Elevator for Air Quality Sensor

By Yilan Lu

Description of the product:

I am currently working on a project that needs to detect air quality on campus with an air quality sensor assembled on a cart. I found that if I attach the sensor directly to the cart, the data I collected would be significantly greater than that I collected when I held the sensor with my hands. I needed to hold the sensor higher to get more accurate data, but making the cart too high will make its movement unsafe. Therefore, I thought I needed an elevator for this sensor. The elevator should stay low when the cart is moving and lift the sensor when I need to detect the air quality. I chose a folding telescope to work as the elevator. I think it will be a good opportunity for me to practice driving a folding telescope structure with a motor. And since I am introducing an Arduino board, I can also learn how to collect data with the Arduino board. I have access to a Raspberry Pi so I used the Raspberry Pi instead of the Esp-32 Feather.

Figure 1: The cart with an air quality sensor on it(left). The elevator is attached to the cart, lifting the sensor up and down.

Electromechanical details:

The folding telescope structure: The folding telescope is made up of parallel four bar linkages structure stretches with a push at the end. The push operation is conducted by a power screw connected to a stepper motor. When the motor rotates in counterclockwise direction the platform goes upward, and vice versa. I dug a hole on the acrylic board which is part of the cart and fixed the whole folding telescope structure to the cart.

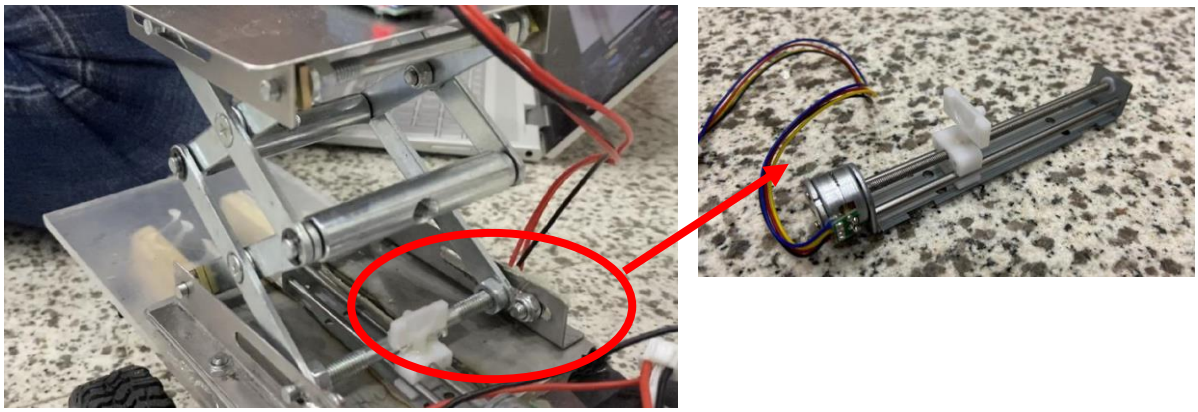

Figure 2: The folding telescope is attached to the cart with a bolt. The fixed part and move part of the power screw are connected to the two bars at the end of the telescope respectively. The other end of the power screw is the motor. The sensor is placed on the top of the platform.

The whole structure: The folding telescope, the Raspberry Pi, the L298N encoder, and the battery are all placed on the acrylic board of the cart.

