BT Blinds

5/30/2022

Yassin Kortam

Dr. A. H. Tabrizi

Table of Contents

Introduction

The design goal of this device is to control window blinds via a mobile device using available hardware.

‘

Equipment

- An Adafruit ESP 32 Feather
- A 28YBJ-48 DC 5V stepper motor
- A ULN2003 Driver Board
- A Micro USB cable
- A power switch
- A Creality Ender 3 pro 3D printer
- 22 AWG wire
- A soldering kit
- A hot glue gun

Hardware

Electronic Components

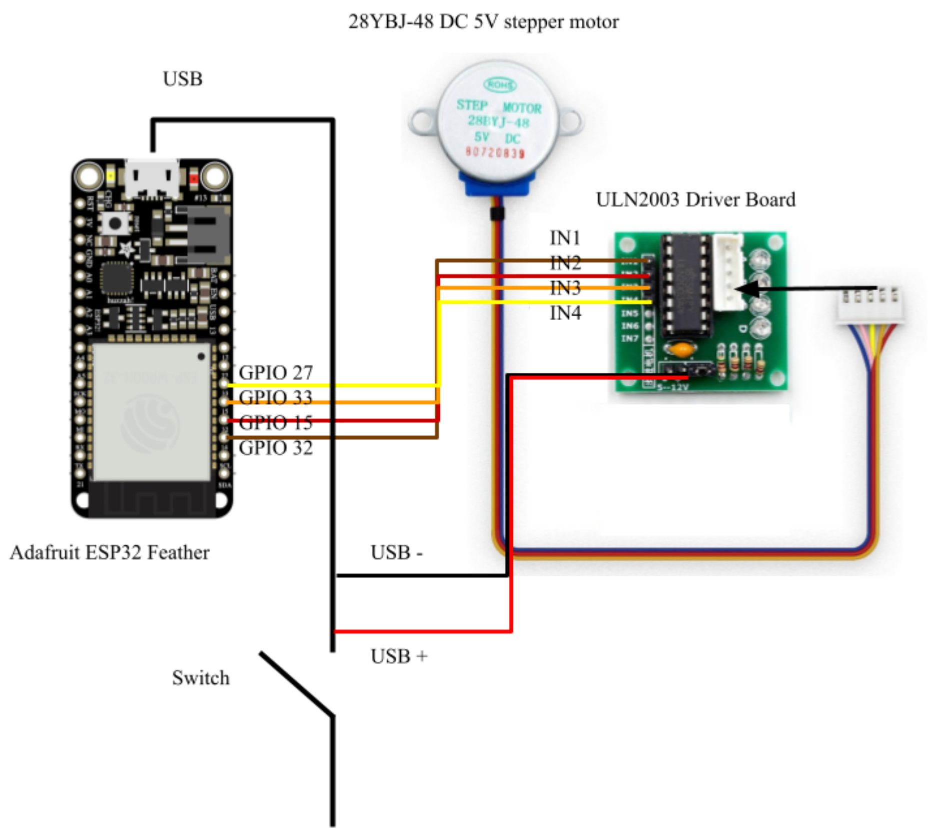


Figure 1: *Blinds device Wiring diagram with labeled components*

First, a Y-connector is made using a micro USB cable and 2 22 AWG wires with female pin headers, splitting the ground and power to supply both the ESP32 and the driver board. A switch is then added to the lower third of the USB cable, splicing the power wire. The female pin headers and male micro USB connector are connected to the driver board and ESP32 respectively.

Next, four 22 AWG wires with female to female pin headers connect GPIO pins 27, 33, 15, and 32 of the ESP32 to input pins 4, 3, 2, and 1 respectively.

Finally, the motor ribbon is connected to the driver board.

Structural Components

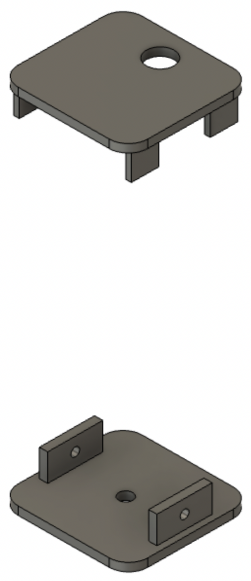


Figure 2: *Top and bottom lids for the blinds device*

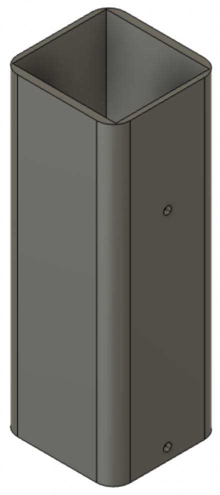


Figure 3: *Main housing for the blinds device (hollow rectangle form factor)*

The top and bottom lids (figure 2) have holes for the stepper motor coupling and the usb cable respectively. They both attach to the main housing via friction fit, with the bottom lid having additional screw holes.

