The Motorized Window Blind

ME 102B Project Report
Charlie Xu

Table of Content

I. Motivation

The product of my project is a motorized window blind. This project fits into the goal of converting my room into a smart room. Currently, I have automated the humidifier, projector, stereo speakers, and all the lights in my room. While there are smart window blinds options for purchase online, they are not customizable and expensive, requiring smart hubs and other accessories. After learning more about motors, drive trains, and C++ in ME 102B, I decided to use the mini-project to motorize my window blinds.

II. Product Description

For my motorized window blind, the top functionality is to use a motor to execute the opening and closing of my window blinds. The design includes three methods of control. First, users can use voice control to command the window blinds. This serves as the primary method of controlling the blinds. It is the most convenient. Second, the users can press the button to change the state of the window blinds. This mostly serves as a backup option for when voice control is not available. Last, the blinds open at 9 AM and close at 5 PM. The specific time can only be adjusted in the backend code. The morning blinds opening can help me get up if there is sunshine on a particular day. The evening blinds closing helps improve the brightness of the room as my room has minimal lighting to reduce energy consumption. The 2 integrated designs can be seen below in Figures 1 and 2. The final implementation is in figure 2; the reason will be explained in the mechanical setup section.

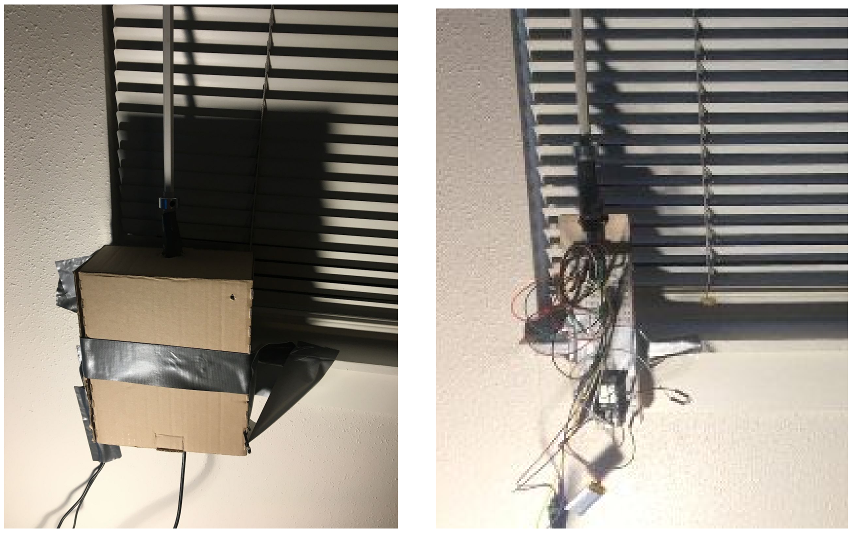

Figure 1&2. My motorized window blind with (Left) and without a box (Right).

III. Mechanical Design

<u>Motor and Shaft</u>

The motor is mounted on a wood panel along with the breadboard. A collar is attached to the motor output shaft, and a different collar is attached to the wand of the blind. Connecting these two parts is a connector made of T-Rex duct tape. One side of the T-Rex duct tape wraps around the motor collar; the other side goes between the wand collar and the wand. These designs can be seen in Figure 3.

