The Curtain Assistant

By Shaobo WANG

General Description:

For a concurrent student trying to take courses synchronously from the other side of the world, I choose to stay up all night and reverse day and night. Usually, with a curtain and an alarm clock, I will be "safe and sound". However, if I forgot to draw the curtain, I would be woken up by the bright sunshine at noon. In order to maintain a healthy sleep cycle, I could really use a smart device to draw the curtain for me whenever the light is too bright. By keeping the control box fixed, one of the possible solutions to draw the curtain is to use a cable drive system. There are existed solutions that requires replacing the entire curtain rod which is neither economic nor convenient.

So I decided to design my own system into a compact box which can be held against the wall by the original adjustable curtain rod. The system uses the ESP32 in the lab kit which can communicate with our smartphone through Wi-Fi.

Figure 1: Curtain Assistant

Mechanical details:

Package: Both the motor and the control circuitry are assembled inside the 3D printed box. A Li-Po battery is used, and the system can also be charge through a 8.4V power supply. Since the MCU communicate through Wi-Fi, only the power cable comes out of the housing.

Installation: The 3D printed box is sandwiched between one end of the curtain rod and the wall. By adjusting the total length of the rod, the box is held against the wall by friction. The other roller for cable transmission is clamped to the rod after tensioning. The out-side ring of the curtain is attached to the cable. Currently, the light sensor is placed inside the room.

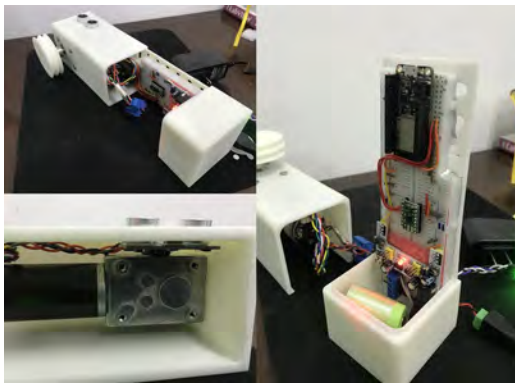

Figure 2: Curtain Assistant Package

Figure 3: Curtain Assistant Installation

Transmission: The BDC's nominal voltage is $6V$, and the nominal speed is 4000 RPM. A worm gear speed reducer with a 72: 1 gear ratio is placed at the output shaft, which also converts the rotational axis. The nominal torque is 0.8 N · m which is sufficient enough to overcome the friction of cable. A pulley is attached to the shaft through a flanged coupler with screws.

Figure 4: Curtain Assistant Transmission

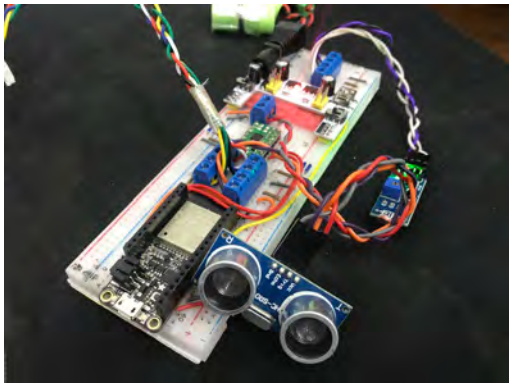

Figure 5: Curtain Assistant Circuitry

Circuit details:

Circuitry: The circuitry contains an ESP32 as both MCU and Wi-Fi communication module. A light sensor is used to test for lighting condition inside the room. An ultra-sonic sensor is used to measure distance in order to tell the location of the curtain. A motor driver is used to drive the BDC motor.

- ESP32: The microcontroller is included in the lab kit. Beyond the basic functions for an MCU, this module also serves as a perfect choice for IoT device. It can communication through Wi-Fi, which provides a useful wireless feature for my design. The Wi-Fi communication can be achieved through Blynk's library.
- Light sensor: The light sensor (photoresistor) module contains a photocell whose resistance change with lighting condition. The resistance which acts as a

voltage divider can be used to indicate light intensity. When the light gets more intensive, the resistance reduces, which results in the decrease of output voltage. Because there are conflicts between ADC#1 and the Wi-Fi library, analog pin with ADC#2 (i.e. A2) should be selected.

