Temperature Sensor

CS3733-D24 Prof. Wong

Designed by Sofia Xie, Gabriel Olafsson and José Manuel Pérez Jiménez

Neon Nymphs

Coach: Joseph Cardarelli

Person	Position	GitHub
Sean Arackal	Back-End Dev	scriblesean
Maddux Berry	Project Manager / Front-End Dev	macethewindu66
Lorenzo Cassano	Back-End Dev	Lorenzo-Cassano
Christian Consiglio	Lead Software Engineer (Front-End)	FastJr
Peter Czepiel	Front-End Dev	peczepiel
Ethan Glasby	Product Owner / Front-End Dev	E-man-dev
Timothy Hutzley	Scrum Master / Front-End Dev	tahutzley
José Manuel Pérez Jiménez	Assistant Lead Software Engineer (Algorithms)	josemanuel657
Gustave Montana	Front-End Dev	gnonk323
Gabriel Olafsson	Assistant Lead Software Engineer (Back-End)	gabrielolafs
Sofia Xie	Document Analyst/ Front-End Dev	SofiaXie

Files for Temperature Sensor:
https://github.com/SofiaXie/Temperature-Sensor

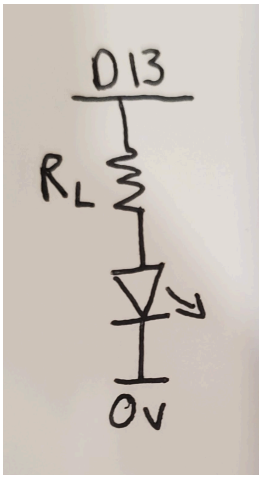
GitHub Link:
https://github.com/CS3733-2024-TeamN

AWS Link:
https://ec2-18-221-189-137.us-east-2.compute.amazonaws.com/

Table of Contents

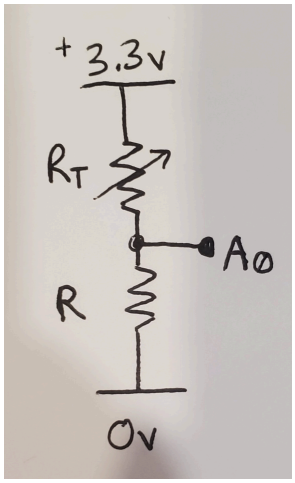
Designing the Circuit

Choosing Resistor Values



LED Circuit

Values: R_L = 1kΩ



Voltage Divider

Values: R_T = 30kΩ - 60kΩ ; R = 57kΩ

Circuit Diagram for Functionality

Ohm's law is integral to your understanding of electronics, and ensures the current is properly limited so as to not burn out any components. The LED circuit has a resistor, R_L, to limit the current flowing through the diode protecting it from burning out. The LED can handle a max current of 10mA. To ensure this current is well below this level the resistance was calculated with I = 3mA. This gave us a standard resistor value of 1kΩ.

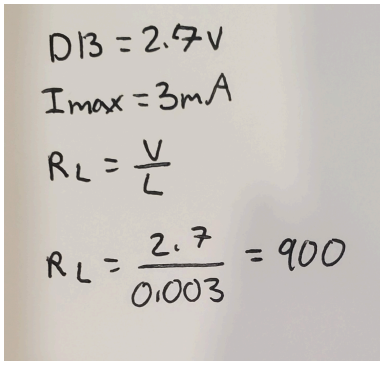
Ohm's Law:

$$V = I * R_L$$

Rearranged to find Resistance:

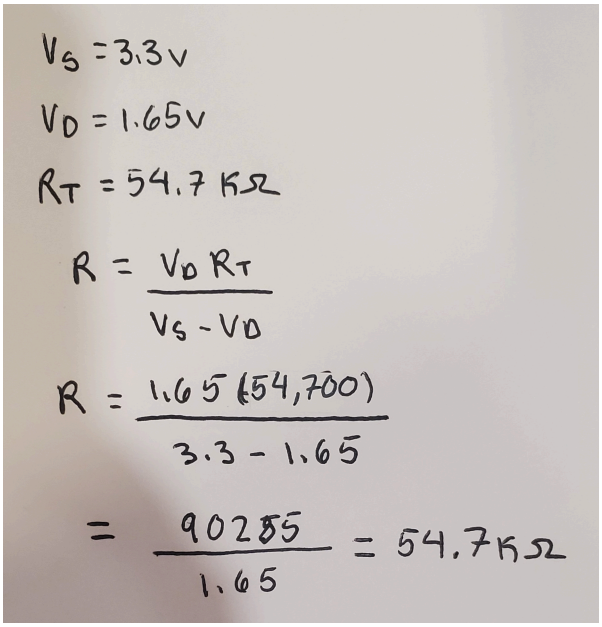
$$R_L = \frac{V}{I}$$

Value Calculations:

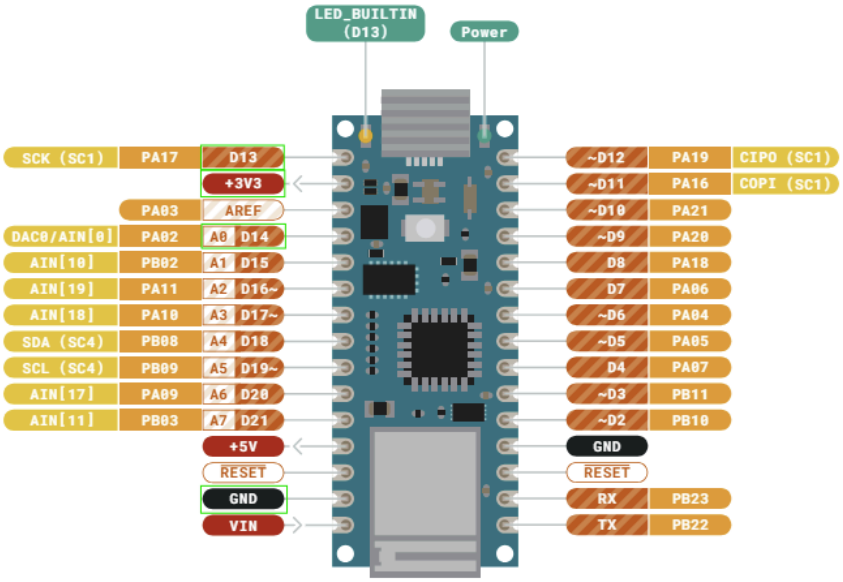


Voltage divider outputs a fraction of the input voltage. This ratio is determined by the resistor values used. R_T is a NTC thermistor which operates linearly between the ranges of 60kΩ(room temp) to 30kΩ (100F). It's real values was measured to be 54.7kΩ. The value of R

was chosen to be a standard value of 57kΩ, to equally divide the supply voltage of 3.3 V to 1.65 V.

Voltage Divider Equation:	Value Calculations:
$V_D = V_S \frac{R}{R+R_T}$ Rearranged to find Resistance: *See full algebra in appendix 1.* $R = \frac{V_S - V_D}{V_D R_T}$	