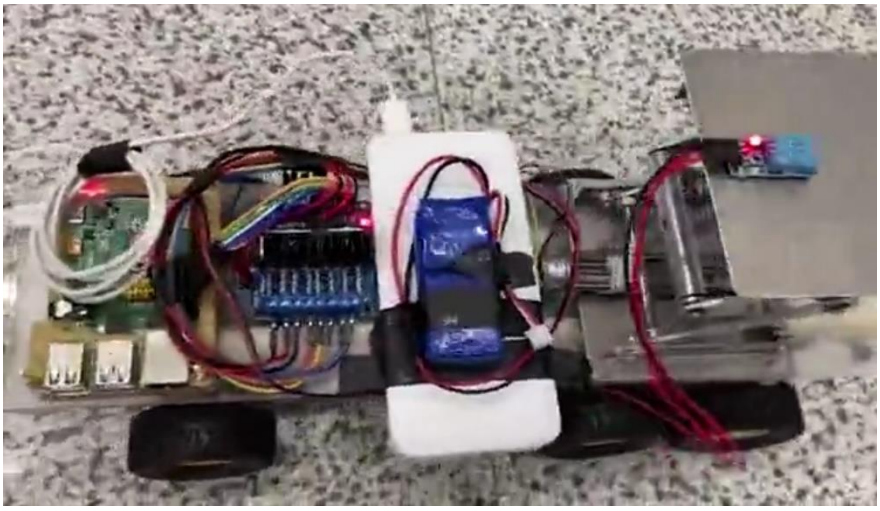

Figure 3: From left to right, there is the Raspberry Pi, the L298N encoder, the battery and the folding telescope. The Raspberry Pi controls the motor and receives signal from the sensor. The L298N motor driver receives signal from the Raspberry Pi and drives the stepper motor. The white batter charges the Raspberry Pi. The blue battery(7.4V) charges the driver.

Circuit:

The circuit serves two purposes: (1) Drive the motor, (2) Collect data from the air quality sensor.

(1) Motor: The stepper motor was bought in pair with the power screw. It has a resolution of 9 degrees. It needs four input signals to control it so the DRV8833 in the labkit seems not good enough. I bought the L298N encoder (5.9 CNY, about $1) recommended by the online shop assistant. The clockwise and counterclockwise movement of the motor is remotely controlled with VNC serve that connects Raspberry Pi and my computer with the Internet.

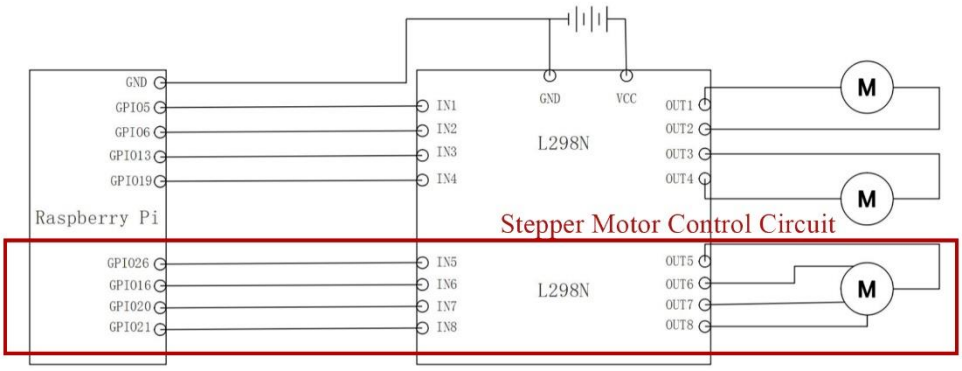

Figure 4: The circuit diagram of the encoder. The upper part is the control circuit of the other parts, the lower part is the control circuit of the stepper motor. The A+, A-, B+, B- of the stepper motor are connected to OUT 5-8 of the encoder respectively. The status table of the motor is in the appendix.

(2) Sensor: A DHT11 temperature and humidity sensor was provided by the program mentioned in the product description. It uses single-bus format to communicate between Raspberry Pi and DHT11 sensor. One communication process is about 4ms.

Figure 5: The circuit diagram of the sensor.

Finite State Machine

Both the motor and the sensor has ON and OFF state. The stepper motor is on either I press “X” or “Z” on my keyboard. “Z” makes the folding telescope stretch and “X” makes it fold. The sensor will return data of temperature and humidity when the start signal is sent.