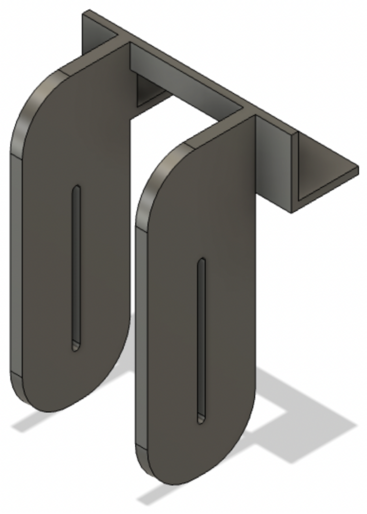


Figure 4: *Blinds device mount (glued to the blinds)*

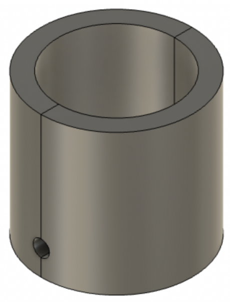


Figure 5: *Power switch casing (glued)*

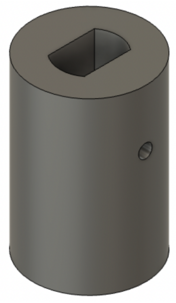


Figure 6: *Stepper motor coupling (friction fit onto motor and glued to hook)*

The mount (figure 4) allows for vertical adjustment of the blinds device to fit a broad range of blinds. The motor is then coupled to the blinds via a hook hot glued to a 3D printed attachment (figure 6).

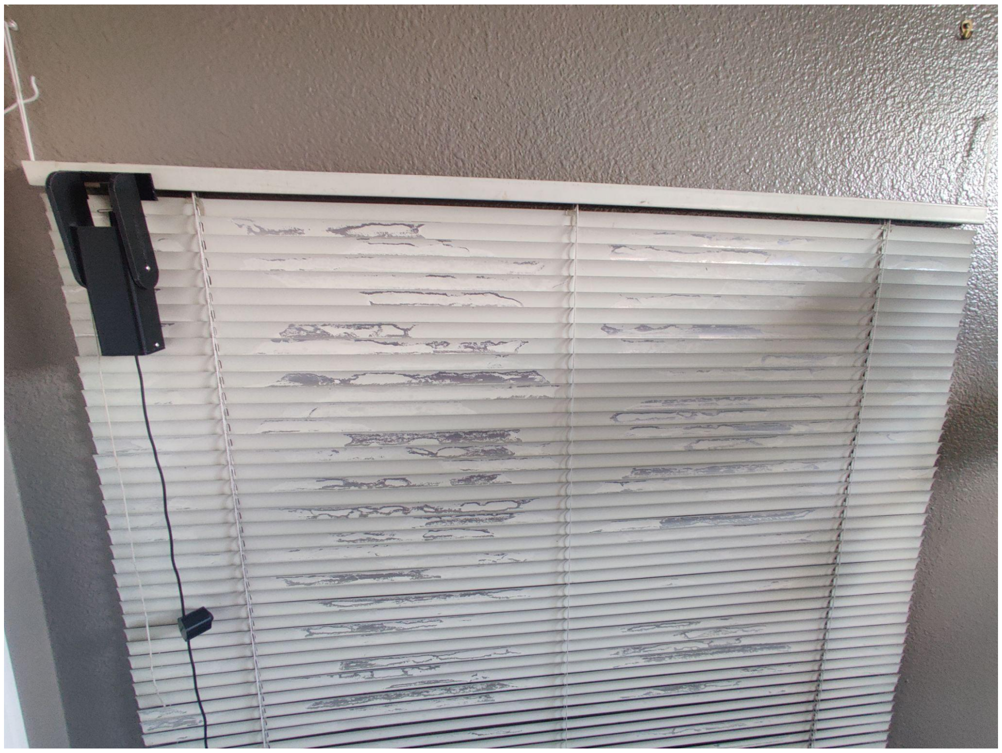


Figure 7: *The fully assembled blinds device installed on blinds.*

Software

Arduino Code

const.h

```
//******PIN DEFINITIONS******
#define IN1 32 //Brown wire
#define IN2 15 //Red wire
#define IN3 33 //Orange wire
#define IN4 27 //Yellow wire

//******MOTOR DEFINITIONS****
#define steps_per_revolution 512

//******BLINDS DEFINITIONS***
#define turn_precision 0.05
```