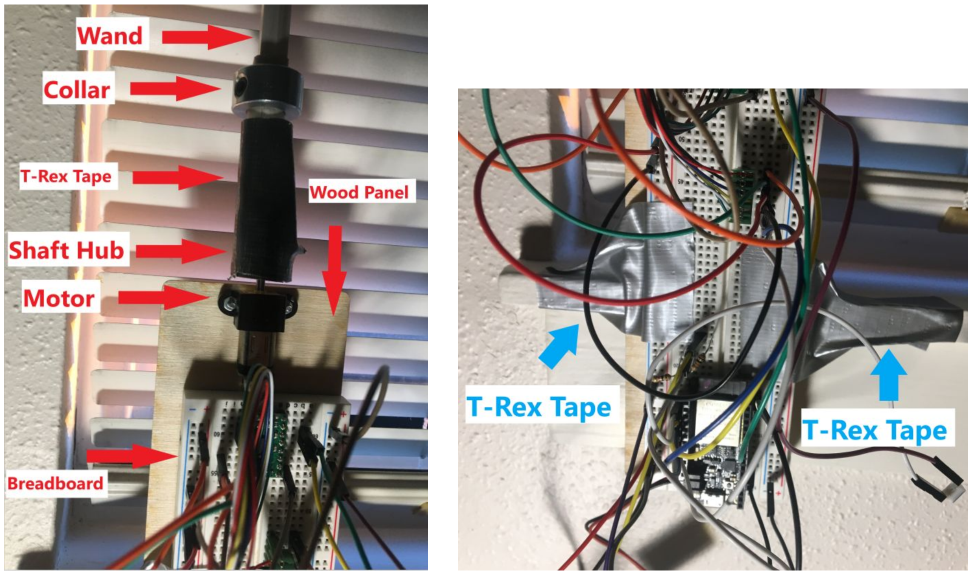

Figure 3&4. Motor and shaft attachments(Left) and wood panel Attachments (Right).

Wood Panel Housing

The wood panel is attached to the wall via T-Rex Tape as seen in Figure 4. This is the final housing selected. One iteration of the housing had the entire wood panel mounted inside a cardboard box; the box was then mounted on to the wall by T-Rex Tape as seen in Figure 1. This setup didn't work well. Since the tapes peeled off much easier than on cardboard than on wood and plastic, the box was not very secure. On top of that, the increased weight and moment caused by the box increased the force the adhesives needed to hold. If Jacobs were to be available, I would've made the box out of acrylic, which bonds with tape very well, using finger-joint designs. If I could modify the wall, I would've mounted the box on the wall using drywall anchors.

IV. Circuit

The circuit of the motorizes window blind interfaces 4 sensors to determine the states and transitions. These are motor encoder, button, smart relay, and wifi clock.

Encoder

The motor encoder records the number of counts as the motor rotates. This count is used whether the blind reached the open position, transitioning to a different state. It also stops the motor from turning the wand beyond the mechanical stop, stalling the motor.

User Interfaces

The button is used to change the blind's state by the user. It requires a pull resistor which is set internally in the code. The smart relay is a voice-controlled device connected to Amazon Alexa. It is also used to change the state of the blind's state by the user. A voltage divider is used to bring the 5v supply to the input range of the ESP-32.

Wifi Clock

The wifi clock is an integrated "sensor" inside the ESP32 controller. It fetches the local time from the server, which is used to determine whether it is time to open or close the blind.

Other Components

The detailed circuit diagram is shown in figure 5. One other part that was not mentioned is the motor driver. It is used to change the direction of the motor rotation and set up to be fast decay.

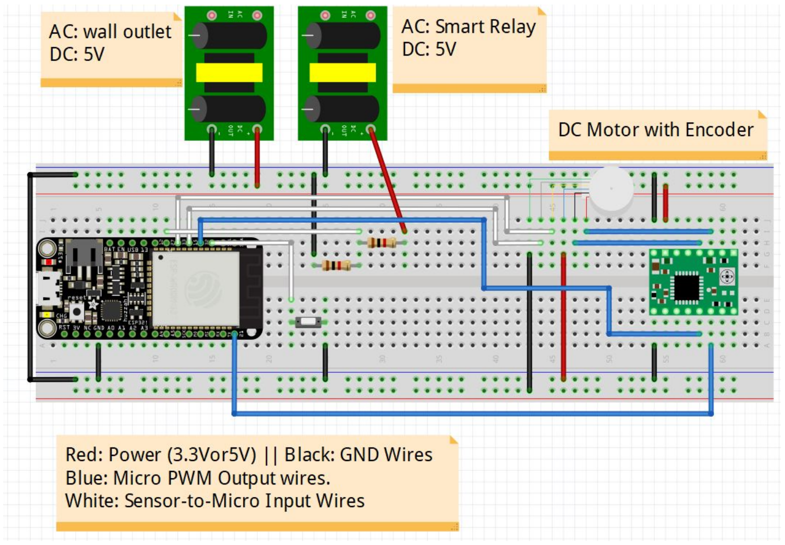
AC: wall outlet
DC: 5V
AC: Smart Relay
DC: 5V
DC Motor with Encoder
Red: Power (3.3Vor5V) || Black: GND Wires
Blue: Micro PWM Output wires.
White: Sensor-to-Micro Input Wires

Figure 5. Fritzing circuit diagram of the motorized window blind.

