

# PI+Feedforward Temperature Controller

## User Manual

### Specifications

Supply Voltage	7.5 V
Supply Current	At least ~2A, limited to at most 3A*
Resolution	0.001C°
Accuracy	$\sqrt{(0.3 + 0.005 \times C^{\circ})^2 + (0.1 + 0.0017 \times C^{\circ})^2}$

\*Power supply should be a benchtop adjustable power supply with the current limited to 3A at most

### Hardware

Part	Model
Microcontroller	Arduino Nano
H-Bridge Controller	HiLetgo BTS7960
4-Wire RTD Module	Atlas Scientific RTD-4-OEM
4-Wire RTD	Netsushin NFR-CF Series

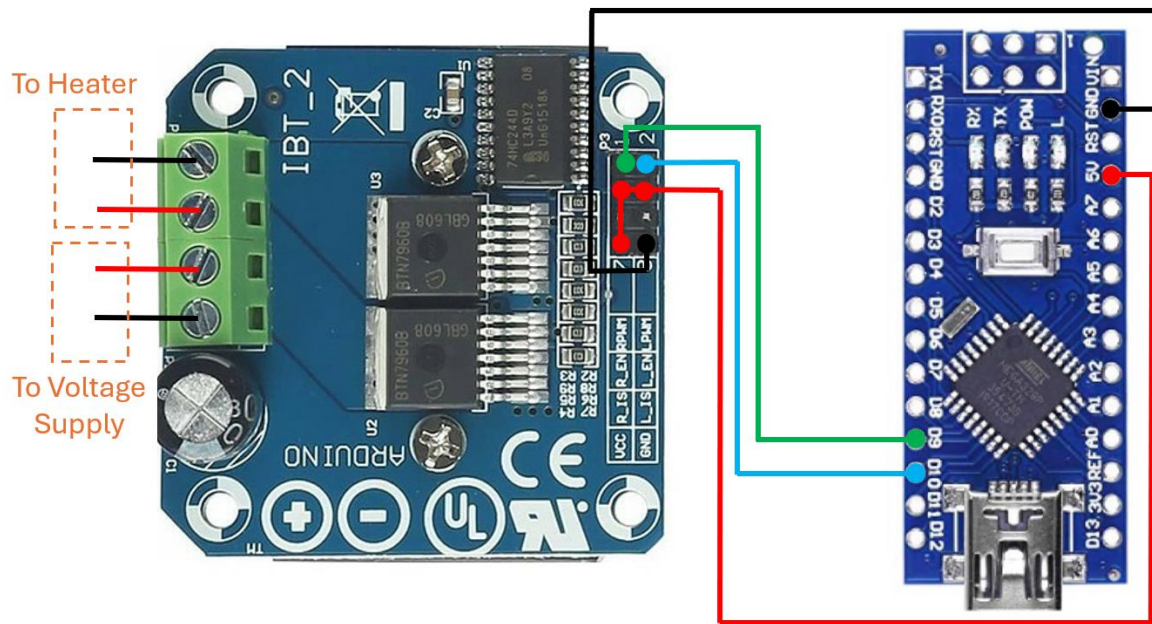
### Software

Arduino IDE (required)