- Ultrasonic sensor: The ultrasonic sensor (HC-SR04) is used to measure distance. The sound wave is triggered with at least $10\mu s$' high-level signal. After the curtain reflects the sound wave, the time of flight gives out the distance. This sensor must works at $5V$, but the echo feedback signal is TTL level which is safe to operate with GPIOs of the MCU.
- Motor driver: The motor driver (DRV8833) is used to drive the BDC motor. The nominal voltage of the BDC motor is $6V$, and working at $5V$ still provides enough rotation speed for this application. By using the same voltage as the ultrasonic sensor, I can use only one voltage regulator module, which saves much space. Since GPIOs are not $5V$ tolerant, the motor driver is used for safer control.

One 2-s Li-Po battery ($7.4V$) is used as internal power which is always connected to the circuit with a DC connector. The battery has a second DC connecter for external power supply and charging through a $8.4V$, $1A$ power supply.

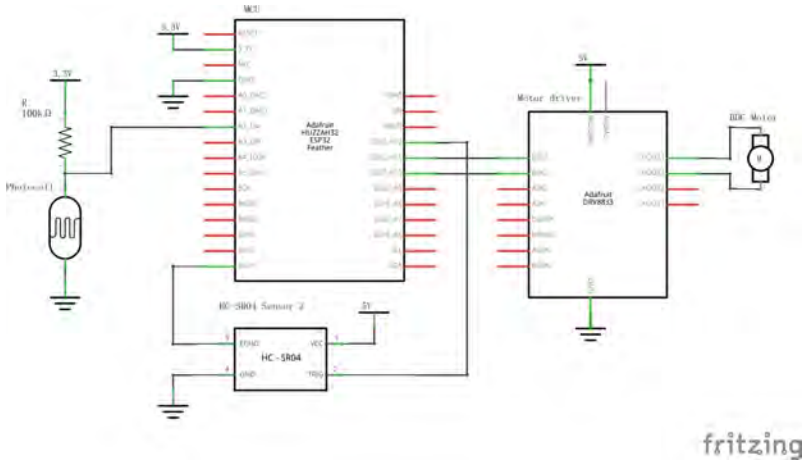

Figure 6: Curtain Assistant Schematic

User Interface design

The user interface is realized by Blynk which offers several functionable interface components that can communicate with the MCU through Wi-Fi. It is designed that user can set control mode by selecting from the virtual switch button, and the alarm clock or light threshold can also be set at this page.

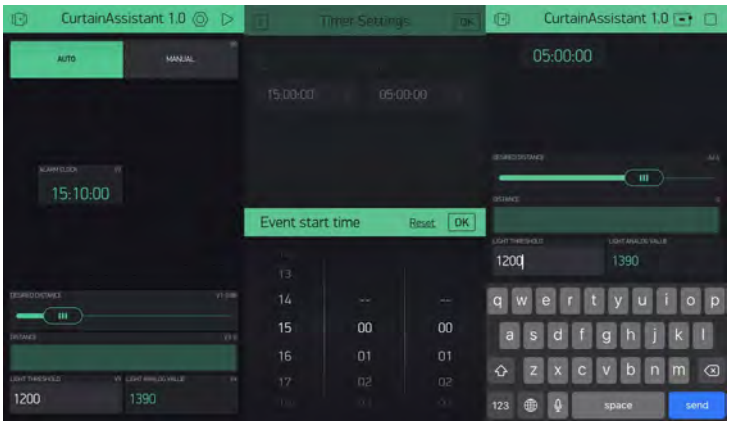

Figure 7: User Interface Design

Finite state machine

There are mainly two control mode: auto control and manual control. Each one of the mode can be changed to the other by sending a command through the smartphone app, which acts as a switch button. A time interval is set at the user interface platform which contains a clock alarm time and a set-to-auto time. During this time interval, the control mode will stay at manual control.

As for auto control, the curtain will respond to the distance feedback, the light intensity feedback and alarm clock feedback. When the light intensity exceeds the set threshold, the motor starts running forward until the curtain is fully closed. After the set clock alarmed, the motor starts running reversely until the curtain is fully opened, and the control mode is set to manual until the set-to-auto time or the user manually switch the mode into auto control. The control mode will automatically change according to the set time interval so that the user will not have to operate every time.

As for manual control, the curtain will respond to the distance feedback and the desired distance send from the user through the app. When the desired distance is different from the current distance, the motor starts running accordingly. After the desired distance is met, the motor stop running until either a new command is send through the phone, or the control mode is changed through the app.

