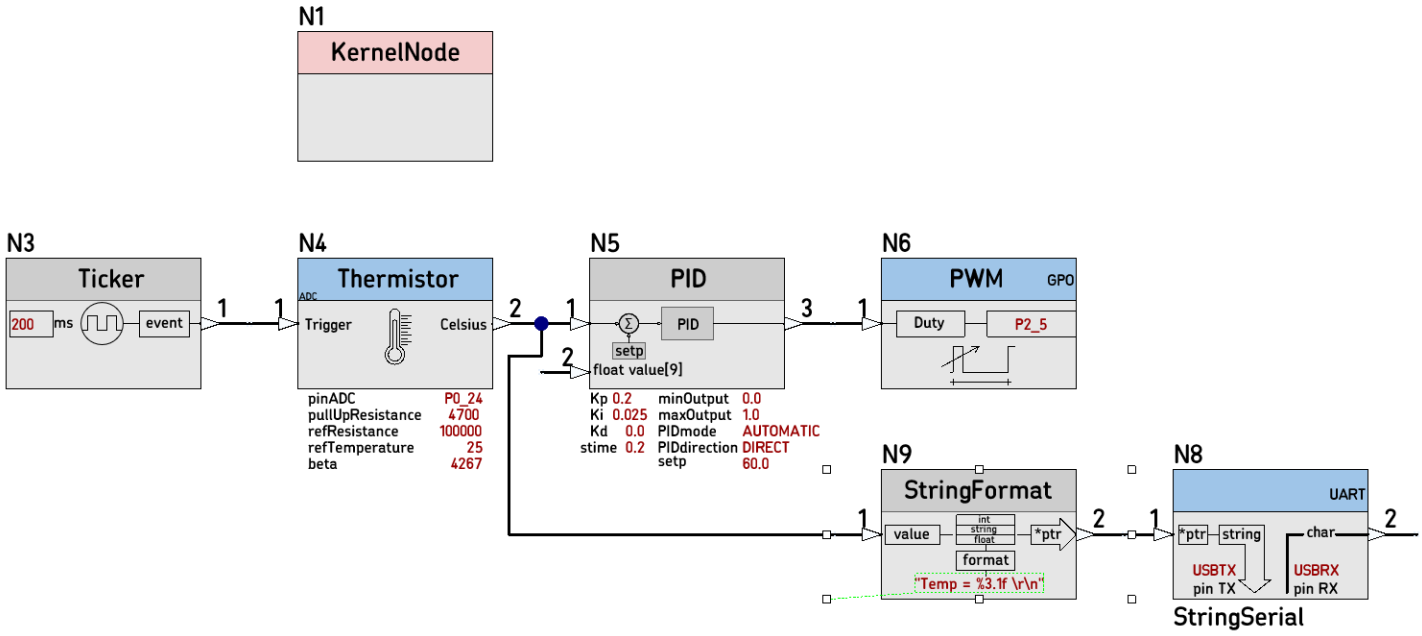
38D_PID_test

Testing program for the nBlocksStudio Node PID. The C++ source code, is automatically generated from the schematic Design 38D_PID_test_SCHEMATIC . Controlls a 40W 3D-printer heating element with PWM via power-MOSFET, using a Thermistor for temperature sensing.