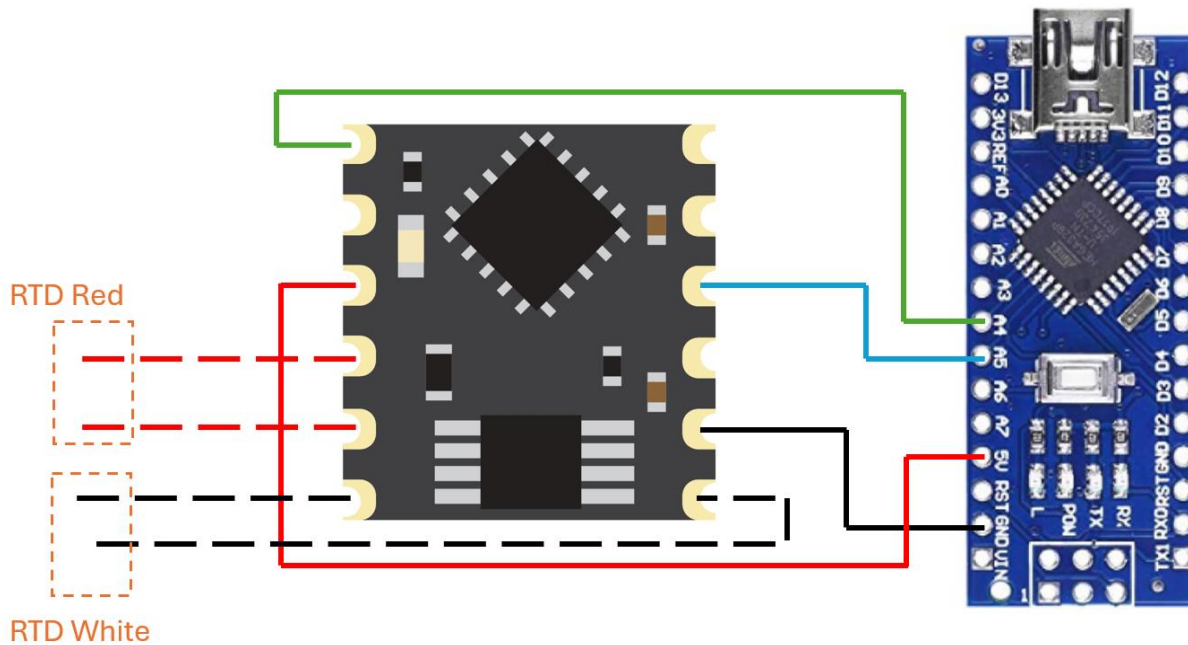
BetterSerialPlotter (recommended)

