Humidifier Assistant Fan

By Ognyan Stefanov

PRODUCT DESCRIPTION

During the colder seasons of the year, the air in my room can get very dry (especially if I turn on the heater). To accommodate for the dry air, I recently purchased an air humidifier. When I leave the humidifier on during the night, I no longer suffer from dry throat in the mornings. However, one of the issues with the humidifier is that it causes water droplets to collect on the surface of my wooden desk. Even when I place the humidifier on the edge of the desk, the ejected vapor still falls straight down on my desk and condensates. In the attempt to come up with an application for my micro-kit components, I had the brilliant idea of placing a small fan behind the vapor exhaust of the humidifier. The intended purpose of the "Humidifier Assistant Fan" is to blow the vapor exhaust away from my desk so that the vapor no longer condensates on my desk. The limited supplies, tools, and financial resources compromise the functionality of the "Humidifier Assistant Fan", but the product still serves as an effective proof of concept. Future designs will build off this prototype without having to suffer from stringent limitations.

Figure 1: humidifier alone (left); humidifier with "Humidifier Assistant Fan" (right)

ELECTROMECHANICAL DETAILS

Fan: The fan was constructed using a water bottle cap, flashcards, and hot glue. The fan blades are made from layered flashcards and angled in a way such that clockwise (CW) rotation causes air to blow forward. The fan is connected to the shaft of the brushed DC motor (provided in our lab kit) using braid jewelry. The braid jewelry was pressed and plastically deformed to offer substantial friction between the shaft and fan drive. A friction fit was suitable for this application because very small loads (torques) are imposed on the fan.

Housing: There are three housing distinct components of the "Humidifier Assistant Fan". The first component is the small, rectifying support (Figure 2.b) which clips on to the humidifier and holds the device in the place. The small support also elevates the position of the fan so that there is enough clearance between the humidifier and the fan blades. This component is made from layered flash cards and hot glue. The second component is the tall backside support (Figure 3), which is responsible for elevating the device to the same height as the vapor exhaust and further preventing the device from moving. The backside support is attached to the back of the bread board via hot glue and built using popsicle sticks, brass pins, and hot glue. The last housing component is the thin wooden board (Figure 2.a). The wooden board supports all the circuitry and the brushed DC motor; the motor is attached to the

board via the motor mounting bracket. In addition, both the small and tall support are hot glued to the wooden board.

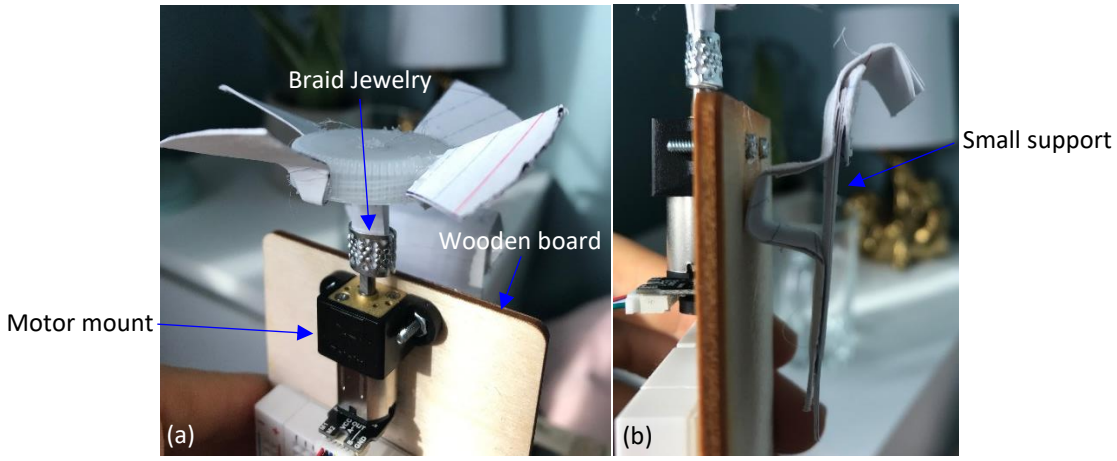

Figure 2: (a) The image on the left is a close-up of the fan; note the braid jewlery. (b) The image on the right shows the small support hot glued to the wooden board; the curved end clips on to the humidifier. The motor mount is visible in both images (black).

Fan Control: The fan is turned ON by pressing the button on the circuit board (Figure 3). Once the fan is ON, the rotational speed is adjusted via the potentiometer; turning the potentiometer CW increases the fan speed.

