using a Wi-Fi module ESP8266 to establish connection between a smartphone and an Arduino Uno board. The ESP8266 has 8 pins. It has an Rx and Tx pins which are to be connected to the Arduino Uno Rx and Tx pins. The Tx and Rx means transmit data and receive data accordingly. We need those two connected between the two boards so that they can communicate between each other. The ESP8266 has two pins for general purpose input/output called GPIO0 and GPIO1. The pins of the GPIO0 must be connected to Ground of the ESP8266 – this switches the Wi-Fi module into programming mode. The ESP8266 has a pin named CH\_PD which is to disable the device when not in use – to save power. The last two pins of the module are the Voltage input and the Ground. The ESP8266 is very delicate and runs on 3.3 volts. 5 volts will destroy it.



**A scheme with every pin described of the ESP8266**

The ESP8266 needs a stable voltage current so I put a voltage regulator TS2940 which makes the voltage stable to 3.3 volts.

At the beginning I did not have the voltage regulator and was experiencing some flakey performance of the ESP8266. I thought that a voltage regulator would fix the issue so I got one. I measured the voltage current with a voltmeter to be sure that I am giving the ESP8266 exactly 3.3 volts. The voltage was correct but the times that the ESP8266 was accepting the code I was trying to upload to it were decreasing. Eventually I could not even upload the basic built-in program called Blink. It is a program that makes the LED diode of the ESP8266 blink. I used this program to check if the ESP8266 is working properly and if I can upload code to it. I think that the ESP8266 is broken right now which constrains me from further development using it.

Another hardware requirement is a laptop. I need the laptop to provide 5 volts to the Arduino Uno board and to the other elements attached to it.

These were the hardware requirements that I need for the project to operate.

Other hardware requirements include a soldering gun. I used a soldering gun to solder the pins of the stepper motor driver A4988. The stepper driver arrives with its’ pins not attached so to use it I had to do some soldering.

I am using a smartphone to communicate with the Arduino Uno board through the ESP8266.

These are the things that I use in order to implement code, debug the program and install new things – a laptop, buttons and a smartphone.

Software requirements for the project are two integrated developing environments. The first one is the Arduino IDE which I use in order to develop, debug and upload code to the Arduino Uno board and to the ESP8266.

I am programming both the Arduino Uno board and the ESP8266 using the C programming language. One reason for this is that I feel comfortable coding in the C programming language. Another reason is that it is more low level programming language and can operate faster on programmable device compared to higher level programming languages.

Using the C programming language also makes the code lighter for the Arduino Uno and the ESP8266. This is because it does not take a lot of space. For example the sketch for the Wi-Fi connection with the application takes only 24% of the space of the Arduino Uno, which is 7782 bytes of memory. The storing of the local variables takes 687 bytes of memory (33% of dynamic memory), but can be optimized in the future to take less space.

C:\Users\Seeker\Desktop\memory on device.png

**A screenshot from the Arduino IDE showing the memory that is used to store the code for the Wi-Fi application program**

The Java application for the Android smartphone takes is about 15 MB. For the mobile application, is not as important to be light, as it is for the Arduino Uno and for the ESP8266. This is because the Arduino Uno board and the ESP8266 have a much smaller storage space than a regular smartphone. The smartphone also has a faster and better processor which makes it faster for even a not-so-well optimized application to run with a decent speed.

The Arduino program that exemplifies the mobile controlled blinds is done by buttons and takes considerably smaller space. It uses 2334 bytes of the Arduino Uno board, which is only 7% of the program storage. The global variables take 282 bytes of space, which is 13% of the dynamic memory of the Arduino Uno board.

C:\Users\Seeker\Desktop\fdsfds.png

**A screenshot from the Arduino IDE showing the memory that is used to store the code for the manual mode with the buttons controled blinds program**

The sketch for the use of the light sensors controlling the blinds takes even less memory on the Arduino Uno board than the manual control program. This program takes 2240 bytes, which is 6% of the memory of the Arduino Uno board. The program takes the same space for the local variables stored in the dynamic memory (282 bytes and 13%) of the board.

C:\Users\Seeker\Desktop\fdsfds.png

**A screenshot from the Arduino IDE showing the memory that is used to store the code for the light sensors controlled blinds program**

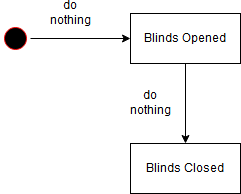
The other integrated developing environment is the Android Studio IDE. I use it to develop, debug and upload code to a smartphone. As I mentioned earlier, the size of this application is not as important as the ones that are uploaded to the Arduino Uno and the ESP8266 board. However, I stuck to the philosophy of keeping it simple. I developed an application as small as I could which also has an easy to understand graphical user interface.

Functional requirements of the project include the operation of the blinds through a spin of a stepper motor. The blinds are in state open or closed. What is open and closed in the blinds is the position of the blinds – the ones usually operated with the solid stick attached to the blinds. My project is not meant to move the blinds up and down.

The stepper motor moves according to what the light sensors are detecting. If there is light on the sensor meant to face inside and not on the one meant to be faced outside the stepper motors spins the blinds to a state of them being closed. If there is light on the sensor facing outside and not on the one facing inside, the stepper motor moves the blinds to a state of them being open.

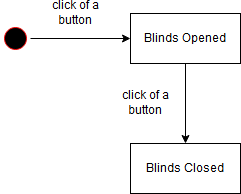
The user can operate the blinds via buttons too. If the user presses the Open button – the blinds open and if the user presses the Close button – the blinds close.

The user is also able to operate the blinds via a smartphone. The user has to input the IP address and the port number of the Wi-Fi module connected to the Arduino Uno and is then able to operate the blinds the same way as with the physical buttons, but this time with buttons on smartphone.

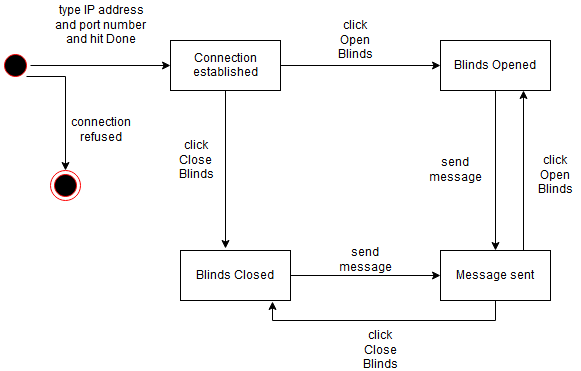


**A state chart diagram showing the process of the blinds going from one state to another without any action of the user**

This diagram shows the different states of the blinds. It starts with opening the blinds because we always assume that the blinds are closed before the start of the program.



**A state chart diagram showing the process of the blinds going from one state to another because of the click of a physical button**

****

**A state chart diagram showing the process of connecting to the Wi-Fi ESP8266 device through the Android application and controlling the blinds going from one state to another**

The state chart above shows the process of connection by the Android application to the Wi-Fi device and controlling the blinds. The application itself is quite self-explanatory so the user does not need any bonus information about it.

When the connection fails to be established it could be because of the lack of Internet connection or because the IP address or the port are not the one that are of the ESP8266.

When the connection fails, a dialog message is shown to the user.

C:\Users\Seeker\Downloads\Untitled Diagram(2).png

**This is a UML sequence diagram of the Wi-Fi controlled blinds application**

Some of the non-functional requirements for the project include easiness of work with the program, stability of the program (not having unwanted behavior) and efficiency.

The easiness of the program is achieved through the simple graphical user interface of the mobile application. It does not have any unexpected behavior and can either connect to the Wi-Fi device or not and can either open or close the blinds through two buttons. If wrong data, such as garbage string, is added into the text boxes the program will not connect and prompt a dialog message saying that it couldn’t establish a connection.