<https://github.com/nathandunk/BetterSerialPlotter>

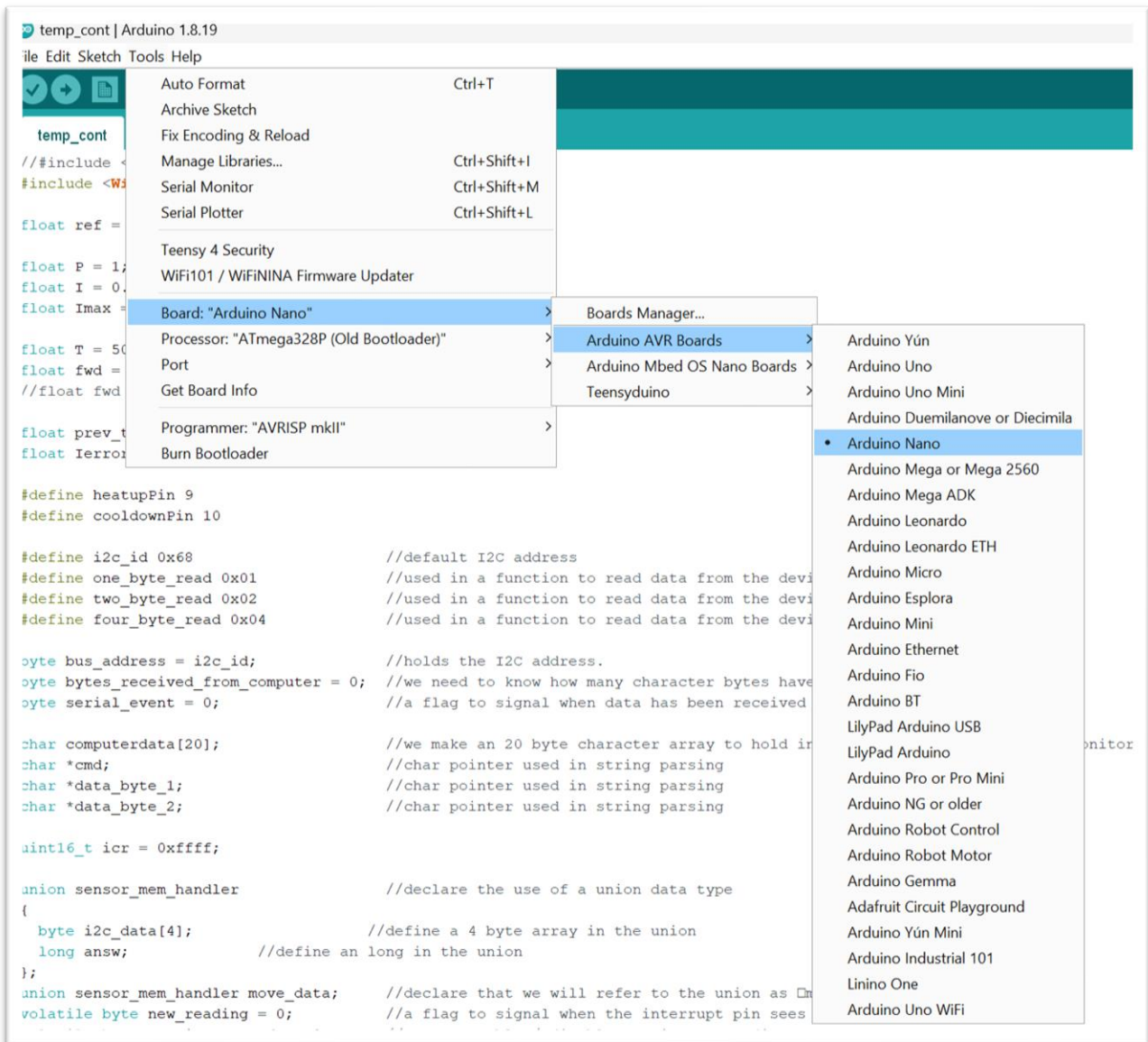
## Wiring



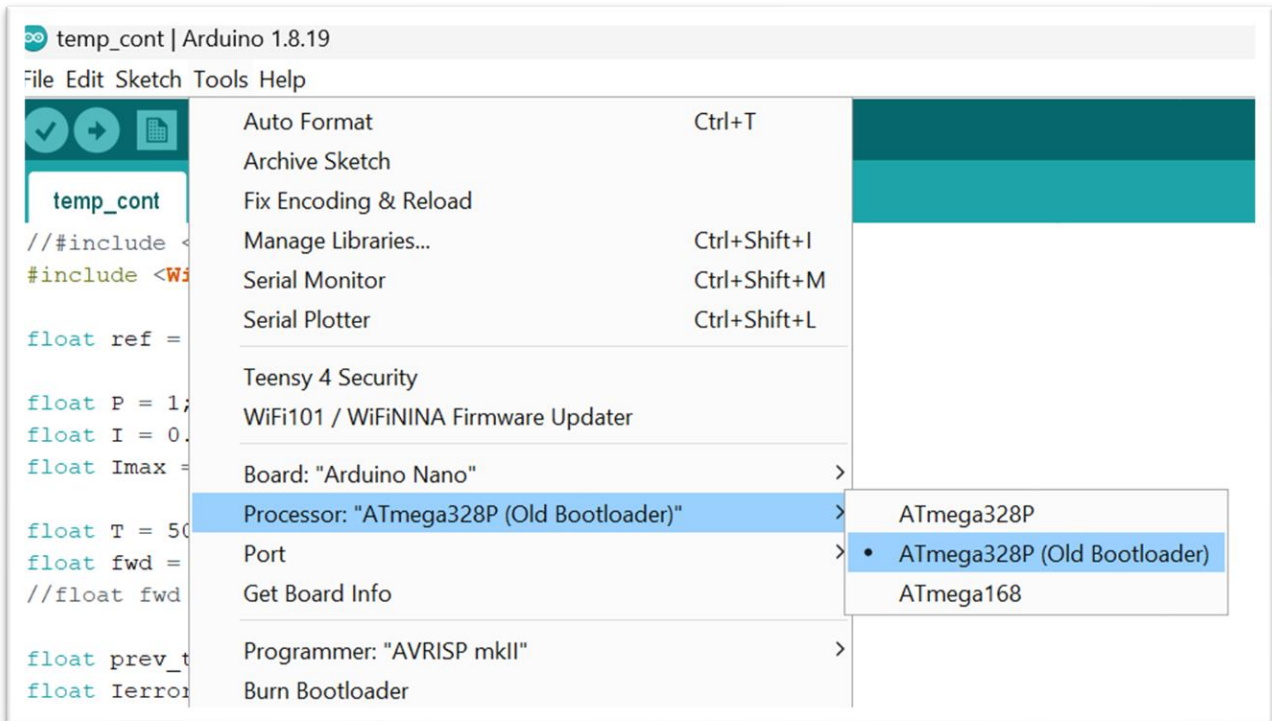
Note: The 5V supply wires are the yellow, orange, and red breadboard wires between the RTD chip and the arduino. They are all connected to 5V under the board.