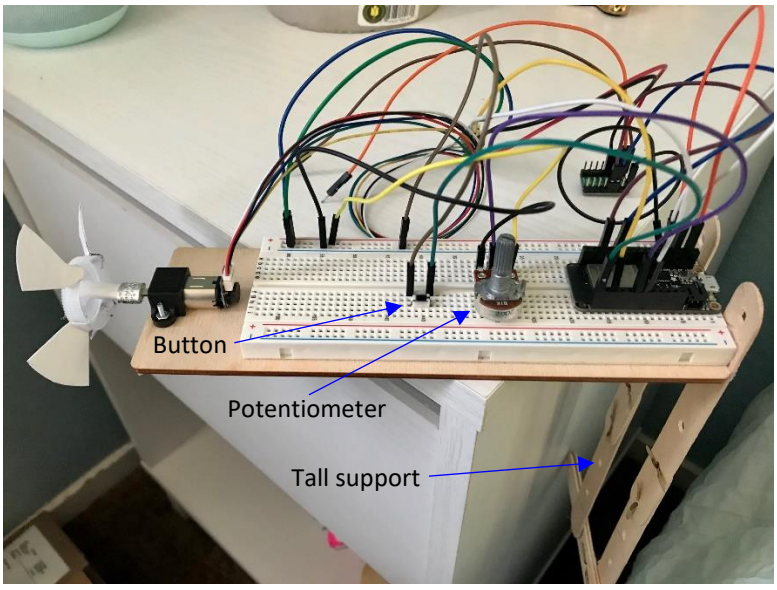

Figure 3: Image of the electronic components of the "Humidifier Assistant Fan". The button and potentiometer are used to control the behavior of the fan. The tall support is seen hanging on the right.

CIRCUIT DIAGRAM

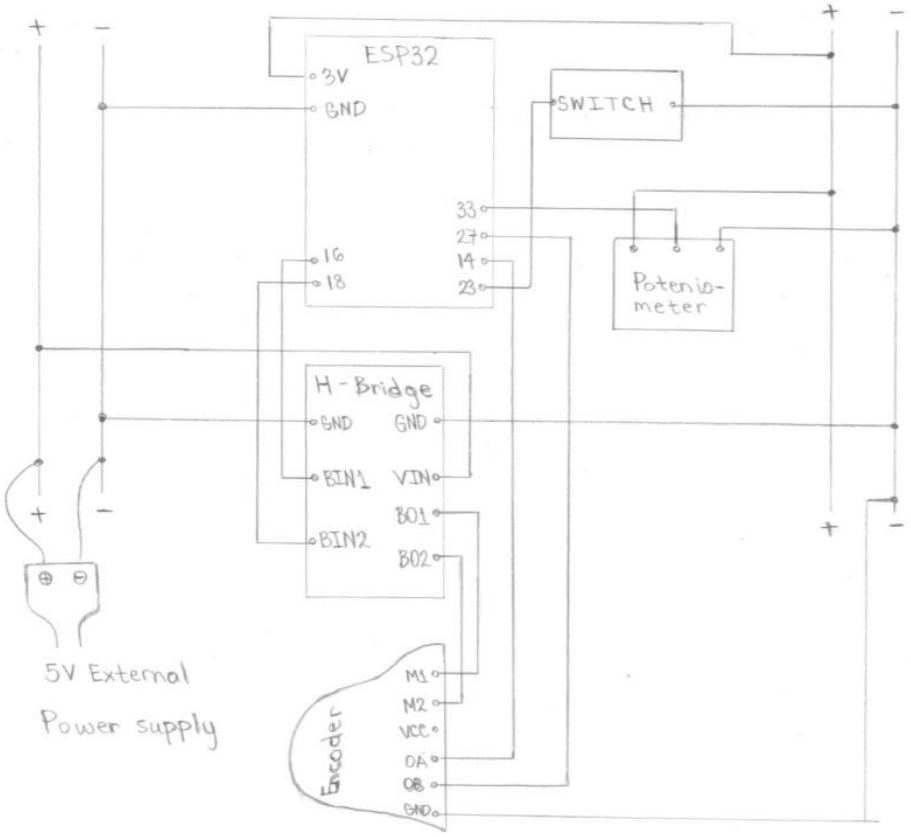

Figure 4: Diagram of the circuitry used to operate the "Humidifier Assistant Fan".