For the easiness of work of the light sensors controlled blinds program the user has absolutely nothing to worry about. He does not have to touch any button. The program is set and works all by itself while the user admires it.

For the easiness of work of the button controlled program, the user does not have much to mess with. There are two buttons and if one is clicked, the blinds switch to one state, and if the other button is clicked, the blinds switch into the other state. There are a green and a red LED diodes which indicate which state in performed.

The stability of the program is great. The only failure that can happen is a fault in the connection to the internet, a fault in the ESP8266, as I figured out it is a delicate device to say the leas and the running out of batteries. Other than that the code works flawlessly. The hardware is connected in a way that makes the project work without any glitches. I am glad that I managed to fix the issues with the hardware because these issues were causing me a lot of head bangs. The main fixing thing was the capacitor that I added to the stepper motor driver. It was really a game changer.

The efficiency of the program is such that it makes the worries of the user less and makes him admire a mundane task. One can say that it saves time, but the idea of the project is not about saving time of the user. The idea is rather to make the experience of an everyday task more pleasurable for the user. It is to make his life easier when he forgets to close the blinds at night when turning the lamp on.

3. Design of the software solution

The following is the explanation of the code for the Arduino Uno board.

In order the blinds to move because of the light sensors, the light sensors need to be initialized. I initialize them on the analog pins 0 and 1 of the Arduino Uno board. I then declare integer variables to hold the digital value of the sensors. Then I declare an integer which will work as a flag to keep saved the state of the blinds (open of closed).

Then I declare the functions for opening and closing of the blinds. This is done by setting the direction pin of the stepper motor driver to either HIGH or LOW depending on which direction we want the stepper motor to be spinning to. Then we form a loop that would spin the stepper motor for a certain amount of milliseconds.

In the setup function of the program, which is run only once, we set the pins corresponding to enabling, step and direction of the stepper motor driver as output.

In the loop function the program reads the value of the two light detectors and stores it into a variable. Then, depending on whether the variables are high or low, the program calls the corresponding function – to close or to open the blinds.

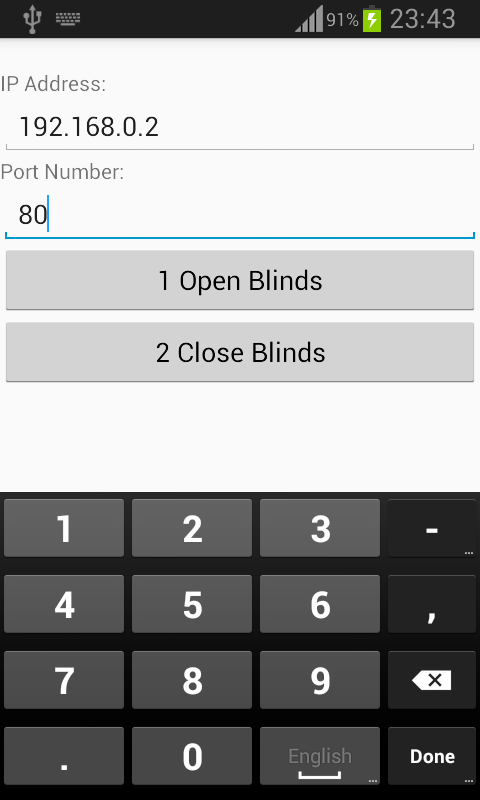
The program starts assuming that the blinds are closed. Then it is up to the variable that holds the value of the blind being closed or open to not allow the blinds to go through the open or the closed function more than once in a row.

For the manual physical button mode we initialize variables for the buttons to be used and set the pins for them in a mode of input. Then again according to which button is pressed the blinds either open or close. There is again the flag to make sure that the blinds are not “opened twice”, or “closed twice”.

For the connection between the ESP8266 and an Android smartphone there is code on both sides.

The Android application has two text boxes. One is for the IP address of the ESP8266. The other one is for the port number. It also has two buttons for the user to either close or open the blinds.

If the entered data for the IP address and the port number corresponds to those of the ESP8266, after pressing a button, the blinds will go into an open position. If the closing button is pressed, the blinds will go into a closed position. There is also a dialog message sent to give the user information of the state of the blinds.



**The user interface looks like on the mobile application (Home Screen)**

I create a function to send data to the ESP8266. The function has three parameters – the command/data, timeout period (if a response is expected in certain amount of time, the timeout parameter can be set to for example expect data to receive in one second). The third parameter is a boolean variable called debug. If set to true it will print out the response from the ESP8266.

The code of the Arduino first resets the ESP8266. Then it changes the mode of the Wi-Fi module to an access point. Then there are the username and the password that are set for the server. The Arduino then configures the ESP8266 to be available for multiple connections and turns on server on port 80.

In the loop function the first thing the program does is to check if the Wi-Fi module is sending a message to the Arduino Uno board. Then there is a check if the string received contains a certain string in it. If it does, it means that there is data sent by the application. I will explain the workflow of the application after explaining the Arduino code first.

After the Arduino finds the string that is desired there is a delay of one second so that the buffer is filled up and collects the whole data that was sent. Then the program stores the connection id to an integer variable by reading it and subtracting 48 from it. The reason for this subtraction is that the ESP8266 sends data in ASCII code. By subtracting 48 the program gets the real value, converted from Ascii.

Then the program, using the find function, looks for “pin=” string in the message. Then the program again converts the pin number from ASCII to the desired value by subtracting 48.

After the pin number is acquired it is time to do the action accordingly. If the Arduino receives a string containing the character “1” it opens the blinds. If it receives a string containing the character “2” in it, it closes the blinds. In both cases creates a string holding the message to the application saying what the action that is performed is.

There is a function about sending a response. This is where the HTTP header is built. The header is needed for the Android application to read the data. If the header is not there, the content would be lost.

The content that is to be sent is then concatenated to the header.

There is a bug in the code which omits the last character sent. To work around with this bug I add an empty space at the end of the string

After that the message becomes data to be sent to the phone. It is sent by a sendCIPData function. The function needs the connection id for each device connected to the ESP8266. It also needs the length of the data.

When the CIP command is obtained then the CIP command is sent along with the message.

The sendData command writes the response to the ESP8266. The write function is used in order to write a response more than 62 characters. SerialWrite can write up to 62 characters only, which does not do the job and that’s why we use the write function.

For the Android code there are three files. There is the MainActivity, the Manifest and the layout of the application.

In the layout file is the user interface of the application. There are the IP address and the port number text boxes. There are also the two buttons for the open and close of the blinds. The open button is labeled as number 1 and the close as number 2.

In the manifest we have to enable the program to allow the phone to have access to the internet. Without this line of code the phone has restricted access to the internet.

The main activity code first saves the data of the IP of the server and the port number so that when closed, on the next launch it is not needed to be typed again. If the application is opened for the first time, then there will be nothing in the text boxes.

Here is where the buttons and the text boxes are created.

The program then finds out which button is pressed. The button id is compared to the id of the button pressed. A variable is assigned to the pin value to be toggled.

Then the http request is executed. What is sent is the parameter value holding the id of the button pressed, the IP address, the port number and the string “pin” which corresponds to the same string in the Arduino code so that the Arduino reads it.

There is a function to send the request to the ESP8266. Here the message is composed together with the variables that make it one with the most important being the pin number to be toggled.

When the ESP8266 sends data back to the Android app, the content is captured and saved in a variable containing the string containing the response message from the Arduino. This is the string that says whether the blinds are open or closed.

After receiving the response the connection is closed.

I created a class to perform tasks in the background. This is required because the http request will block the user interaction with the application. It might even freeze the application so this class lets the program run processes, in this case the http request.

Inside this class I implemented the creation of the dialog box.

What I need the program to do in the background is to send the request.