V. Finite State Machine

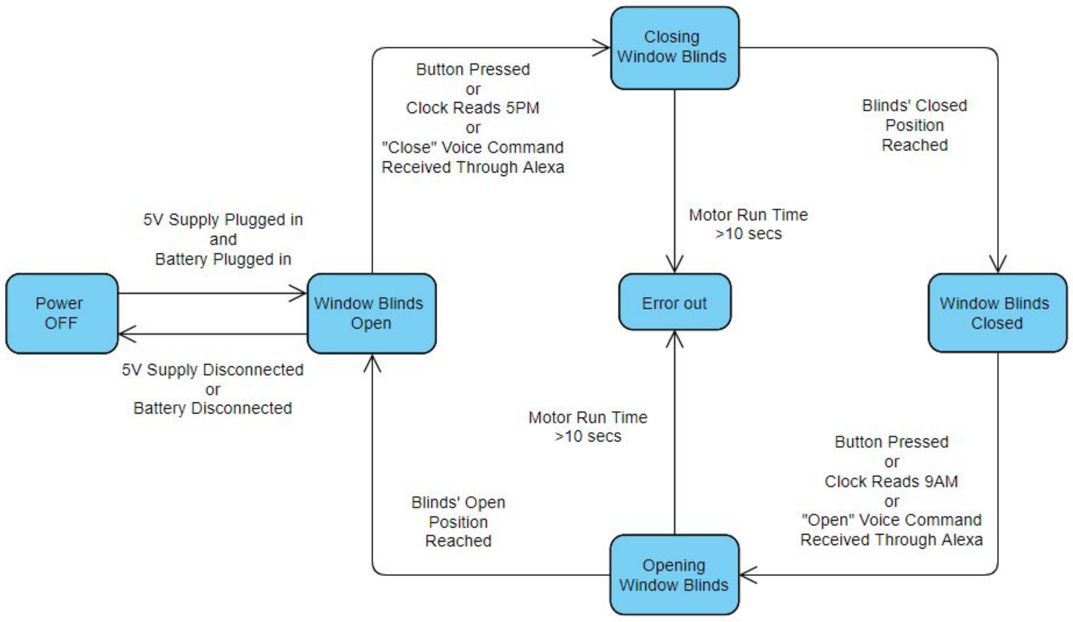
Closing Window Blinds
Button Pressed or Clock Reads 5PM or "Close" Voice Command Received Through Alexa
Blinds' Closed Position Reached
5V Supply Plugged in and Battery Plugged in
Motor Run Time >10 secs
Power OFF
Window Blinds Open
Error out
Window Blinds Closed
5V Supply Disconnected or Battery Disconnected
Motor Run Time >10 secs
Button Pressed or Clock Reads 9AM or "Open" Voice Command Received Through Alexa
Blinds' Open Position Reached
Opening Window Blinds

Figure 6. Finite State Diagram for the motorized window blind.

<u>Single Direction Loop</u>

The four main states form an irreversible loop in FSD as seen in Figure 6. This is designed to avoid the wand being turn beyond the mechanical stops. Since each dynamic state (closing or opening the blinds) is associated with a particle motor rotation direction and relative rotational position, going backward in the state loop would result in motor turning beyond the mechanical stops. Therefore, the loop is single-directional.

<u>Single-path State Transition and Stall Timeout</u>

The transition from dynamic states (motor on) to static states (open or closed) has only one condition which is reaching the desired position, as seen in Figure 6. This means that the dynamic states are also pass-by states where the machine doesn't linger in unless some failure happens. A stall timeout is also implemented to cut the motor power if it takes too long to reach the desire position. It sends the machine to an error state where it will not respond to any other code and light up the LED on the ESP-32. This tells the user that an error happened.