FINITE STATE MACHINE

The machine behavior of the "Humidifier Assistant Fan" is straightforward. The two operating machine states are the fan not spinning and the fan spinning. Switching between the two states is accomplished by using a physical interrupt trigger. When the button is pressed, an interrupt is triggered such that the Arduino code switches to a new case corresponding to a new state. For specificity, the complete Arduino IDE code is available in Appendix I. While in the "Fan ON" state, the speed of the fan can be varied by changing the duty cycle seen by the motor driver. Changing the duty cycle is accomplished by turning the potentiometer, which in turn causes the ESP32 to interpret different values from the corresponding ADC pin; potentiometer values are mapped from 0-4095 to 0-255. To ensure the fan is always spinning while in the "ON" state, the duty cycle has a set lower limit. Open loop control is most suitable for this application because the exact fan speed is not important. The user of the "Humidifier Assistant Fan" is only concerned with blowing the vapor exhaust away from their desk; they can visually determine the appropriate fan speed by inspecting the extent to which the vapor is displaced. A visual demonstration of the finite state machine behavior is available in the video link attached below and in Figure 6.

Video Demonstration: [link provided by instructor]

* You will notice in the video that the "Humidifier Assistant Fan" fails to effectively displace the vapor exhaust. The gearbox attached to the DC motor serves as a speed reducer, which in turn reduces the rotational fan speed. Refer to "Limitations" for further discussion.

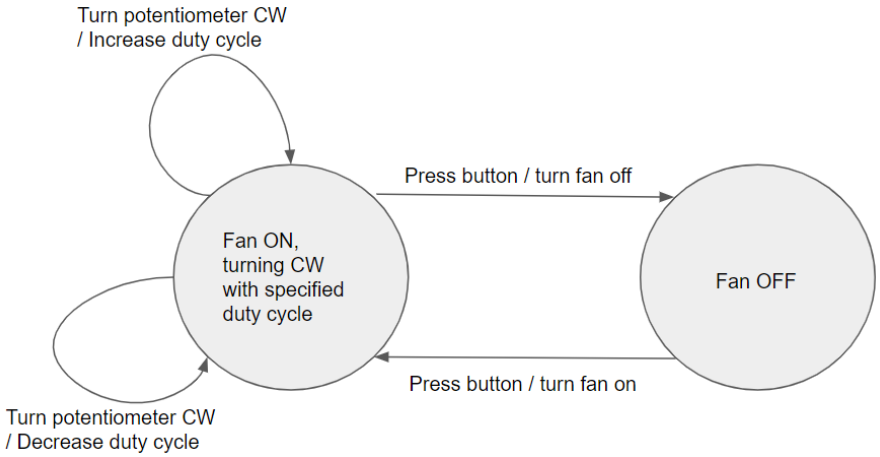

Figure 5: Finite state diagram of the intended machine behavior

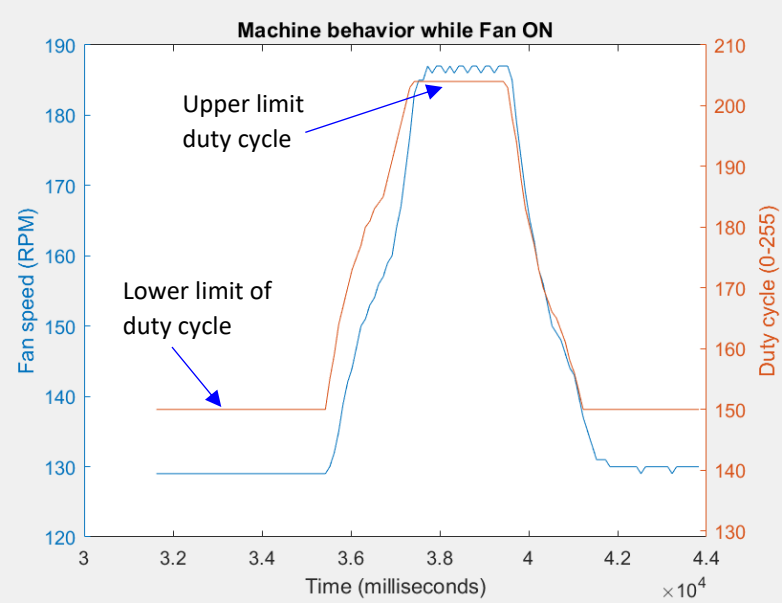

Figure 6: Display of machine behavior while system is ON

LIMITATIONS

One of the biggest limitations that interfered with the functionality of the "Humidifier Assistant Fan" is the gearbox attached the brushed DC motor. I wanted to remove the gearbox from the DC motor so that the motor shaft could spin faster (and consequently make the fan blow harder) but I did not have a screwdriver small enough to remove the screws from the gearbox. In addition, mounting the motor to the wooden board would have been more difficult without the gearbox. The provided mount was designed to interface with the pre-existing gearbox. The other major limitation that hindered the functionality of the "Humidifier Assistant Fan" was the fan design. Designing a fan out of hand cut materials for optimal air displacement is challenging to say the least. Had I been allocated more time and resources, I would have invested in a manufactured polymer based fan. The cooling fans for desktop computers would have been a suitable choice.