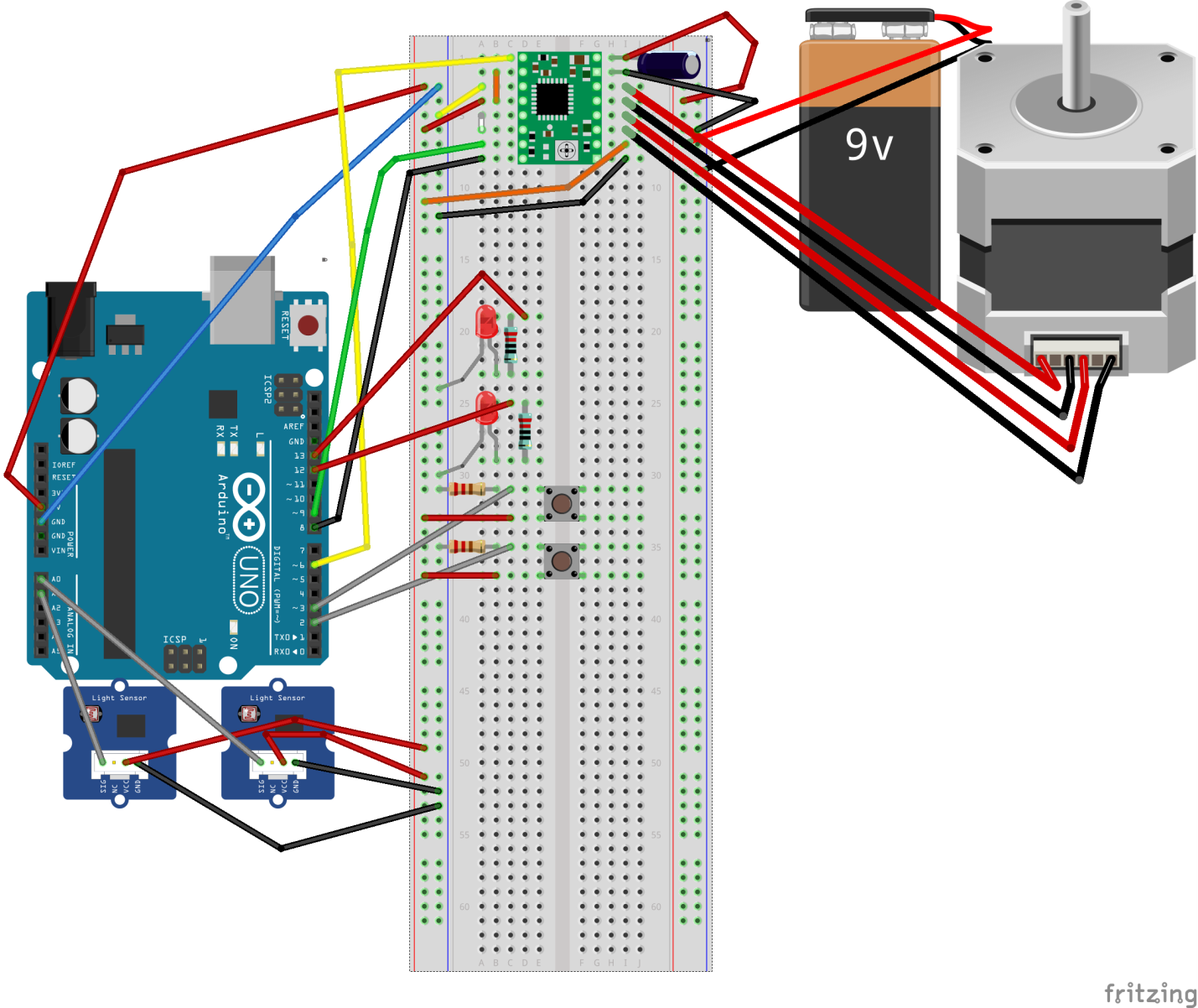
Then there is a function that changes the dialog while it is displaying something. This is the dialog saying that information is being sent and it changes to display the message sent by the Arduino about the state of the blinds.

There is also a function to execute the opening of the empty dialog before everything else is executed.

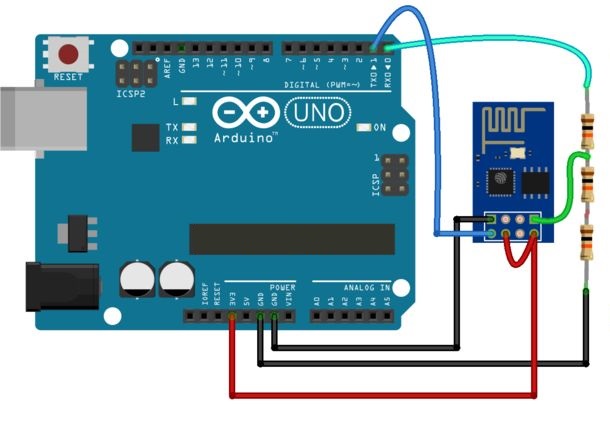
These functions do not need to be called one by one. An execute function calls all of them in the correct order so I do not have to worry about messing them up.

This is a diagram of the aforementioned classes:

C:\Users\Seeker\Downloads\Untitled Diagram(4).png



**Schematic of the elements on the Arduino Uno board**



**Schematic of the connection of the ESP8266 module**

There is no need to implement anti-hacking algorithms for this project. If someone tries to attack the Arduino via Wi-Fi, he or she should know the IP address and the port number.

I think that it is safe enough that the IP address and the port number need to be written down so that a connection is established.

4. Implementation

I am using the physical computing platform Arduino Uno. I think it is the best piece of technology for my project because it is easy to use and it also has the functionalities I need for my project. The microcontroller is based on the ATmega328P chip. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button. The Arduino Uno has 32 KB of flash memory which is enough for the purpose I am using it.

Another physical board that I am using is the ESP8266. It is a simple Wi-Fi module. It allows microcontrollers to connect to a Wi-Fi network and make simple TCP/IP connections.

The driver for the ESP8266 can be downloaded through the Arduino IDE and can be installed easily. The Arduino IDE is available for free on the official website of Arduino. It has a built in compiler. I use the IDE to upload code to the Arduino Uno and the ESP8266 board. The IDE is a little too simple and misses some of the functionalities which a more modern IDE has but it gets the job done.

The upload of the code to the ESP8266 can be tough sometimes. I figured out that it has a flakey behavior even when the voltage is stabilized on its’ input. After all the struggle it turned out that it does not even want to accept data to it.

I used the C programming language because it is easy for me to use and it is also quite a low-level language so that I get some good performance from the Arduino Uno. I also used the Java programming language to develop the mobile application for Android. I am writing it in Java because it is the best suited language with Android devices.

I used the library SoftwareSerial.h for the Arduino. I find it useful for my project since it is not too complex and the functions in it are developed to serve the purpose of a small project.

The application is installed through the USB port of the Android smartphone. I did this by enabling the developer options and directly running the application on the smartphone.

This is a fragment of the code showing how the stepper motor spins the blinds to a closed position:

void blinds\_close()

{

digitalWrite(8,LOW); //direction of the stepper motor

for(int Index = 0; Index < 500; Index++) //movement of the stepper motor

{

digitalWrite(9,HIGH);

delayMicroseconds(500);

digitalWrite(9,LOW);

delayMicroseconds(500);

}

}

Pin 8 is connected to the Direction pin of the stepper motor driver Pololu A4988 and when it receives LOW state it means that it spins to stepper motor to the direction needed so that it is closed. If the state is HIGH, the stepper motor is spinning on the opposite direction and the blinds reach a state of being opened.

The loop is to make the stepper motor spin for a certain period of time. The delays are to make the stepper motor remain in certain state for the milliseconds specified.