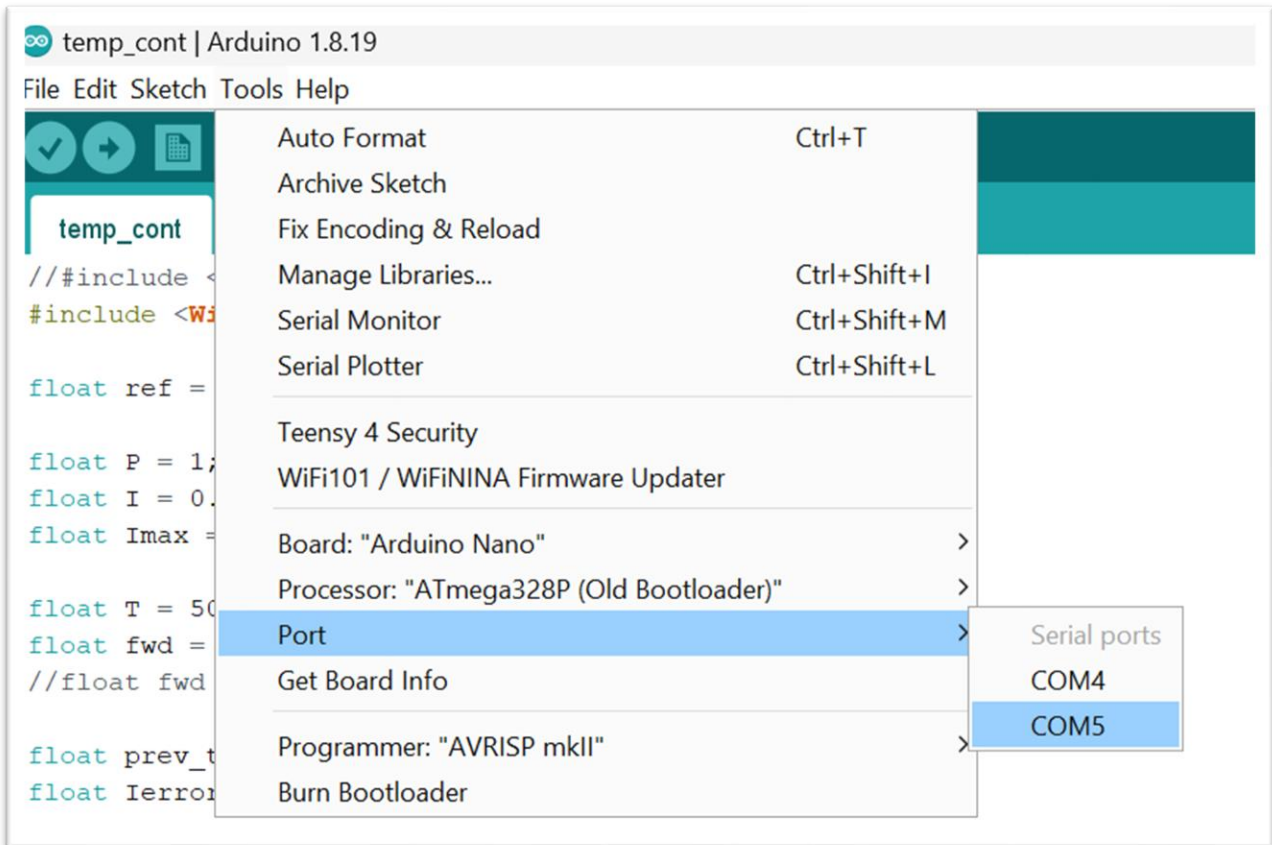
## Uploading Code



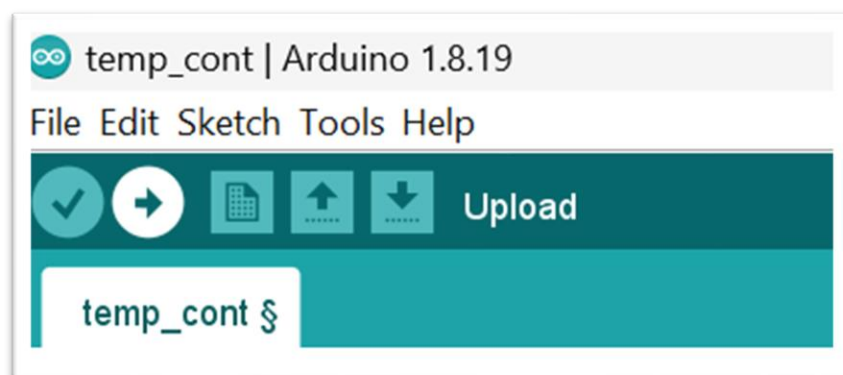
Make sure the Arduino Nano is selected in the board menu under tools.