Const.h contains hardware specific constants: pin definitions, steps per each revolution for the stepper motor used, and a numeric multiplier that indicates the smallest step size possible for smooth, continuous motion (experimentally determined).

config.h

```
/************************ Adafruit IO Config *******************************/

// visit io.adafruit.com if you need to create an account,
// or if you need your Adafruit IO key.
#define IO_USERNAME "example"
#define IO_KEY "example"

/******************************* WIFI **************************************/

// the AdafruitIO_WiFi client will work with the following boards:
//   - HUZZAH ESP8266 Breakout -> https://www.adafruit.com/products/2471
//   - Feather HUZZAH ESP8266 -> https://www.adafruit.com/products/2821
//   - Feather HUZZAH ESP32 -> https://www.adafruit.com/product/3405
//   - Feather M0 WiFi -> https://www.adafruit.com/products/3010
//   - Feather WICED -> https://www.adafruit.com/products/3056
//   - Adafruit PyPortal -> https://www.adafruit.com/product/4116
//   - Adafruit Metro M4 Express AirLift Lite ->
//   https://www.adafruit.com/product/4000
//   - Adafruit AirLift Breakout -> https://www.adafruit.com/product/4201
//   - Adafruit AirLift Shield -> https://www.adafruit.com/product/4285
//   - Adafruit AirLift FeatherWing -> https://www.adafruit.com/product/4264

#define WIFI_SSID "example"
#define WIFI_PASS "example"

// uncomment the following line if you are using airlift
// #define USE_AIRLIFT

// uncomment the following line if you are using winc1500
// #define USE_WINC1500

// comment out the following lines if you are using fona or ethernet
#include "AdafruitIO_WiFi.h"

#if defined(USE_AIRLIFT) || defined(ADAFRUIT_METRO_M4_AIRLIFT_LITE) ||         \
   defined(ADAFRUIT_PYPORTAL)
// Configure the pins used for the ESP32 connection
#if !defined(SPIWIFI_SS) // if the wifi definition isnt in the board variant
// Don't change the names of these #define's! they match the variant ones
#define SPIWIFI SPI
#define SPIWIFI_SS 10 // Chip select pin
#define NINA_ACK 9    // a.k.a BUSY or READY pin
#define NINA_RESETN 6 // Reset pin
#define NINA_GPIO0 -1 // Not connected
#endif
AdafruitIO_WiFi io(IO_USERNAME, IO_KEY, WIFI_SSID, WIFI_PASS, SPIWIFI_SS,
                  NINA_ACK, NINA_RESETN, NINA_GPIO0, &SPIWIFI);
#else
AdafruitIO_WiFi io(IO_USERNAME, IO_KEY, WIFI_SSID, WIFI_PASS);
#endif
/******************************* FONA **************************************/

// the AdafruitIO_FONA client will work with the following boards:
//   - Feather 32u4 FONA -> https://www.adafruit.com/product/3027

// uncomment the following two lines for 32u4 FONA,
// and comment out the AdafruitIO_WiFi client in the WIFI section
// #include "AdafruitIO_FONA.h"
// AdafruitIO_FONA io(IO_USERNAME, IO_KEY);

/**************************** ETHERNET ************************************/

// the AdafruitIO_Ethernet client will work with the following boards:
//   - Ethernet FeatherWing -> https://www.adafruit.com/products/3201

// uncomment the following two lines for ethernet,
// and comment out the AdafruitIO_WiFi client in the WIFI section
// #include "AdafruitIO_Ethernet.h"
// AdafruitIO_Ethernet io(IO_USERNAME, IO_KEY);
```

Config.h is used to initialize an internet connection and adafruit IO. This feature was not used in the device due to it leading to an excessively large program size (the ESP32 board has a limited flash storage capacity).