This is how the server is set on the ESP8266:

sendCommand("AT+RST\r\n",2000,DEBUG); // reset module

sendCommand("AT+CWMODE=1\r\n",1000,DEBUG); // configure as access point

sendCommand("AT+CWJAP=\"AAAA\",\"123\"\r\n",3000,DEBUG);

delay(10000);

sendCommand("AT+CIFSR\r\n",1000,DEBUG); // get ip address

sendCommand("AT+CIPMUX=1\r\n",1000,DEBUG); // configure for multiple connections

sendCommand("AT+CIPSERVER=1,80\r\n",1000,DEBUG); // turn on server on port 80

This code also prints to the Serial Monitor the needed IP address which is then written by the user in the mobile application. Here I use the function I implemented called sendCommand(). It is a simple function that uses the built-in function print() to send data to the ESP8266.

The sendCommand() function looks like this:

String sendCommand(String command, const int timeout, boolean debug)

{

String response = "";

esp8266.print(command); // send the read character to the esp8266

long int time = millis();

while( (time+timeout) > millis())

{

while(esp8266.available())

{

// The esp has data so display its output to the serial window

char c = esp8266.read(); // read the next character.

response+=c;

}

}

if(debug)

{

Serial.print(response);

}

return response;

}

It takes three parameters – the command itself, a possible timeout and a debug boolean variable which if set to true will print the response that was sent to the Arduino by the ESP8266.

When the Arduino is receiving information from the Android application is looking for the specific string contained in the message and it sets its’ cursor exactly there. This is the code that finds that specific string:

esp8266.find("pin="); // advance cursor to "pin="

It is looking for the string “pin=” in the message which means that this has to be sent from the application. On the application side here is the message that is sent:

if(ipAddress.length()>0 && portNumber.length()>0) {

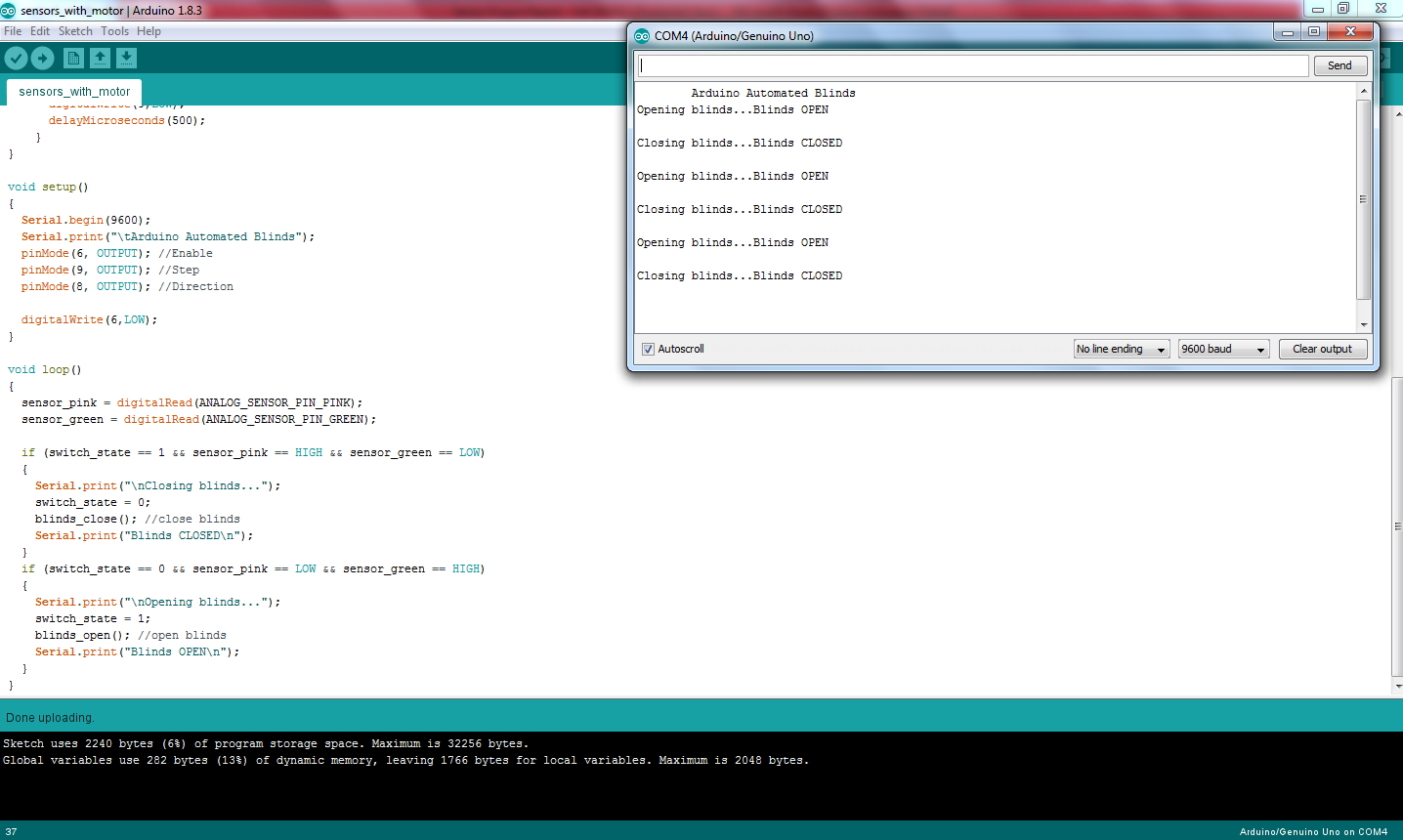
new HttpRequestAsyncTask(

view.getContext(), parameterValue, ipAddress, portNumber, "pin"

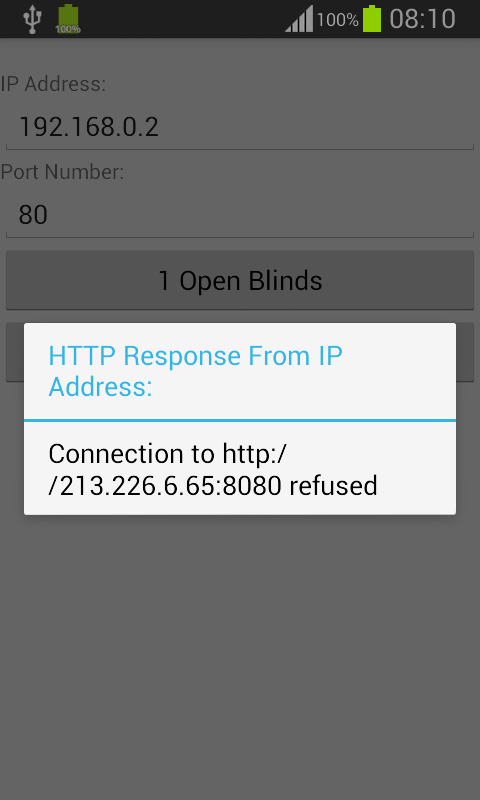
).execute();

}

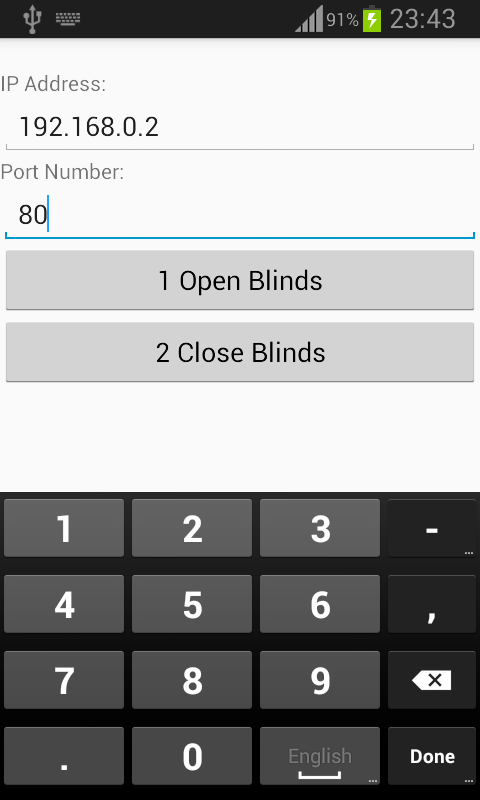
5. Screenshots and Photos



**This is a screenshot showing the output to the Serial Monitor informing the user for the state and the progress of the blinds movements:**