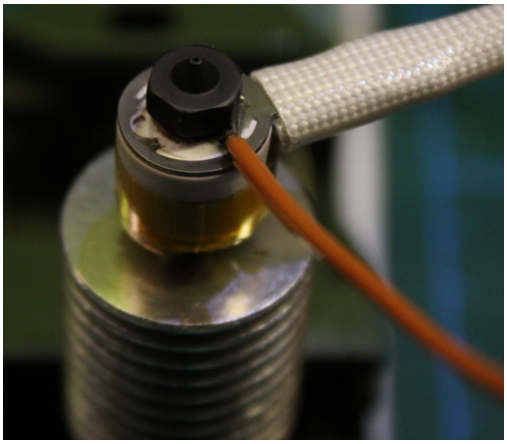
nBlocksStudio Schematic Design



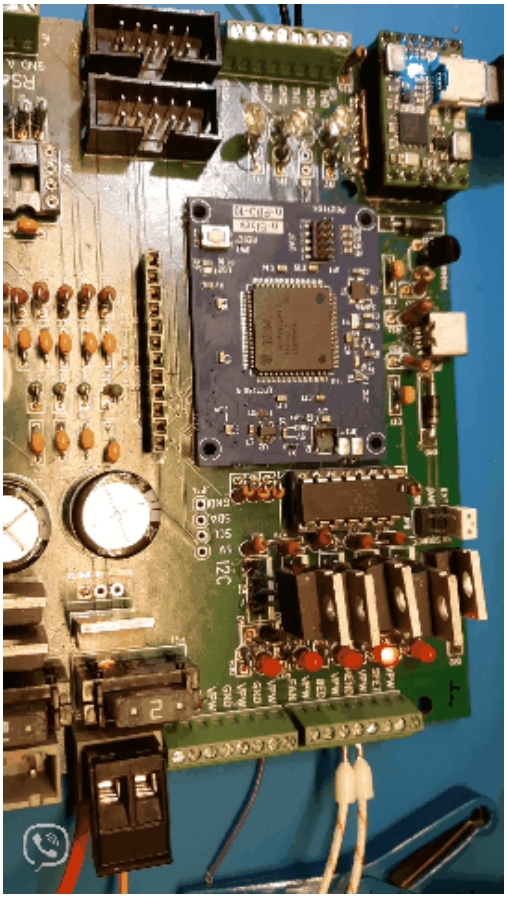
- N1: The KernelNode is configured for mbed
- N3: The ticker is configured to trigger the sensing / control loop, every 200ms
- N4: The thermistor Node is configured for:
  - Microcontroller ADC-pin to be used for measuring the Thermistor voltage.
  - The Thermistor characteristics and Bias
- N5: The PID Node configured with:
  - PID-Coefficients
  - Temperature set at 60 °C
- N6: The PWM Node drives the Heating ellement via a Power-Mosfet from Microcontroller pin P2.5
- N9 and N8 are used for dedugging / Monitoring via USB PC connection and a serial Terminal.

Setup

> Hot-End and an external thermocouple probe for temperature validation (red cable)