blinds.h

```
#include <Stepper.h>
#include "const.h"

int revolutions_per_blind = 8;
int max_steps = steps_per_revolution*revolutions_per_blind;
int steps_per_increment = steps_per_revolution*turn_precision;
int steps = 0;

void open_blinds();
void close_blinds();
void step_open_blinds();
void step_close_blinds();

void open_blinds(Stepper motor){
   if (steps < max_steps){
       motor.step(max_steps - steps);
       steps = max_steps;
   }
}

void close_blinds(Stepper motor){
   if (steps > 0){
       motor.step(-steps);
```

```
        steps = 0;
    }
}

void step_open_blinds(Stepper motor){
   if (steps < max_steps){
        motor.step(steps_per_increment);
        steps += steps_per_increment;
    }
}

void step_close_blinds(Stepper motor){
    if (steps >= steps_per_increment){
        motor.step(-steps_per_increment);
        steps -= steps_per_increment;
    }
}
```

Blinds.h contains two categories of functions that control the stepper motor: step open/close and open/close. Step open/close moves the stepper motor in small increments- ”steps_per_increment” -determined by multiplying the steps per revolution by the multiplier in const.h. The open/close functions open or close the blinds completely. None of these functions are implemented due to memory limitations; position variables are wiped when the device is turned off and the flash storage will be destroyed if it is written to frequently.

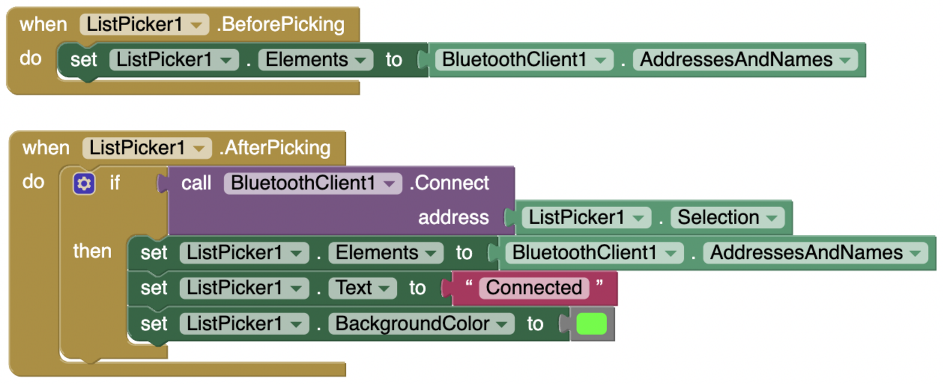
main.cpp

```
#include <Arduino.h>
#include <SPI.h>
#include "BluetoothSerial.h"
#include "const.h"
#include "blinds.h"
#include "config.h"

Stepper motor(steps_per_revolution,IN1,IN3,IN2,IN4);

BluetoothSerial SerialBT;

//Object to send position data to adafruit io
//AdafruitIO_Feed *position = io.feed("position");
```

```
void setup() {

 Serial.begin(9600);
 SerialBT.begin("Yassin_Blinds");

 pinMode(IN1, OUTPUT);
 pinMode(IN2, OUTPUT);
 pinMode(IN3, OUTPUT);
 pinMode(IN4, OUTPUT);

 motor.setSpeed(60);

 /*
 // connect to io.adafruit.com
 Serial.print("Connecting to Adafruit IO");
 io.connect();

 // wait for a connection
 long connect_time = 0;
 long start = millis(), finish;
 while(io.status() < AIO_CONNECTED) {
   Serial.print(".");
   finish = millis();
   connect_time = finish - start;
   if (connect_time > 4000){
     break;
   }
 }
 if(io.status() < AIO_CONNECTED){
   Serial.print("No WiFi");
 }
 else{
   Serial.print("Connected");
 }
 */
}

void clearBuffer(){
 while(SerialBT.read() != -1){
   SerialBT.read();
 }
}

void loop() {
/*
//maintain io connection
 if(io.status() < AIO_CONNECTED){
   io.run();
 }
*/
 int cmd = SerialBT.read();

 if (cmd != -1){
   Serial.println(cmd);
 }
 if(cmd == 113){
   while(cmd != 114){
     motor.step(steps_per_increment);
     //step_open_blinds(motor);
     cmd = SerialBT.read();
   }
 }
 if(cmd == 119){
   while(cmd != 114){
     motor.step(-steps_per_increment);
     //step_close_blinds(motor);
     cmd = SerialBT.read();
   }
 }
 if(cmd == 49){
   open_blinds(motor);
   clearBuffer();
 }
 if(cmd == 50){
   close_blinds(motor);
   clearBuffer();
 }

 //upload position to adafruit io
 //position->save(steps);
}
```