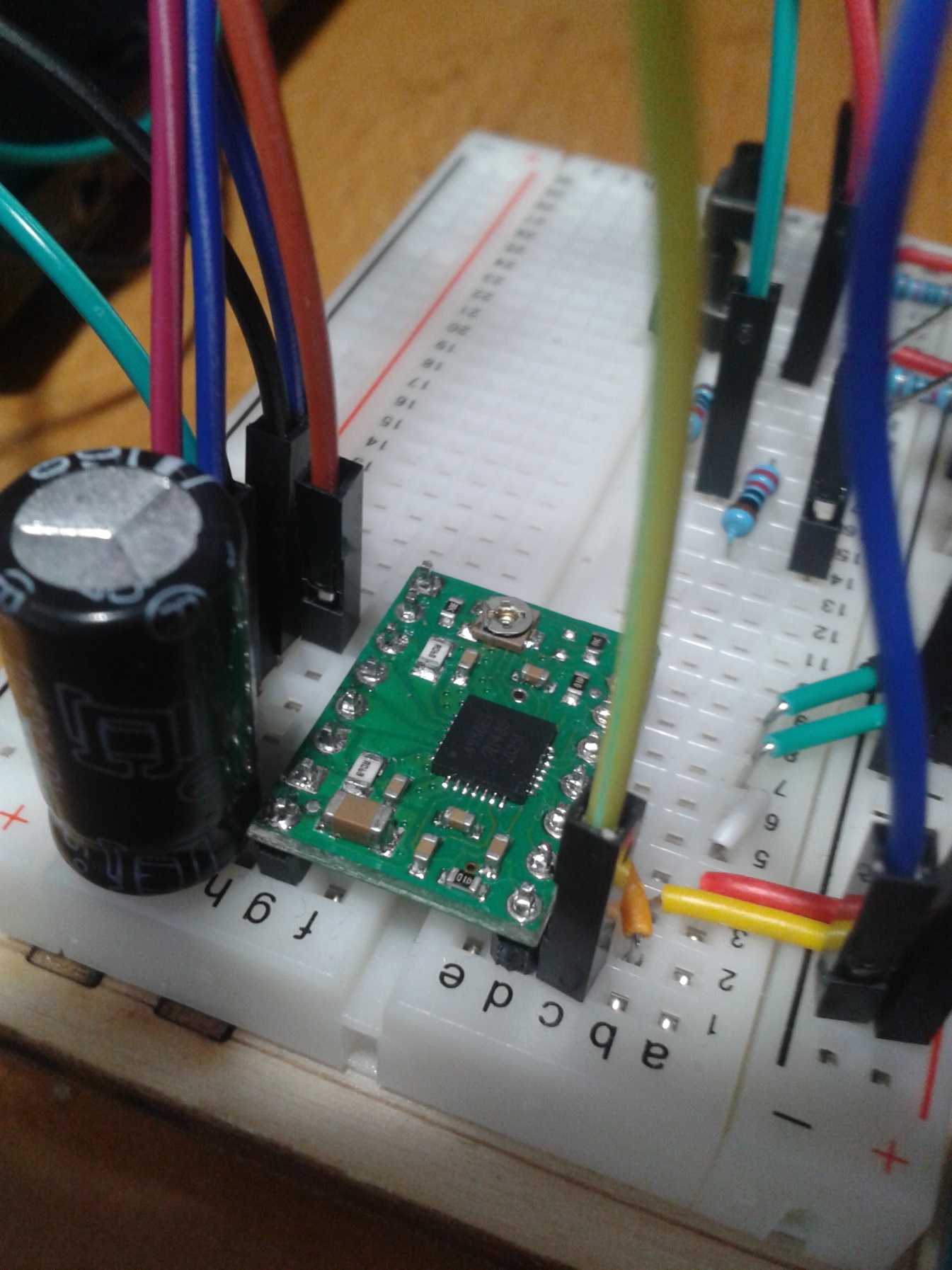


**A dialog message shown to the user when the connection to the ESP8266 fails**

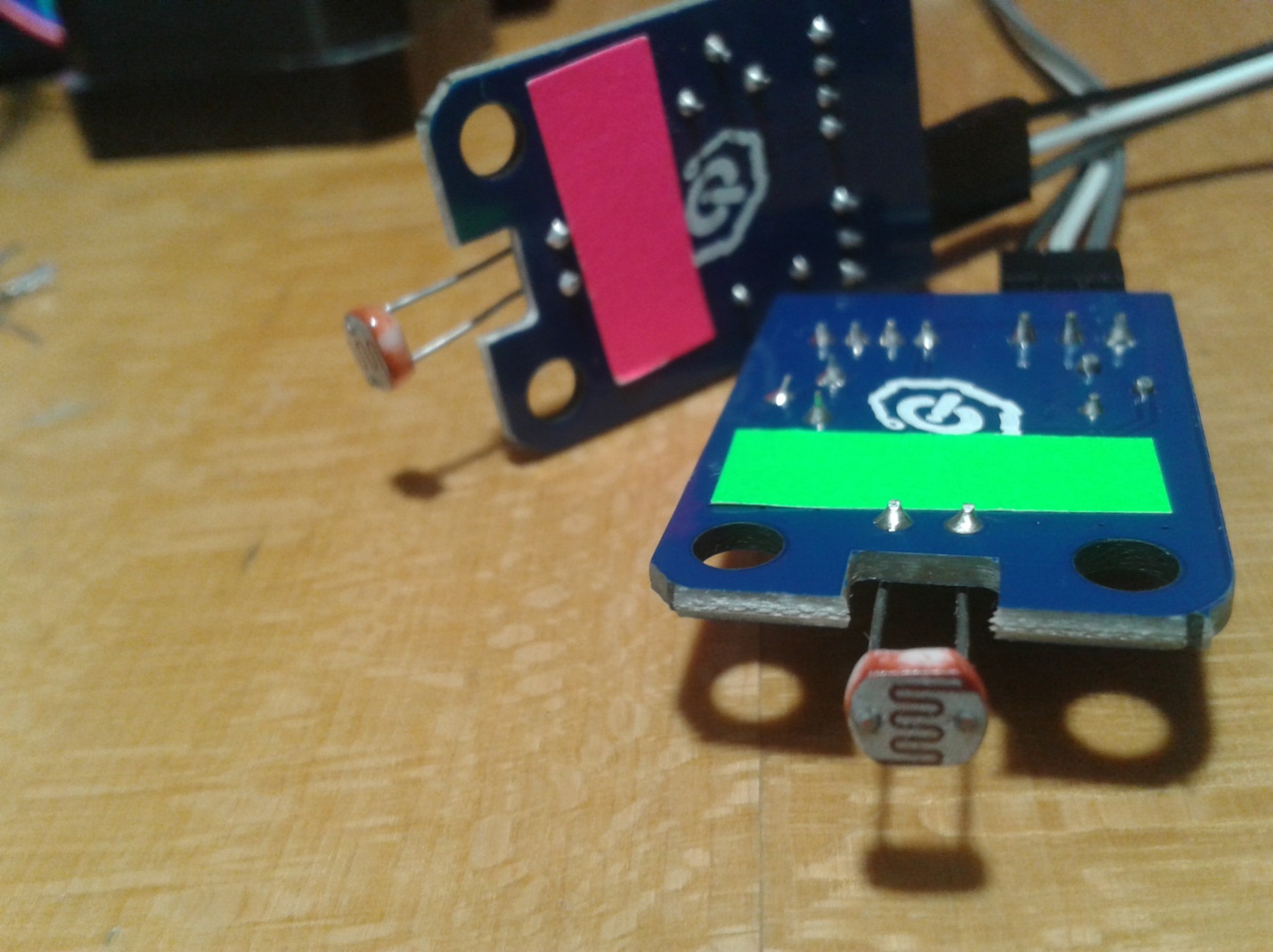


**Home screen of the application**

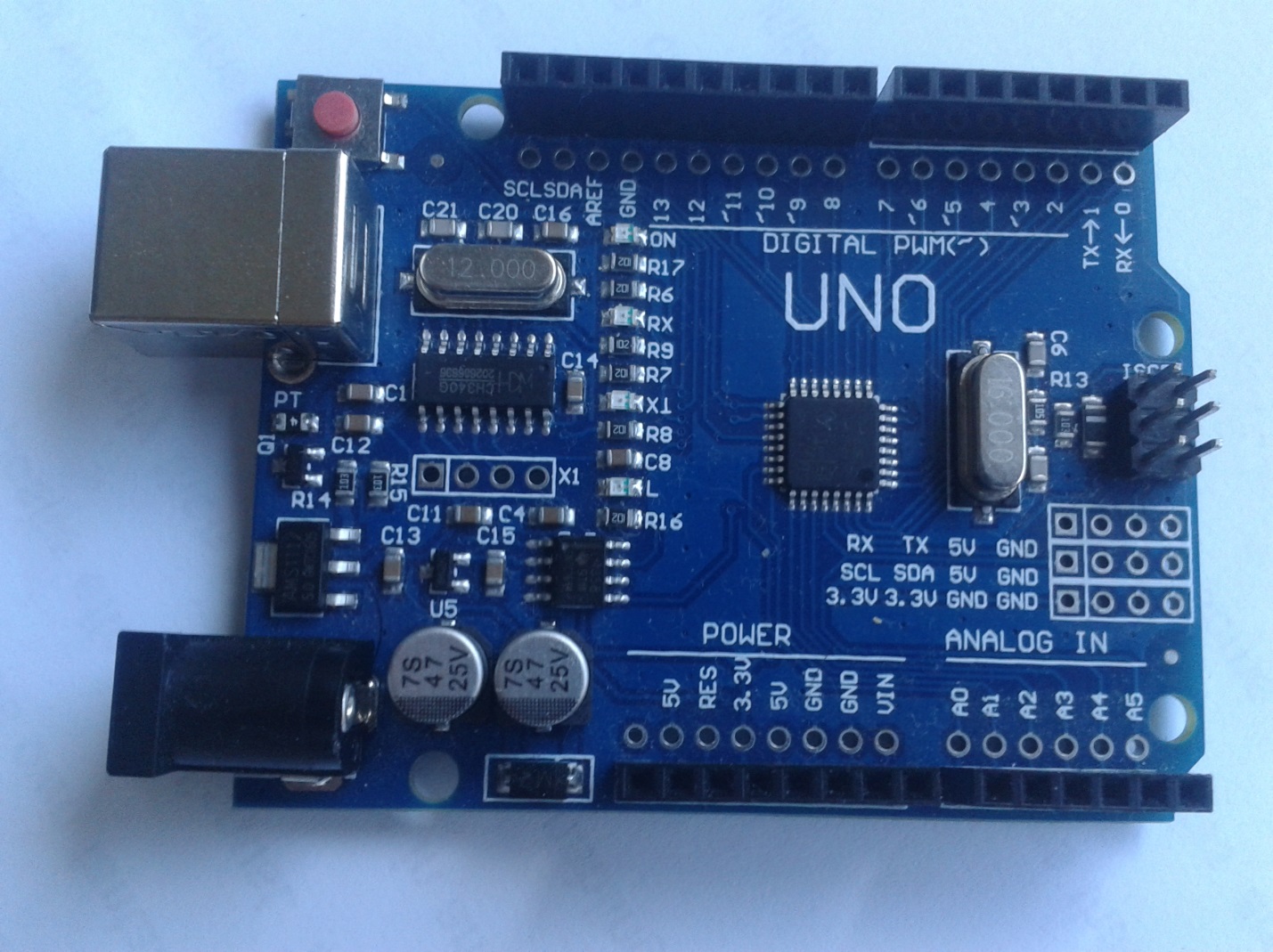
If the user clicks on one of the buttons for opening or closing the blinds, a dialog message is shown in the