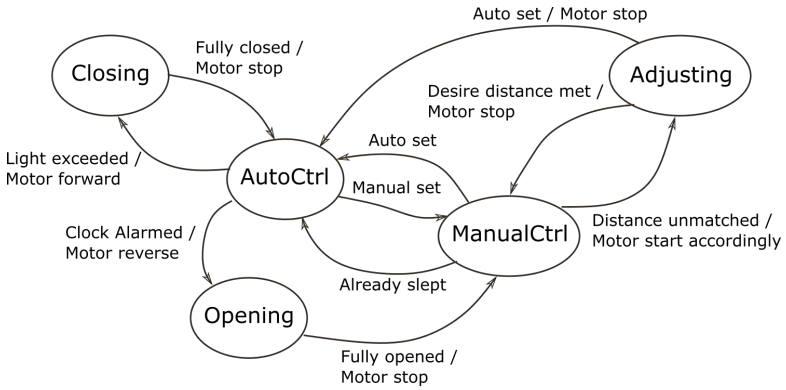

Figure 3: Curtain Assistant Finite State Diagram

Appendix: Code

```
#define BLYNK_PRINT Serial

#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>

/***Pin define***/
#define PHOTO_PIN A2
#define MOTOR_PIN1 12
#define MOTOR_PIN2 27
#define ENCODER_IN1 33
#define ENCODER_IN2 15
#define TRIG 13
#define ECHO 21
#define CTRL_PIN V0
#define DESDIS_PIN V1
#define ALARM_PIN V2
#define DIS_PIN V3
#define LA_PIN V4
#define LSH_PIN V5

#define isFastDecay true

char auth[] = "I-AnISZqWQ8viRk1uvPrhut66jRyCg0Y";
BlynkTimer blynkTimer;.
char ssid[] = "WHU-E6-427";
char pass[] = "88661188";

/***Constants***/
const unsigned int feedbackTimerInterval = 100000; // Refresh every 100ms
const int totalDis = 120; // 120cm total length
const int disThreshold = 5; // 5cm distance
const int debounce = 200;
/***Variables***/
// FSM
enum FSM_STATE {
  autoCtrl = 0,
  manualCtrl,
  curtainAdjusting,
  curtainOpening,
  curtainClosing
} currentState = autoCtrl;
enum MODE {
  setAuto = 1,
  setManual = 2
} currentMode = setAuto;

// Motor control
enum MOTOR_CTRL {
  forward = 0,
  reverse,
```

```
  halt
};
// Alarm clock
bool alarmClock = false;
// Ultrasonic
int desDis = 20; // Desired distance
int distance = 0; // Distance between curtain and wall
// Photocell
int lightAnalog = 0; // Light analog value
int lightThreshold = 1500; // Light threshold
// Data feedback
hw_timer_t * feedbackTimer = NULL;
portMUX_TYPE feedbackTimerMux = portMUX_INITIALIZER_UNLOCKED;
volatile unsigned int isrCounter_dataOutput = 0; // Dataoutput timer interrupt
// hysteresis
int lasttime = 0;

/***ISR function***/
void IRAM_ATTR onfeedbackTimer() {
  portENTER_CRITICAL_ISR(&feedbackTimerMux);
  isrCounter_dataOutput++;
  portEXIT_CRITICAL_ISR(&feedbackTimerMux);
}

/***Blynk function***/
BLYNK_WRITE(CTRL_PIN) {
  currentMode = (enum MODE)param.asInt();
  Serial.print("Control mode is: ");
  Serial.println(currentMode);
}

BLYNK_WRITE(DESDIS_PIN) {
  desDis = param.asInt();
  Serial.print("Desired distance is: ");
  Serial.println(desDis);
}

BLYNK_WRITE(ALARM_PIN) {
  alarmClock = param.asInt();
  Serial.print("Alarm clock is: ");
  Serial.println(alarmClock);
}
void blynkTimerEvent() {
  Blynk.virtualWrite(CTRL_PIN, currentMode);
  Blynk.virtualWrite(DIS_PIN, distance);
  Blynk.virtualWrite(LA_PIN, lightAnalog);
}

BLYNK_WRITE(LSH_PIN) {
  lightThreshold = param.asInt();
}
```

```
void setup() {
  Serial.begin(115200);
  Blynk.begin(auth, ssid, pass);

  pinMode(MOTOR_PIN1, OUTPUT);
  pinMode(MOTOR_PIN2, OUTPUT);
  pinMode(TRIG, OUTPUT);
  pinMode(ECHO, INPUT);

  // Blynk timer
  blynkTimer.setInterval(100L, blynkTimerEvent);

  // Motor poweroff
  motorCtrl(halt);
  // Feedback timer config
  feedbackTimer = timerBegin(0, 80, true); // Timer parameters
  timerAttachInterrupt(feedbackTimer, &onfeedbackTimer, true);
  timerAlarmWrite(feedbackTimer, feedbackTimerInterval, true);
  timerAlarmEnable(feedbackTimer);
}

void loop() {
  Blynk.run();
  blynkTimer.run();

  dataOutput();

  switch (currentState) {

    case autoCtrl:
      if (currentMode == setManual) {
        Serial.println("Manual");
        currentState = manualCtrl;
      } else if (alarmClock == true) {
        // Set time arrived
        Serial.println("Opening");
        motorCtrl(reverse); // Open curtain
        currentState = curtainOpening;