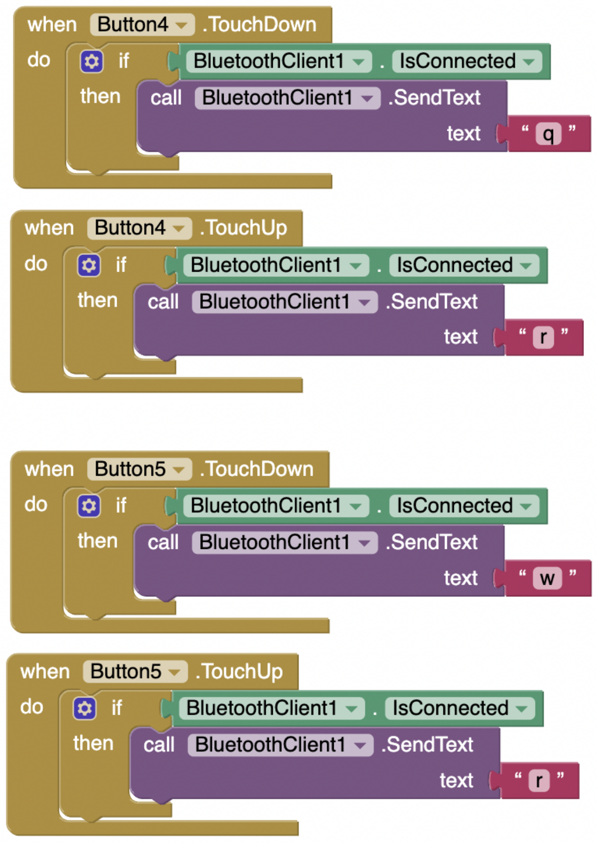
Main.cpp is the driver program for the device. After initialization, the program checks bluetooth serial for incoming ASCII character commands from a mobile device. Note only commands 113 and 119 are accessible via mobile device (commands 49 and 50 require the permanent storage of position data). Note that portions of the program related to adafruit IO IoT functionality are disabled due to the memory issues.

Mobile Phone App Code



```
when ListPicker1 .BeforePicking
do  set ListPicker1 . Elements to  BluetoothClient1 . AddressesAndNames

when ListPicker1 .AfterPicking
do  if    call BluetoothClient1 .Connect
                              address  ListPicker1 . Selection
    then  set ListPicker1 . Elements to  BluetoothClient1 . AddressesAndNames
          set ListPicker1 . Text to  " Connected "
          set ListPicker1 . BackgroundColor to 🟩
```

Figure 8: *This portion of the program initializes the bluetooth connection to the device given a user input from a drop down menu.*



```
when Button4 .TouchDown
do  if    BluetoothClient1 . IsConnected
    then  call BluetoothClient1 .SendText
                               text  " q "

when Button4 .TouchUp
do  if    BluetoothClient1 . IsConnected
    then  call BluetoothClient1 .SendText
                               text  " r "

when Button5 .TouchDown
do  if    BluetoothClient1 . IsConnected
    then  call BluetoothClient1 .SendText
                               text  " w "

when Button5 .TouchUp
do  if    BluetoothClient1 . IsConnected
    then  call BluetoothClient1 .SendText
                               text  " r "
```

Figure 9: *This portion of the program sends commands to the blinds device in the form of ASCII characters depending on button presses.*

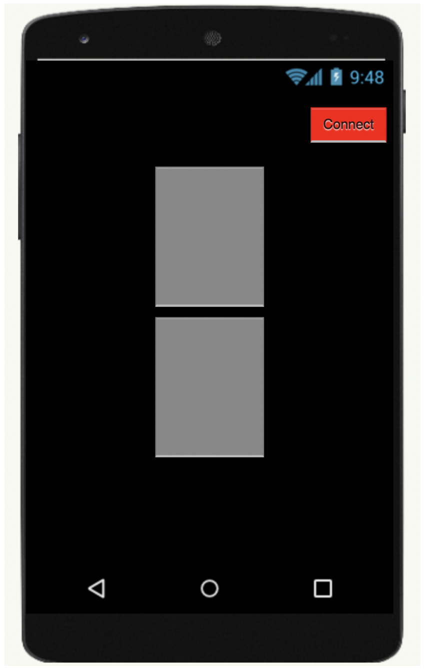


Figure 8: *Mobile phone app user interface. The two gray buttons (top and bottom) open and close the blinds respectively while they are pressed. The connect button changes color depending on the bluetooth connection to the device.*