same way as when a failed connection message is shown. The message says either that the blinds are closed or open.



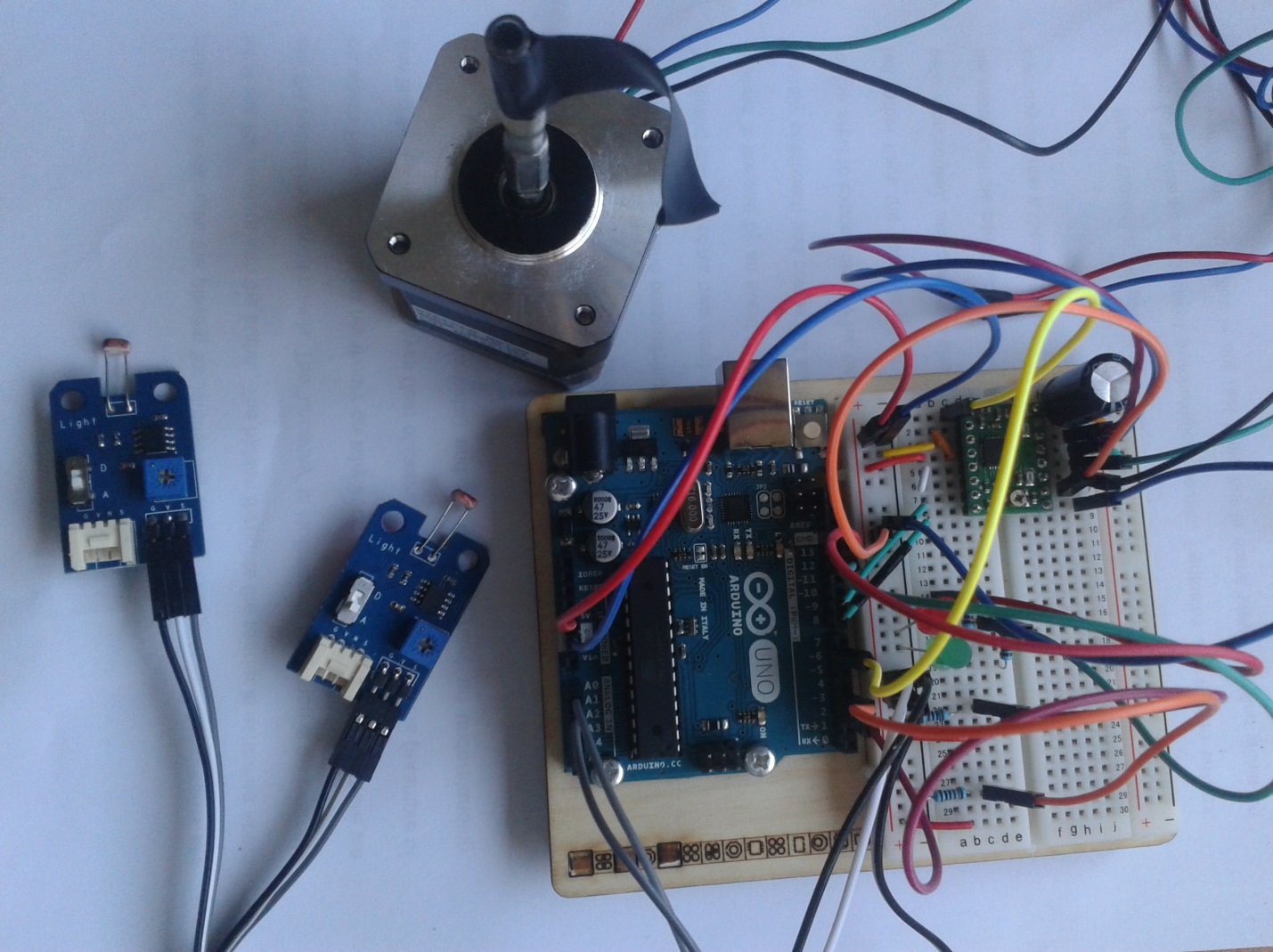
**The Pololu A4988 upside down along with the capacitor for the battery. I soldered it like that and then realized that the names of the pins are on the other side. This did not stop me though.**