Connecting to Ardunio

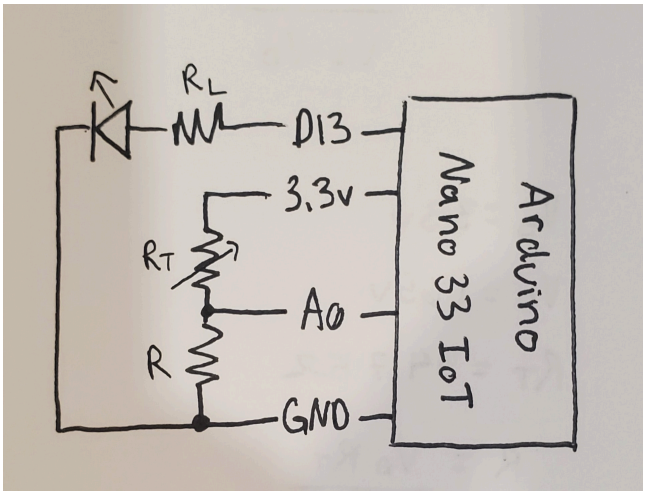


Pinout Diagram for the Arduino Nano 33 IoT

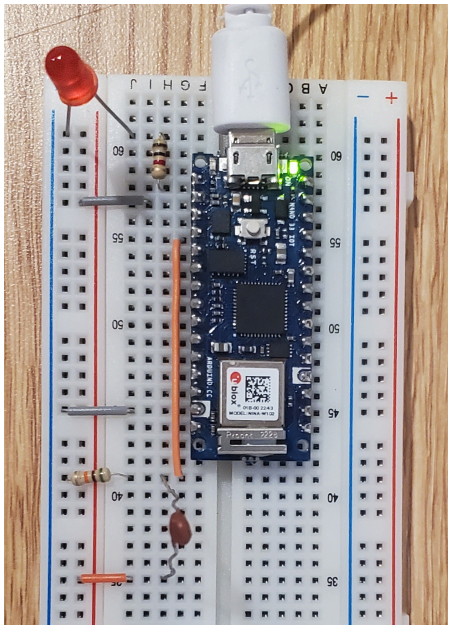
LED Circuit:
D13 provides the voltage to turn on the LED, it was measured to be 2.7V.

Voltage Divider:
Input (V_S) of 3.3V was used to power the circuit. The GND pin is connected to 0V. A0 was connected to the output of the voltage divider, V_D.

Putting it All Together



Circuit Connection



Final Breadboard Design

Reading the Data

Coding the arduino to read voltage and convert to temperature.

Why Nano33 IoT?

When first conducting research on how to display analog data to the website I was looking at using BLE (bluetooth Low Energy) Technology and chose the Arduino Nano33 IoT. After talking with team members and learning more about coding the Arduino it became apparent that sending data via bluetooth required creating a client to receive data, and was proven to be too difficult. We also looked into sending data over wifi but quickly discovered it would be challenging due to encryption.

Learning to Code Arduino

Ardunio has very thorough documentation on how to use its devices and provides great starter tutorials. I used "Controlling LED" and "Connecting to WiFi" to learn to program arduino.

Converting Voltage to Temperature

Math Time! I did not want a linear relationship so I hardcoded voltage ranges to match certain temperature values.

```
void loop() {
 unsigned long currentMillis = millis();

  if(currentMillis - previousMillis >= interval){
    previousMillis = currentMillis;
    int sensorValue = analogRead(A0); //reads 1023 values
    float voltage = sensorValue * (3.3/1023.0); //read voltage

    if (voltage < 1.5) {
      temp = (60.0/1.5)*voltage;
    }
    if(voltage < 1.6 & voltage > 1.5){
      temp = (70.0/1.6)*voltage;
    }
    if(voltage < 1.7 & voltage >1.8){
      temp = (80.0/1.8)*voltage;
    }
    if(voltage < 1.8  & voltage > 1.9);{
      temp = (90.0/1.9)*voltage;
    }
    if(voltage > 2.0 & voltage < 2.1){
      temp = (95.0/2.0)*voltage;
    }

    // If temp to high led turns on!
    if(voltage > 3.2){
      temp = (95.0/2.0)*voltage;
      //turn on led, wait 5s then turn off
      digitalWrite(LED_PIN, HIGH);
      Serial.println("ledOn");
    }

    // digitalWrite(LED_PIN, LOW);
    Serial.println(temp);
    timeDelay();
  }
}
```

Sending to Local Host

Why localhost?