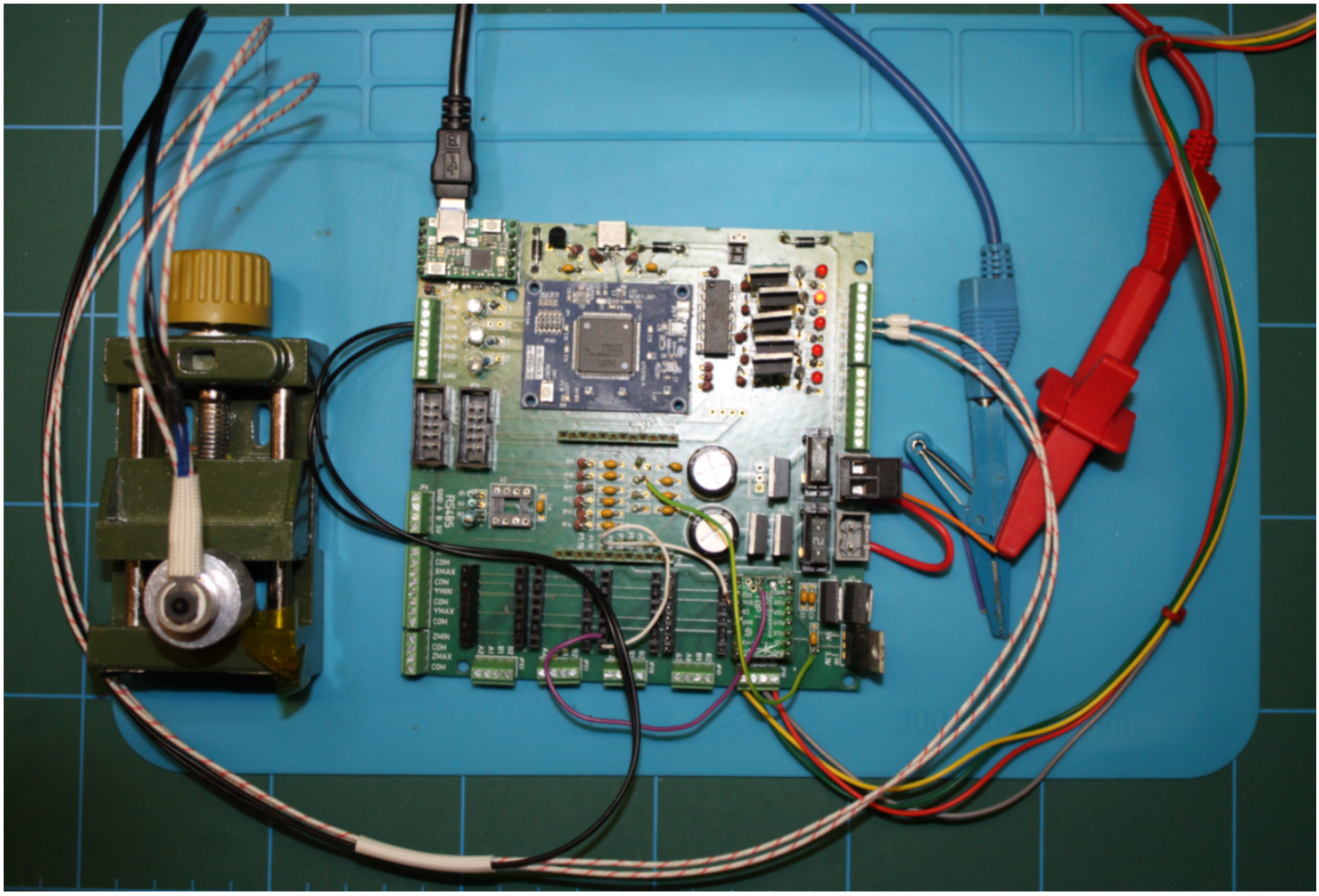


> The PID test program runs in a n-3DP board, with sampling rate 200ms, driving the Hot-End



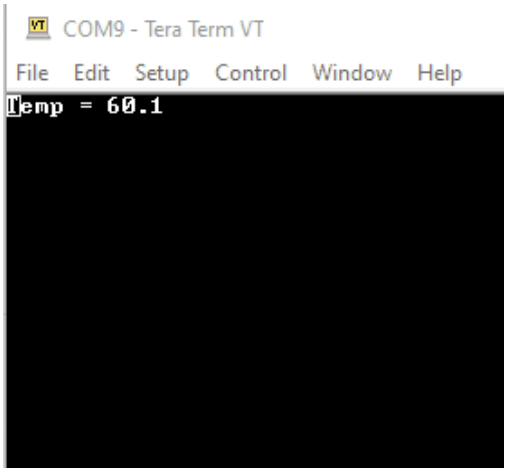
n-3DP motion control/3D printer motherboardc with n-PRO-10 LPC1768 microcontroller board.

- LPC1768 PWM output drives the power MOSFET which powers the Hot-End (white/red pair of cables)
- The Thermistor is connected to the terrminal Block, filtered and then connected to an LPC1768 ADC input (black pair of cables)



Temperature controll results

> Top: Stabilized temeprature at 60 °C measured by Thermistor



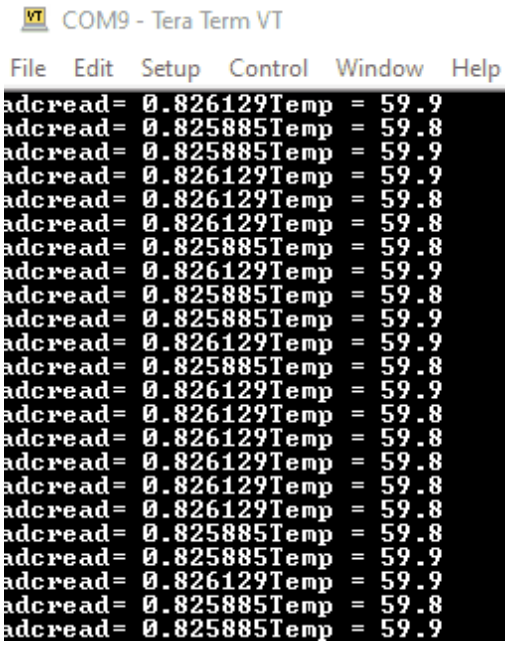
> Middle: Temeperature measured with an external Thermocouple probe

> Bottom: n-3DP supplied with 19V and the the Current consumption is stabilized at 135 mA (Power = 2.5W)

Thermistor Node

Thermistor Node with an added debugging printf(), prints the ADC value for the temperature value it exports.

```
28  void nBlock_Thermistor::endFrame() {
29      if (_read_requested) {
30          _read_requested = 0;
31
32          // This is a normalized value respective to Vdd
33          // (in other words, adc_read=0.3 means 0.3*Vdd)
34          float adc_read = _adc.read();
35          printf("adcread= %f-->\n", adc_read);
36          // Relative current = I / Vdd (normalized to Vdd)
37          // It is the current in [A] if Vdd was 1.0V
38          float relative_current = (1.0-adc_read)/_pullup;
39          float therm_R = adc_read / relative_current;
40
41          // This node uses the beta equation:
42          // 1/T = 1/TO + (1/β) . ln (R/RO)
43          float inv_T = _inv_ref_temp + _inv_beta * log(therm_R / _ref_res);
44          float result = (1.0/inv_T) - T_K;
45
46          output[0] = PackFloat(result);      //output[0] = PackFloat(result/2.5);
47          available[0] = 1;
48      }
49  }
50
```