<u>Multi-path State Transitions</u>

As mentioned in the previous part, there are three methods to change the state of the blind. As seen in Figure 6, the transition from the static states to the dynamic states can be triggered by 3 different conditions. The 3 conditions are separated by "if, else if" statements to ensure only one is triggered at a time. The relay indicates the desired states (open or closed) instead of a state change. Therefore, if the state would be locked in if a simple "if, else if". The remedy is to first detect there is a change in relay condition and only trigger a state change when that happens.

<u>Arduino IDE Implementation</u>

This can be seen in Appendix I.

Appendix

I. Arduino IDE code

See next page.

```
#include <Arduino.h>
#include <ESP32Encoder.h>
#include <WiFi.h>
#include <NTPClient.h>
#include <WiFiUdp.h>

// Universial Variables
  int state = 1; //state used for state transition
  int AIN1 = 21; //Initialize AIN1 Pin
  int AIN2 = 32; ////Initialize AIN2 Pin

// Encorder Variables
  ESP32Encoder encoder;
  int currentCount = 0;

//Timer Interrupt variable setup
  hw_timer_t * timer = NULL;
  volatile SemaphoreHandle_t timerSemaphore;
  portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;
  volatile uint32_t isrCounter = 0; //counter for # of interrupts
  //Timer Interrupt functions
  void IRAM_ATTR onTimer(){
    // Increment the counter
    portENTER_CRITICAL_ISR(&timerMux);
    isrCounter++;
    portEXIT_CRITICAL_ISR(&timerMux);
    // Give a semaphore that we can check in the loop
    xSemaphoreGiveFromISR(timerSemaphore, NULL);
  }

// Alexa Variables & Functions
  int alexaPin = 27;
  int alexaVal = 0;
  int oldAlexaVal = 0;

// Transition 1 Change Motor Direction Variables & Functions
  // Debounce Variables (Max one press per 2 seconds )
  unsigned long lastDebounceTime = 0;
  unsigned long debounceDelay = 2000000;
  //setup Buttons
  struct Button {
      const uint8_t PIN;
      uint32_t numberKeyPresses;
      bool pressed;
  };
  Button MotorButton = {14, 0, false};//Initialize Button

  //GPIO Interrupt functions for the button with debouncing
  void IRAM_ATTR isr(void* arg) {
      Button* s = static_cast<Button*>(arg);
      s->numberKeyPresses += 1;
      s->pressed = true;
  }
  void IRAM_ATTR isr() {
      //debouncing
      if ((micros() - lastDebounceTime) > debounceDelay){
         MotorButton.numberKeyPresses += 1;
         MotorButton.pressed = true;
      }
      lastDebounceTime = micros(); //reset timer
  }

// Clock Variables
  // Replace with your network credentials
  const char* ssid     = "PorQueFi";
  const char* password = "CanIUseTheRoomFor20Minutes?";
  // Define NTP Client to get time
  WiFiUDP ntpUDP;
  NTPClient timeClient(ntpUDP);

  // Variables to save date and time
  String formattedDate;
  String timeStamp;
  String dawn = "9:00:00";
  String dusk = "17:00:00";

// Error Conditions:
  int startTime =0;
  int currentTime =0;

//Setup loop
void setup() {
// Universal Setup
  Serial.begin(115200);

// Wifi and clock setup
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  // Print local IP address and start web server
  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

// Initialize a NTPClient to get time
```

```
  timeClient.begin();
  timeClient.setTimeOffset(-28800);

//Counting Setup
  // Enable the weak pull up resistors
  ESP32Encoder::useInternalWeakPullResistors=UP;
  // Attache pins for use as encoder pins
  encoder.attachFullQuad(33, 15);
  // set starting count value after attaching
  encoder.setCount(0);
  // clear the encoder's raw count and set the tracked count to zero