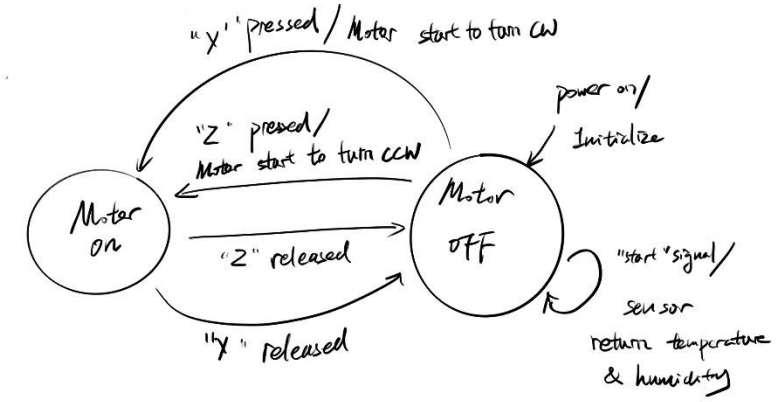

Figure 6: The finite state diagram.

I didn’t measure the change in height versus time because it means that I need to introduce a new sensor just to finish the report. But I did measure the rising and falling time of the “elevator” to show how this machine performed. The process takes less than 2 seconds in average, which is acceptable for adjustment.

Table 1: Measurement of the Rising and Falling Time

Group	Rise time (s)	Fall time (s)
1	2‘05	1‘24
2	1‘48	1‘27
3	1‘53	1‘43
Average	1‘54	1‘31

The Status Table of the Stepper Motor is in Appendix I. The complete Arduino Code is in Appendix II. The performance of the sensor is in Appendix III.

Appendix I: The Status Table of the Stepper Motor

A+	A-	B+	B-	Forward/ Up	Backward/ Down
1	0	0	0		
1	0	1	0		
0	0	1	0		
0	1	1	0		
0	1	0	0		
0	1	0	1		
0	0	0	1		
1	0	0	1		