**The light sensors (photo resistors) labeled the same way as in the code for the Arduino (pink and green)**

****

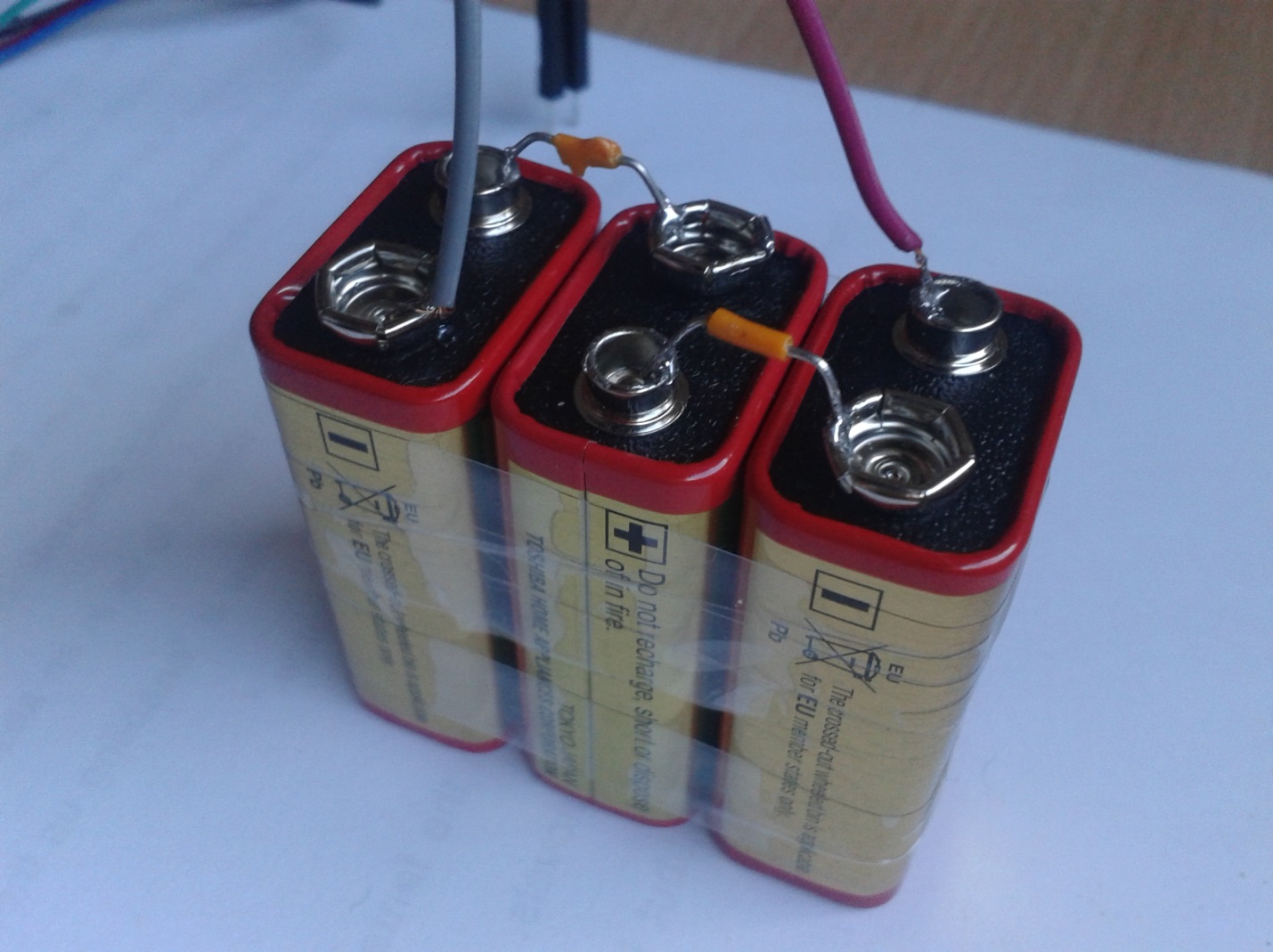
**The Chinese replica of the Arduino Uno board with the CH340G chip**

****

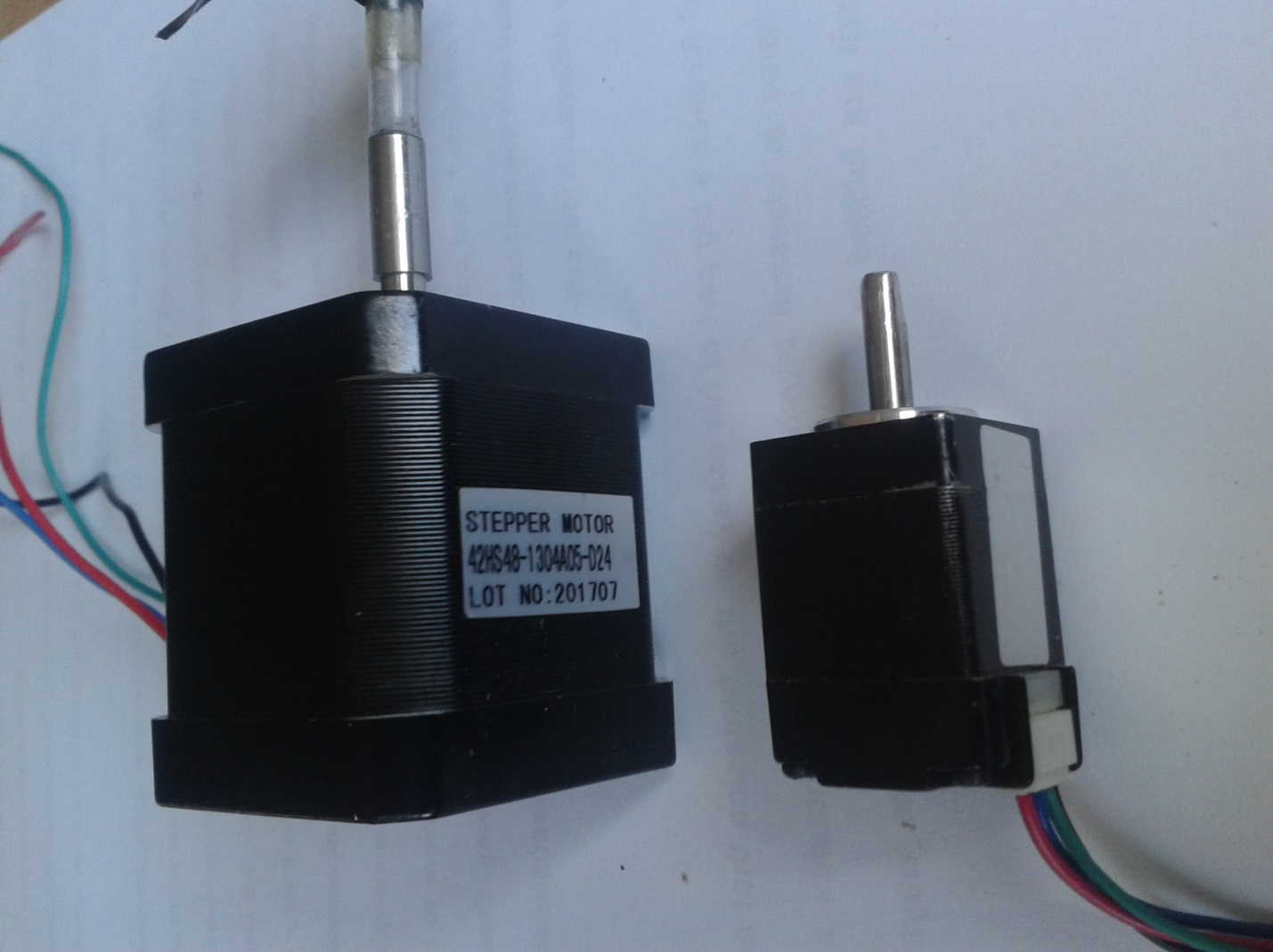
**The whole project including the Arduino Uno board, breadboard, the stepper motor, the stepper motor driver and the light sensors**