  encoder.clearCount();


// Create semaphore to inform us when the timer has fired
  timerSemaphore = xSemaphoreCreateBinary();
  timer = timerBegin(0, 80, true); //Create a timer
  // Attach onTimer function to our timer.
  timerAttachInterrupt(timer, &onTimer, true);
  // Define Interrupt Condition for 2Hz
  // for 1Hz blink, turn on LED 2 of every 4 interrupts
  timerAlarmWrite(timer, 1000000, true);
  timerAlarmEnable(timer);  // Start an alarm

//forwards-motor-turning Setup
  ledcAttachPin(AIN1, 0);
  ledcSetup(0, 12000, 8);
  pinMode(AIN1, OUTPUT);

//Button Setup
  pinMode(MotorButton.PIN, INPUT_PULLUP);
  attachInterrupt(MotorButton.PIN, isr, FALLING);


//Backwards-motor-turning Setup
  ledcAttachPin(AIN2, 1);
  ledcSetup(1, 12000, 8);
  pinMode(AIN2, OUTPUT);

//Alexa Control setup
  pinMode(alexaPin, INPUT_PULLUP);
  alexaVal = digitalRead(alexaPin);
  oldAlexaVal = alexaVal;

// Error out setup
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
   // Clock
```

```
if (xSemaphoreTake(timerSemaphore, 0) == pdTRUE){
  while(!timeClient.update()) {
    timeClient.forceUpdate();
  }

  // Extract date
  formattedDate = timeClient.getFormattedDate();
  int splitT = formattedDate.indexOf("T");
  // Extract time
  timeStamp = formattedDate.substring(splitT+1, formattedDate.length()-1);
  //Serial.print("HOUR: ");
  //Serial.println(timeStamp);
}
switch (state) {
  case 1:{
    // Idle state when the window blinds is open
    // When the button is pressed switch to state2
    ledcWrite(0, 0);
    ledcWrite(1, 0);
    encoder.clearCount();
    alexaVal = digitalRead(alexaPin);

    if (MotorButton.pressed) {
      MotorButton.pressed = false;
      state = 2;
      startTime =millis();
    }
    else if (timeStamp == dusk){
      state = 2;
      startTime =millis();
    }
    else if (alexaVal!=oldAlexaVal){
      if (alexaVal==0){
        state = 2;
        startTime =millis();
      }
    }
    oldAlexaVal =alexaVal;
    break;
   }
  case 2 :{
    // Actuation state closing window blinds
    ledcWrite(1, 0);
    ledcWrite(0, 255);

    // After reaching closed position, switch to state3
    currentCount = (int32_t)encoder.getCount();
    Serial.println(currentCount);
    if (abs(currentCount)>4000) {
      state = 3;
      }
      currentTime =millis();
      if ((currentTime-startTime)>10000){
        ledcWrite(0, 0);
        ledcWrite(1, 0);
        state =5;
      }
      break;
     }
    case 3:{
      // Idle state when the window blinds is closed
      ledcWrite(0, 0);
      ledcWrite(1, 0);
      encoder.clearCount();
      alexaVal = digitalRead(alexaPin);

      // When the button is pressed switch to state4
      if (MotorButton.pressed) {
        MotorButton.pressed = false;
        state = 4;
        startTime =millis();
      }
      // When it's 9AM open the blinds
      else if (timeStamp == dawn){
        state = 4;
        startTime =millis();
      }
      else if (alexaVal!=oldAlexaVal){
        if (alexaVal==1){
          state = 4;
          startTime =millis();
        }
      }
      oldAlexaVal =alexaVal;
      break;
     }
    case 4 :{
      //Actuation state opening window blinds
      ledcWrite(0, 0);
      ledcWrite(1, 255);

      // Switch back to state 1 when button is pressed
      currentCount = (int32_t)encoder.getCount();
      Serial.println(currentCount);
      if (abs(currentCount)>4000) {
        state = 1;
      }
      currentTime =millis();
      if ((currentTime-startTime)>10000){
        ledcWrite(0, 0);
        ledcWrite(1, 0);
        state =5;
      }
      break;
    }
    case 5 :{
      digitalWrite(LED_BUILTIN, HIGH);
    }
  }
}
```