      } else if (lightAnalog <= lightThreshold) {
        if (millis() - lasttime >= debounce) {
          // Too much light
          Serial.println("Closing");
          motorCtrl(forward); // Close curtain
          currentState = curtainClosing;
        } lasttime = millis();
      } break;

    case curtainClosing:
      currentMode = setAuto;
      if (distance <= disThreshold) {
        if (millis() - lasttime >= debounce) {
          // distance below threshold
```

```
        Serial.println("Auto");
        motorCtrl(halt);
        currentState = autoCtrl;
      } lasttime = millis();
    } break;

  case curtainOpening:
    currentMode = setAuto;
    if (distance >= (totalDis - disThreshold)) {
      if (millis() - lasttime >= debounce) {
        // distance exceed threshold
        Serial.println("Auto");
        motorCtrl(halt);
        currentMode = setManual;
        desDis = distance;
        Blynk.virtualWrite(DESDIS_PIN, desDis );
        currentState = manualCtrl;
      } lasttime = millis();
    } break;

  case manualCtrl:
    if (currentMode == setAuto) {
      // set as auto control
      Serial.println("Auto");
      motorCtrl(halt);
      currentState = autoCtrl;
    } else if (alarmClock == false) {
      // Forgot to change
      Serial.println("Auto");
      motorCtrl(halt);
      currentMode = setAuto;
      currentState = autoCtrl;
    } else if ((desDis - distance) >= disThreshold) {
      if (millis() - lasttime >= debounce) {
        // adjust to desired distance
        Serial.println("Adjusting reverse");
        motorCtrl(reverse);
        currentState = curtainAdjusting;
      } lasttime = millis();
    } else if ((desDis - distance) <= -disThreshold) {
      if (millis() - lasttime >= debounce) {
        // adjust to desired distance
        Serial.println("Adjusting forward");
        motorCtrl(forward);
        currentState = curtainAdjusting;
      } lasttime = millis();
    } break;

  case curtainAdjusting:
    currentMode = setManual;
    if ((desDis - distance) <= disThreshold && (desDis - distance) >= -
disThreshold) {
```

```
        if (millis() - lasttime >= debounce) {
          // set as manual
          Serial.println("Manual");
          motorCtrl(halt);
          currentState = manualCtrl;
        } lasttime = millis();
      } else if (currentMode == setAuto) {
        // set as auto control
        Serial.println("Auto");
        motorCtrl(halt);
        currentState = autoCtrl;
      } break;
  }
}

void dataOutput() {
  if (isrCounter_dataOutput > 0) {
    portENTER_CRITICAL_ISR(&feedbackTimerMux);
    isrCounter_dataOutput--;
    portEXIT_CRITICAL_ISR(&feedbackTimerMux);

    digitalWrite(TRIG, LOW);
    delayMicroseconds(5);
    digitalWrite(TRIG, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG, LOW);
    distance = pulseIn(ECHO, HIGH) / 58;
    if (distance > totalDis) distance = totalDis;
    lightAnalog = analogRead(PHOTO_PIN);
  }
}

void motorCtrl(enum MOTOR_CTRL ctrl) {
  if (ctrl == forward) {
    digitalWrite(MOTOR_PIN1, HIGH);
    digitalWrite(MOTOR_PIN2, LOW);
  } else if (ctrl == reverse) {
    digitalWrite(MOTOR_PIN1, LOW);
    digitalWrite(MOTOR_PIN2, HIGH);
  } else if (ctrl == halt) {
    if (isFastDecay) {
      digitalWrite(MOTOR_PIN1, LOW);
      digitalWrite(MOTOR_PIN2, LOW);
    } else {
      digitalWrite(MOTOR_PIN1, HIGH);
      digitalWrite(MOTOR_PIN2, HIGH);
    }
  }
}
```