****

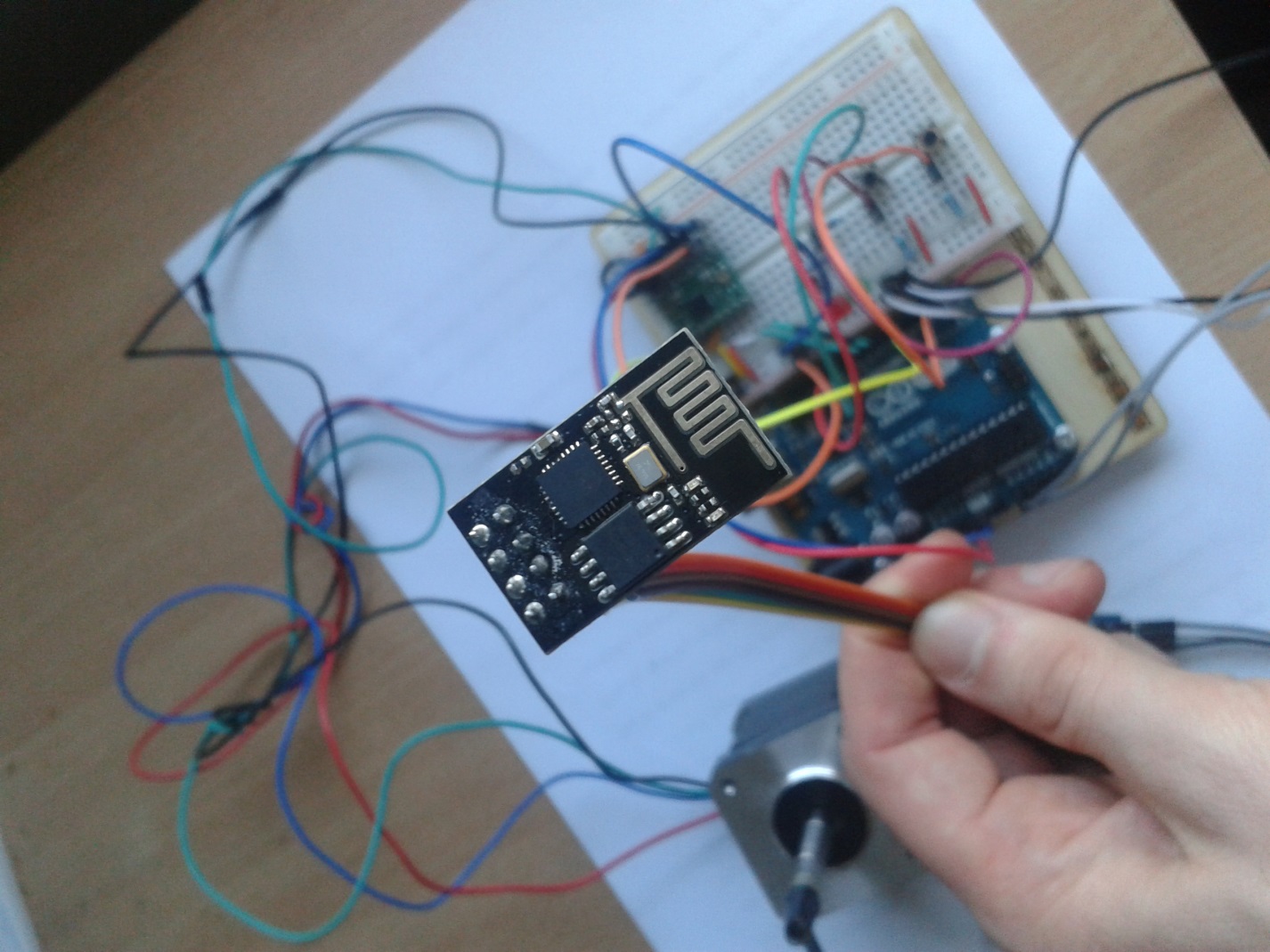
**The stepper motor**

****

**Three 9 volt batteries which sum up to a total of 27 volts that are powering the stepper motor**

****

**The stepper motor that I am using for my project with its’ smaller brother**

****

**The ESP8266 Wi-Fi module**

5. Results and Conclusion

With the project I achieved gaining understanding about how the create an Android application. It was the first time I am doing something like this and now I feel more comfortable about it.

I also understood some concepts of internet connections and the sending of data between two devices. I understand and have the knowledge to send messages between very simple devices such as the ESP8266.

It all worked well until the ESP8266 decided to stop accepting code uploaded to it. I am still happy with the result I accomplished by programming those small devices. I started this project with some doubt about this hardware and the hardware failed me. I learned a lesson to not be too dependent on hardware.

A missing feature in this project is the fixed connection between the ESP8266 and the Android smart phone.

I have learned that some devices need very stable voltage current. I learned also that a capacitor is needed to keep a current stable because it works as if a small battery.

The knowledge I got from this project I can use to establish connection between two devices. I can also better understand circuits of electronic elements. I am able to operate a stepper motor controlled by software. A future project might be creating of a 3d printer. With the knowledge I gained from this project I now know that a 3d printer consists of stepper motors which I can already easily control.

I learned how to upload code on a board and use software to control a piece of hardware. I find this useful in my future development since I got my hands on something complex enough to make me wonder through it with days.

6. Resources

1. Summers, L. (2014). Using buttons and switches with an Arduino. [online] connectedly.com. Available at: <http://www.connectedly.com/using-buttons-and-switches-arduino>
2. Evans, B. (2015). *Arduino Programming Notebook*. 1st ed. [ebook] Ca, USA: Creative Common, pp.Page 7-8. Available at: <http://playground.arduino.cc/uploads/Main/arduino_notebook_v1-1.pdf>
3. Instructables website - [http://www.instructables.com](http://www.instructables.com/id/Using-ESP8266-SPIFFS/)
4. The official website of the ESP8266 Wi-Fi module - <http://esp8266.com>
5. Github repository with information about the ESP8266 and instructions on how to connect it to Arduino - <https://github.com/esp8266/Arduino>
6. Arduino forum - <https://forum.arduino.cc>
7. Android developers website - <https://developer.android.com/index.html>
8. Java Oracle website - <https://docs.oracle.com>