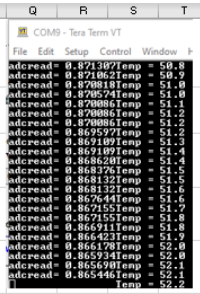


The formula used in the Node C++ code is validated and confirmed using an excel spreadsheet and online Thermistor calculators. The theoretical Temperature value for the corresponding ADC measurement is precise. Measuring with the Multimeter indicates a slight difference from the measured voltage from ADC.

Conclusion on Thermistor Node precision

The Thermistor Node works good, the ADC Hardware front-end needs some improvement, but the precision is still good for a hot-end temperature control.

	T_K	PULLUP	REFRES	refTemp celcius	REFTEMP	BETA	INV_REF_TEMP	INV_BETA	ADC_READ	REL_CURRENT= (1.0-adc_read)/_pullup;	THERMISTOR_R = adc_read / relative_current;	inv_T = _inv_ref_temp + _inv_beta * log(therm_R / _ref_res);	result = (1.0/inv_T)	result = (1.0/inv_T) - T_K;
2	273	4700	100000	25	298	4267	0.003355705	0.000234357	0.1	0.000191489	522.2222222	0.0021242	470.7655534	197.7655534
3	273	4700	100000	25	298	4267	0.003355705	0.000234357	0.5	0.000106383	4700	0.002639134	378.9121906	105.9121906
4	273	4700	100000	25	298	4267	0.003355705	0.000234357	0.8664	2.84255E-05	30479.64072	0.003077263	324.9641124	51.9641124
5	273	4700	100000	25	298	4267	0.003355705	0.000234357	0.8255	3.71277E-05	22234.09742	0.003003339	332.9627126	59.9627126
6	273	4700	100000	25	298	4267	0.003355705	0.000234357	0.95510984	9.5511E-06	100000.0055	0.003355705	297.9999989	24.99999885



https://www.electro-tech-online.com/tools/thermistor-resistance-calculator.php

https://www.giangrandi.org/electronics/ntc/ntc.shtml

```
void nBlock_Thermistor::endFrame() {
    if (_read_requested) {
        _read_requested = 0;

        // This is a normalized value respective to Vdd
        // (in other words, adc_read=0.3 means 0.3*Vdd)
        float adc_read = _adc.read();
        // Relative current = I / Vdd (normalized to Vdd)
        // It is the current in [A] if Vdd was 1.0V
        float relative_current = (1.0-adc_read)/_pullup;
        float therm_R = adc_read / relative_current;

        // This node uses the beta equation:
        // 1/T = 1/TO + (1/β) . ln (R/RO)
        float inv_T = _inv_ref_temp + _inv_beta * log(therm_R / _ref_res);
        float result = (1.0/inv_T) - T_K;

        output[0] = PackFloat(result);
        available[0] = 1;
    }
}
```

NTC Thermistor Resistance Calculator V2.0
Enter Thermistor Datasheet Values.

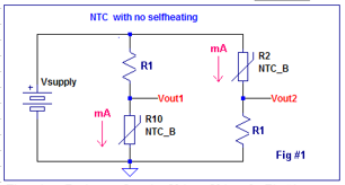
Rnew=R0*exp-B*((1/273+Tref)-(1/273+Tnew))

Thermistor Beta Value: 4267
Resistance at Ref Temperature, Ohms: 100000
Reference Temperature, Cdeg: 25
Target Temperature, Cdeg: 52

Thermistor Resistance Value at Target Temperature
Resistance Value at 52 Cdeg is: 30470 Ohms.