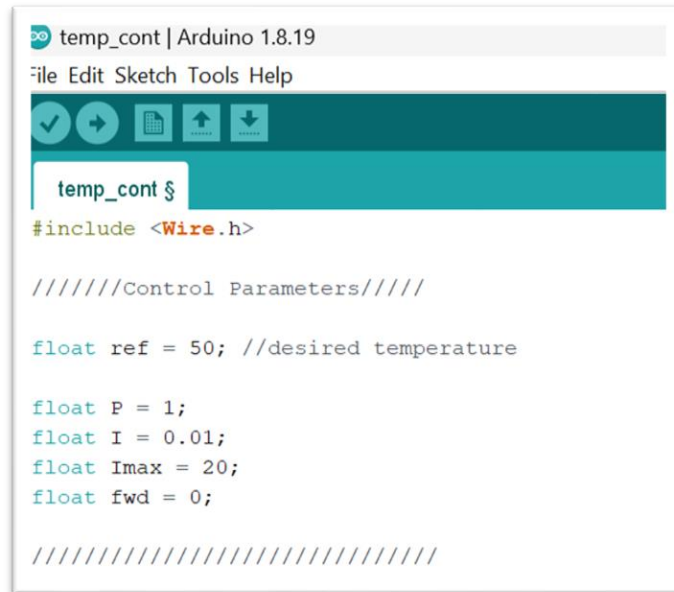
The ATmega328P (Old Bootloader) must be selected as the processor. The regular bootloader does not work with this board.



Select the COM port of the Arduino. This will vary for you depending on what you have plugged in and your computer hardware. One way to find out which COM port it is without trial and error is to look at the menu before plugging in the Arduino, and then plug it in and whichever COM port appears will be the Arduino.