Appendix II: Arduino Code

The code includes other part that controls the motor on the cart. They are not coded for the ME102B final project, but I can't separate them apart. The codes my final project used are highlighted with red squares.

```
#Motor.py
import RPi.GPIO as GPIO
import time
import sys
import tty
import termios

GPIO.setwarnings(False)
IN1 = 5
IN2 = 6
IN3 = 13
IN4 = 19
IN5 = 26
IN6 = 16
IN7 = 20
IN8 = 21

def readchar():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch

def readkey(getchar_fn=None):
    getchar = getchar_fn or readchar
    c1 = getchar()
    if ord(c1) != 0x1b:
        return c1
    c2 = getchar()
    if ord(c2) != 0x5b:
        return c1
    c3 = getchar()
    return chr(0x10 + ord(c3) - 65)

def init():
```

```
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(IN1,GPIO.OUT)
    GPIO.setup(IN2,GPIO.OUT)
    GPIO.setup(IN3,GPIO.OUT)
    GPIO.setup(IN4,GPIO.OUT)
    GPIO.setup(IN5,GPIO.OUT)
    GPIO.setup(IN6,GPIO.OUT)
    GPIO.setup(IN7,GPIO.OUT)
    GPIO.setup(IN8,GPIO.OUT)


def forward():
        if a==0 and d==0:
            GPIO.output(IN5,GPIO.LOW)
            GPIO.output(IN6,GPIO.HIGH)
            GPIO.output(IN7,GPIO.LOW)
            GPIO.output(IN8,GPIO.LOW)
            GPIO.cleanup()
        else:
            GPIO.output(IN5,GPIO.LOW)
            GPIO.output(IN6,GPIO.HIGH)
            GPIO.cleanup()

def back():
        if a==0 and d==0:
            GPIO.output(IN5,GPIO.HIGH)
            GPIO.output(IN6,GPIO.LOW)
            GPIO.output(IN7,GPIO.LOW)
            GPIO.output(IN8,GPIO.LOW)
            GPIO.cleanup()
        else:
            GPIO.output(IN5,GPIO.HIGH)
            GPIO.output(IN6,GPIO.LOW)
            GPIO.cleanup()

def right():
        GPIO.output(IN7,GPIO.LOW)
        GPIO.output(IN8,GPIO.HIGH)
        GPIO.cleanup()

def left():
        GPIO.output(IN7,GPIO.HIGH)
        GPIO.output(IN8,GPIO.LOW)
        GPIO.cleanup()
```

```
def stop():
        GPIO.output(IN5,GPIO.LOW)
        GPIO.output(IN6,GPIO.LOW)
        GPIO.output(IN7,GPIO.LOW)
        GPIO.output(IN8,GPIO.LOW)
        GPIO.cleanup()

def down():
    n=0
    while n<150:
     GPIO.output(IN1,GPIO.HIGH)
     GPIO.output(IN2,GPIO.LOW)
     GPIO.output(IN3,GPIO.HIGH)
     GPIO.output(IN4,GPIO.LOW)
     time.sleep(t)

     GPIO.output(IN1,GPIO.LOW)
     GPIO.output(IN2,GPIO.HIGH)
     GPIO.output(IN3,GPIO.HIGH)
     GPIO.output(IN4,GPIO.LOW)
     time.sleep(t)

     GPIO.output(IN1,GPIO.LOW)
     GPIO.output(IN2,GPIO.HIGH)
     GPIO.output(IN3,GPIO.LOW)
     GPIO.output(IN4,GPIO.HIGH)
     time.sleep(t)

     GPIO.output(IN1,GPIO.HIGH)
     GPIO.output(IN2,GPIO.LOW)
     GPIO.output(IN3,GPIO.LOW)
     GPIO.output(IN4,GPIO.HIGH)
     time.sleep(t)

     n=n+1
    GPIO.cleanup()

def up():
    n=0
    while n<150:
     GPIO.output(IN1,GPIO.HIGH)
     GPIO.output(IN2,GPIO.LOW)
     GPIO.output(IN3,GPIO.LOW)
```

```
    GPIO.output(IN4,GPIO.HIGH)
    time.sleep(t)

    GPIO.output(IN1,GPIO.LOW)
    GPIO.output(IN2,GPIO.HIGH)
    GPIO.output(IN3,GPIO.LOW)
    GPIO.output(IN4,GPIO.HIGH)
    time.sleep(t)

    GPIO.output(IN1,GPIO.LOW)
    GPIO.output(IN2,GPIO.HIGH)
    GPIO.output(IN3,GPIO.HIGH)
    GPIO.output(IN4,GPIO.LOW)
    time.sleep(t)

    GPIO.output(IN1,GPIO.HIGH)
    GPIO.output(IN2,GPIO.LOW)
    GPIO.output(IN3,GPIO.HIGH)
    GPIO.output(IN4,GPIO.LOW)
    time.sleep(t)

    n=n+1
  GPIO.cleanup()
init()

while 1:
        t=0.002
        w=0
        s=0
        a=0
        d=0
        res=readkey()
        if res=="w":
            w=1
            s=0
            a=0
            d=0
        if res=="s":
            w=0
            s=1
            a=0
            d=0
        if res=="a":
```

```
            w=0
            s=0
            a=1
            d=0
        if res=="d":
            w=0
            s=0
            a=0
            d=1
        if res=="wa":
            w=1
            s=0
            a=1
            d=0
        if res=="wd":
            w=1
            s=0
            a=0
            d=1
        if res=="sa":
            w=0
            s=1
            a=1
            d=0
        if res=="sd":
            w=0
            s=1
            a=0
            d=1
        if res=="b":
            w=0
            s=0
            a=0
            d=0
        if res=="z":
            up()
            init()
        if res=="x":
            down()
            init()
        if w==1:
                forward()
                init()
        if s==1:
```

```
            back()
            init()
    if a==1:
            left()
            init()
    if d==1:
            right()
            init()
    if w==0 and s==0 and a==0 and d==0:
            stop()
            init()
```

Appendix III: The Performance of the Sensor

The first 4 data were taken at road side. The latter 4 were taken at lake bank. We can see significant rise in humidity.