APPENDIX I: ARDUINO CODE

* Disclaimer: the Arduino code below contains RPM measurements for testing purposes!

```
#include <ESP32Encoder.h>


ESP32Encoder encoder;


// encoder pins
const int encoderA = 14;
const int encoderB = 27;


// PWM specs
const int freq = 24000;
const int ledChannel = 0;
const int ledChannel2 = 1;
const int resolution = 8;
const int ledPin = 16; // motor input 1
const int ledPin2 = 18; // motor input 2
const int potPin = 33;
int potVal;
int dutyVal;


// button pin (ISR) variables
const int switchPin = 23;
volatile byte butState = false;
int state = 0;


// RPM variables
int lastTime = 0;
int lastCount = 0;
int currentTime = 0;
```

```
int currentCount = 0;
int rpm = 0;

// timer ISR variables
volatile int interruptCounter;
hw_timer_t * timer = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

// timer ISR for rpm calculation
void IRAM_ATTR onTimer() {
  portENTER_CRITICAL_ISR(&timerMux);
  interruptCounter++;
  portEXIT_CRITICAL_ISR(&timerMux);
}

void setup() {
  Serial.begin(115200);
  ESP32Encoder::useInternalWeakPullResistors=UP;
  pinMode(switchPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(switchPin), togg, FALLING);

  timer = timerBegin(0, 80, true);
  timerAttachInterrupt(timer, &onTimer, true);
  timerAlarmWrite(timer, 100000, true);
  //timerAlarmEnable(timer);

  encoder.attachFullQuad(encoderA, encoderB);
  encoder.setCount(0);
  encoder.clearCount();
```

```
  ledcSetup(ledChannel, freq, resolution); // configure LED PWM functionalitites
  ledcSetup(ledChannel2, freq, resolution); // configure LED PWM functionalitites
  ledcAttachPin(ledPin, ledChannel); // attach channel to the GPIO (pin 16) to be controlled
  ledcAttachPin(ledPin2, ledChannel2); // attach channel to the GPIO (pin 18) to be controlled
}

void loop() {

  switch (state) {
    case 0: // motor off
      Service0();
      if (butState == true){
        state = 1;
        butState = false;
      }
      break;

    case 1: // motor on
      potVal = analogRead(potPin);
      dutyVal = map(potVal, 0, 4095, 0, 255);

      // Limiting the duty cycle to 80% of its max value to
      // ensure the motor runs below its max continous operating current.
      if (dutyVal > 204) {
        dutyVal = 204;
      }

      // Limiting the lower limit of the duty cyle to ensure the motor
      // always spins when the fan is "ON".
      if (dutyVal < 150) {
```

```
      dutyVal = 150;
    }

    Service1();

    if (interruptCounter > 0) {

    portENTER_CRITICAL(&timerMux);
    interruptCounter--;
    portEXIT_CRITICAL(&timerMux);

    currentTime = millis();
    currentCount = (int32_t)encoder.getCount();
    rpm = (currentCount - lastCount)*65.9/(currentTime - lastTime);
    Serial.print(rpm);
    Serial.print(" ");
    Serial.println(dutyVal);

    lastTime = currentTime;
    lastCount = currentCount;
   }
    if (butState == true){
      state = 0;
      butState = false;
    }
    break;
  }

}
```

```
void Service0() {
  timerAlarmDisable(timer);
  ledcWrite(ledChannel,0);
  ledcWrite(ledChannel2,0);
}

void Service1() {
  timerAlarmEnable(timer);
  ledcWrite(ledChannel2,0);
  ledcWrite(ledChannel, dutyVal);
}

void togg(){
  static unsigned long last_interrupt_time = 0;
  unsigned long interrupt_time = millis();

  // If interrupts come faster than 150ms, assume it's a bounce and ignore
  if (interrupt_time - last_interrupt_time > 150){
     butState = true;
  }
  last_interrupt_time = interrupt_time;
}
```