Upload the code with the highlighted button with a right arrow.



```
temp_cont | Arduino 1.8.19
File Edit Sketch Tools Help

temp_cont $
#include <Wire.h>

////////Control Parameters////////

float ref = 50; //desired temperature

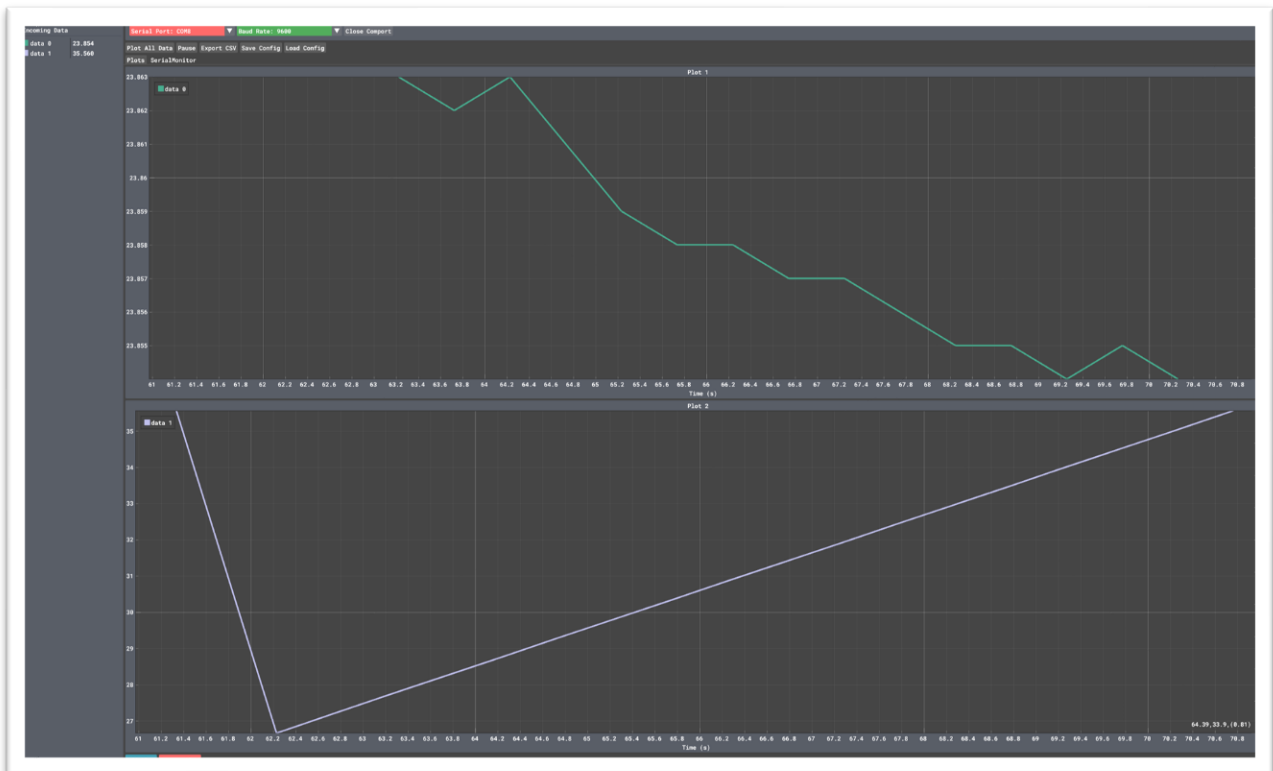
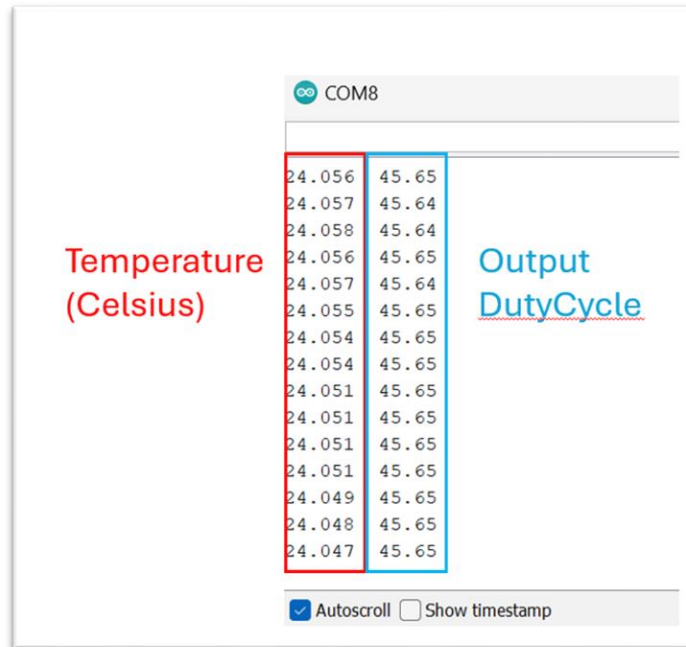
float P = 1;
float I = 0.01;
float Imax = 20;
float fwd = 0;

////////////////////////////////////
```

The only part of the code should be necessary to change is located at the top of the program. Under normal operation only the ref variable should be changed, this is what sets the desired temperature. The other variables control the tuning parameters of the control system.

## Data Acquisition

Two variables are sent through serial communication: 1) Temperature in Celsius 2) Current Duty cycle. This can be collected by using the Arduino serial monitor, however I recommend to use BetterSerialPlotter. It can graph the data in real time on separate graphs and has an export csv option.



## Control

$$Output = F + Pe(t) + I \int_0^t e(t)dt$$

$$Output = fwd + P * error + I * Ierror$$

### Default Parameters

P	1
I	0.01
I <sub>max</sub>	20
F	0

### Parameter description

P-Proportional term:

Adjusts the output directly proportionally to the error

I-Integral term:

Adjust the output proportionally to the integrated error. This term accounts for steady state error, modeling error, and changing environment. Error will integrate into Ierror unless Ierror exceeds I<sub>max</sub>.

F-Feedforward term:

Adjusts the output that is constant. This is meant to counteract the heat loss.

### Tuning Strategy 1 (Basic)

Use default parameters.

### Tuning Strategy 2 (Advanced)

The main advantage to this strategy is that the response time will be shorter and will likely also overshoot less. Set F to approximately the duty cycle needed to maintain the desired temperature. This is best set after an experiment at that temperature has already been run and the duty cycle when the temperature is steady is recorded. This only needs to be approximate as the integral should compensate for error. I<sub>max</sub> can be decreased since the integral term will no longer be solely responsible for maintaining the steady state temperature. This will reduce overshoot.



## Calibration