Click to Calculate Thermistor Resistance

Supply Voltage: 3.173
R1. Series Load Resistor, Ohms: 4700



Thermistor/Resistance Junction Voltage Values for Fig #1
Vout1:= 2.75V.... Vout2:= 0.42V. NTC current:=0.09mA

Click to Calculate the Junction Voutputs

NTC parameters:
R_{25} = 100000 Ω
β = 4267 K
Temperature and resistance:
R = 30470 Ω Calculate T
T = 52 °C Calculate R

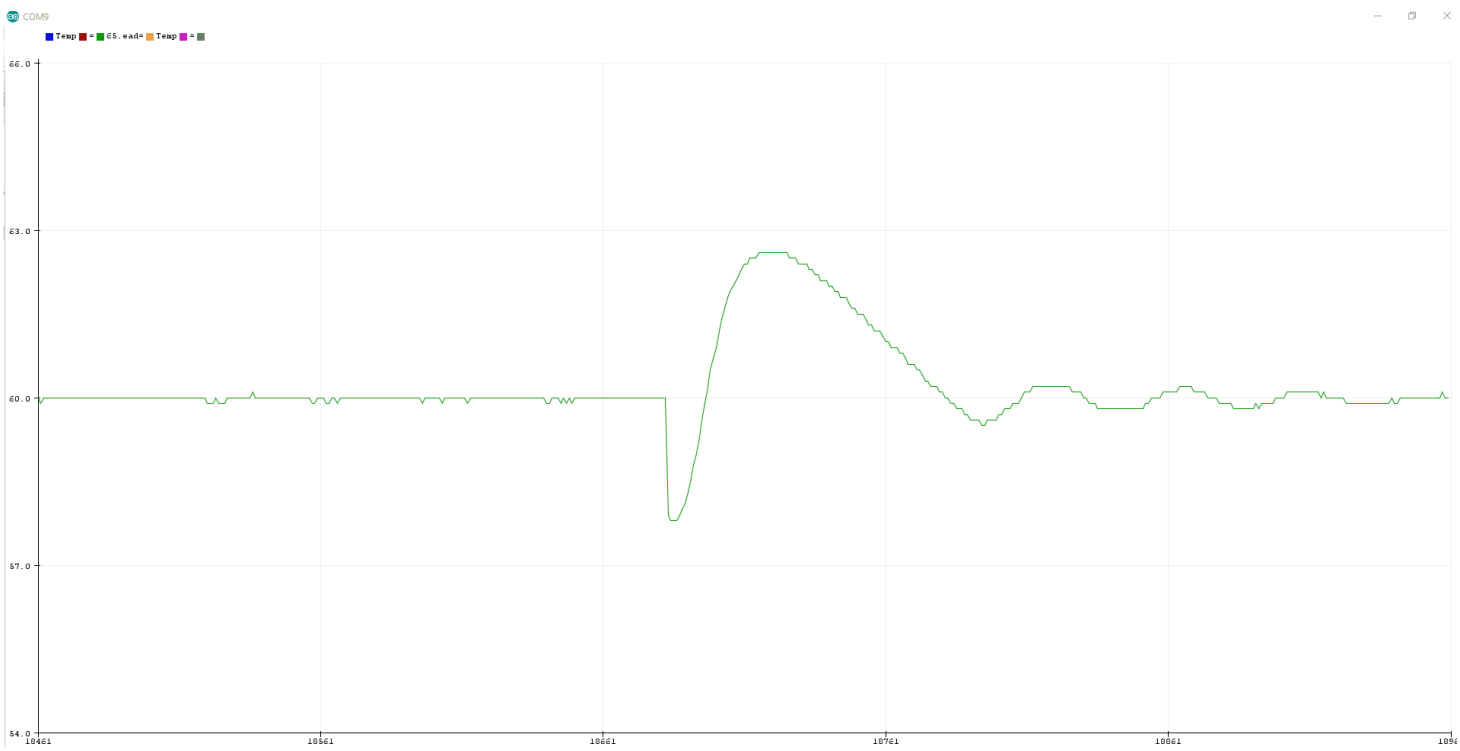
PID control evaluation

```
nBlock_PID          nb_nBlockNode3_PID          (0.2, 0.025, 0.0, 0.2, 0.0, 1.0, AUTOMATIC, DIRECT, 60.0);
```

After trial and error, the values below result in an acceptable controlling performance.

- P = 0.2
- I = 0.025
- D = 0
- SamplingTime = 0.2 sec
- minOutput = 0
- maxOutput = 1
- setPoint = 60 °C

Powered-on at a temperature 40 °C, the PID controller tries to fix the temperature to the 60 °C set point fast, so we have an overshoot, then the system is stabilized with a slight oscillation around the 60 °C value.



Plotted with Arduino plotter.

Disturbing the heat balance, by *just slightly-blowing air for 1 sec*, to the hot-end.
The Sensing and PID system is sensitive enough, to detect and correct the deviation.