First consider having it on the cloud and sending via wifi, or bluetooth. Through WPI wifi required certificates and extra encryption. Could require us to host another server to pass data to. So reading from the local easiest and cheapest for us.

```python
def post_data(temp = 101):
    curr_time = time.localtime()
    day = curr_time.tm_mday
    date_to_post = f"{time.strftime("%A", curr_time)}, {time.strftime("%B", curr_time)} {day}{"st" if day % 10 == 1 else "nd" if day % 10 == 2 else "rd" if day % 10 == 3 else "th"}"
    time_to_post = f"{curr_time.tm_hour % 12 if curr_time.tm_hour % 12 else 12}:{curr_time.tm_min:02} {"AM" if curr_time.tm_hour < 12 or curr_time.tm_hour == 24 else "PM"}"

    try:
        data = {
            'temp': float(temp),
            'time': time_to_post,
            'date': date_to_post
        }

        response = requests.post(" http://localhost:3000/api/weather", json=data)

        if response.status_code == 200:
            print('that worked!')
        else:
            print(f"status code {response.status_code}")
            print('that did not work')
            print(response.text)

    except Exception as e:
        print("error", str(e))
```

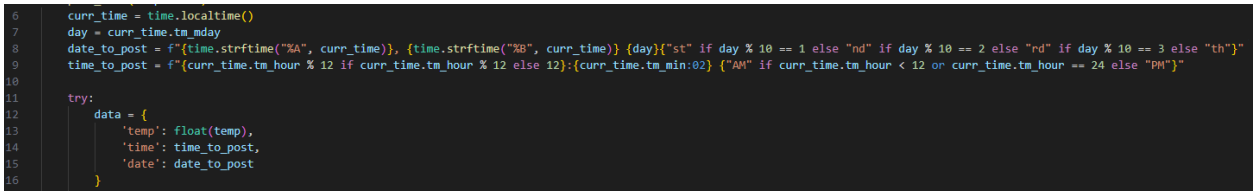
Reading from Arduino IDE and sending as json.

```python
ardunio_port = "COM5"
baud_rate = 9600

ser = serial.Serial(ardunio_port, baud_rate)
time.sleep(2); #connection settle time

def avg(alist):
    temp = 0
    for i in alist:
        temp += float(i)

    return temp / len(alist)
```

Take an avg of 10 samples.

```python
def avg(alist):
    temp = 0
    for i in alist:
        temp += float(i)

    return temp / len(alist)
```

Python code also uses a built-in library to fetch time and date.



```
    curr_time = time.localtime()
    day = curr_time.tm_mday
    date_to_post = f"{time.strftime("%A", curr_time)}, {time.strftime("%B", curr_time)} {day}{"st" if day % 10 == 1 else "nd" if day % 10 == 2 else "rd" if day % 10 == 3 else "th"}"
    time_to_post = f"{curr_time.tm_hour % 12 if curr_time.tm_hour % 12 else 12}:{curr_time.tm_min:02} {"AM" if curr_time.tm_hour < 12 or curr_time.tm_hour == 24 else "PM"}"

    try:
        data = {
            'temp': float(temp),
            'time': time_to_post,
            'date': date_to_post
        }
```

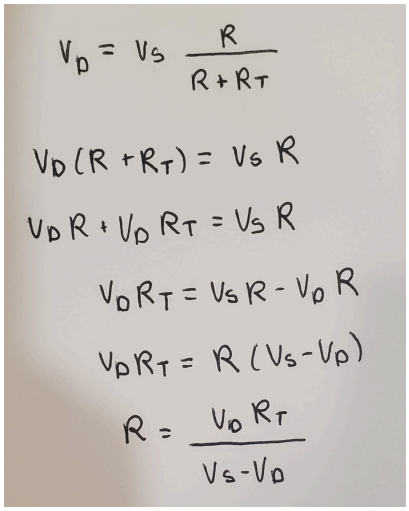
Displaying to Website

Sending from a python file, creating a route to the website

```
try:
    alist = []
    while True:
        if ser.in_waiting > 0:
            if len(alist) == 10:
                post_data(round(avg(alist), 2))
                alist = []
            else:
                line = ser.readline().decode('utf-8').rstrip()
                alist.append(line)

except KeyboardInterrupt:
    print("Exiting program")

temp = line

ser.close()
```

Appendix

Appendix 1: Algebra for Voltage Divider.



Appendix 2: Learning to Code with the Arduino