Project Work at MPIK

Temperature Control System

Stefan Dickopf

supervised by: Marc Schuh and Jochen Ketter

Contents

1 Introduction **1**

2 Experimental setup **1**
2.1 Temperature sensor circuit 1
2.2 Heater, cooler circuit 3
2.3 Realization of the circuitboard 3

3 Measurements **7**
3.1 Measurements for the heater/cooler shifter circuit 7
3.2 Measurements of temperature sensor circuit 7

4 The temperature control system **8**
4.1 Controlling the PID 8
4.2 Testing the temperature control system 9
4.2.1 Holding the temperature constant 9
4.2.2 Change in temperature reference 11
4.2.3 Turning the lights on 15
4.2.4 Temperature stabilization over several days 16

5 Overview and discussion **18**

6 Appendix **20**
6.1 Miscellaneous 20

1 Introduction

During the 6 week project work the PID temperature control system in the control room of the THe-Trap experiment was to be developed further. A network communication was established using a Ethernet Shield on the Arduino Due platform which acts as the controller. Through this interface it is now possible to read out values and change the behaviour of the PID controller with a script running on a local machine.
The level shifting circuit board for the in- and outputs on the Arduino was reconstructed into a sturdier design using SMD-parts on a milled circuit board.
At last the temperature control was tested using the new network interface.

2 Experimental setup

A temperature sensor, a heater and a cooler are to be connected to a Arduino board. In order for it to work shifters for the voltages are needed such that the ranges fit the inputs/outputs on the Arduino. This was first done by Vanessa Scheller using cables and a breadboard, her design was reworked using a milled circuit board and SMD-parts.

2.1 Temperature sensor circuit

The circuit for the temperature sensor looks as follows:

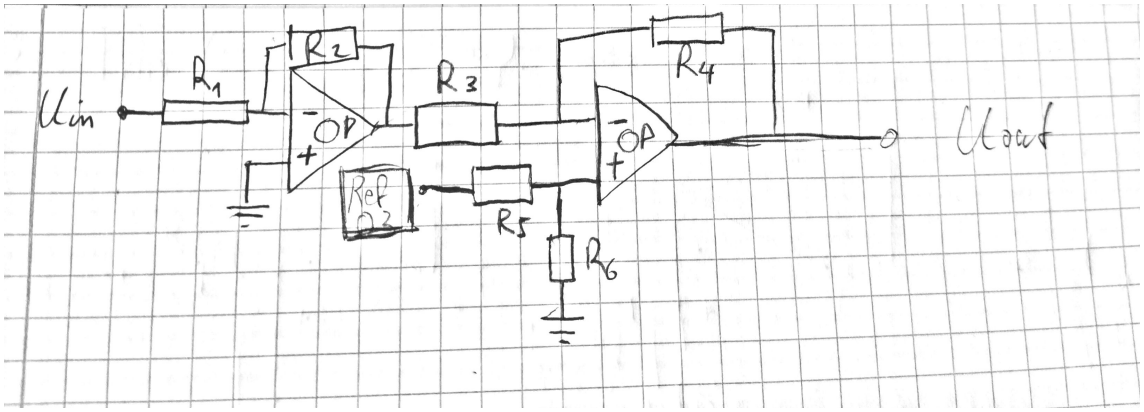


Figure 1: Shifter circuit for the temperature input. The design with two OpAmps ensures that outside the desired input range the output voltages do not change anymore.

The input range from the temperature sensor that is relevant is:

$$-1.6\,\mathrm{V} < \mathrm{U_{in}} < 1.6\,\mathrm{V}$$

The supply voltage of the op-amps is 15 V. A cutoff after the first OpAmp at the input range boundaries is needed such that the output voltage does not exceed the input range of the Arduino. This can be achieved with setting the amplification factor with R_1 and R_2. The output range should equal the input range of the Arduino analogue input which goes from 0-3.3 V. The values found for the resistances are as follows (the added value corresponds to a potentiometer in

series):

	R_1	R_2	R_3	R_4	R_5	R_6
$R[k\Omega]$	15	133	133	$12+5$	10	$15+5$

Table 1: Resistances for temp. shifter circuit

This gives to following response to the input Voltage U_{in}

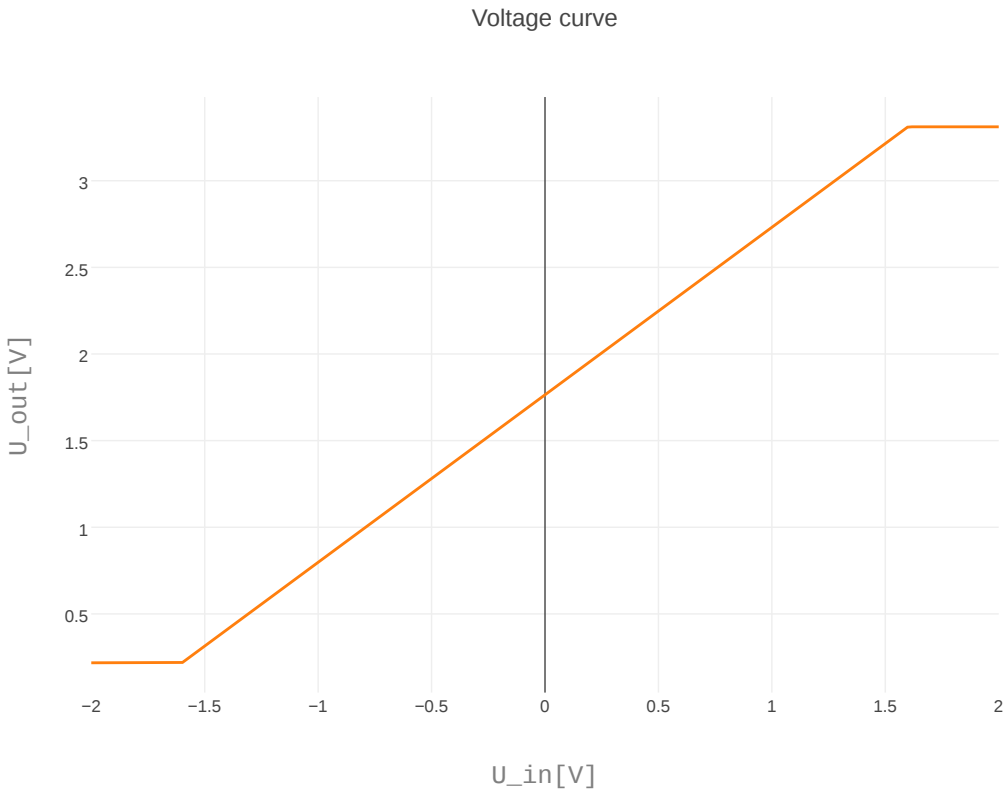


Figure 2: The theoretical response of the temperature shifter circuit shows the linear slope at the input range and the constant output range outside the input range.

2.2 Heater, cooler circuit

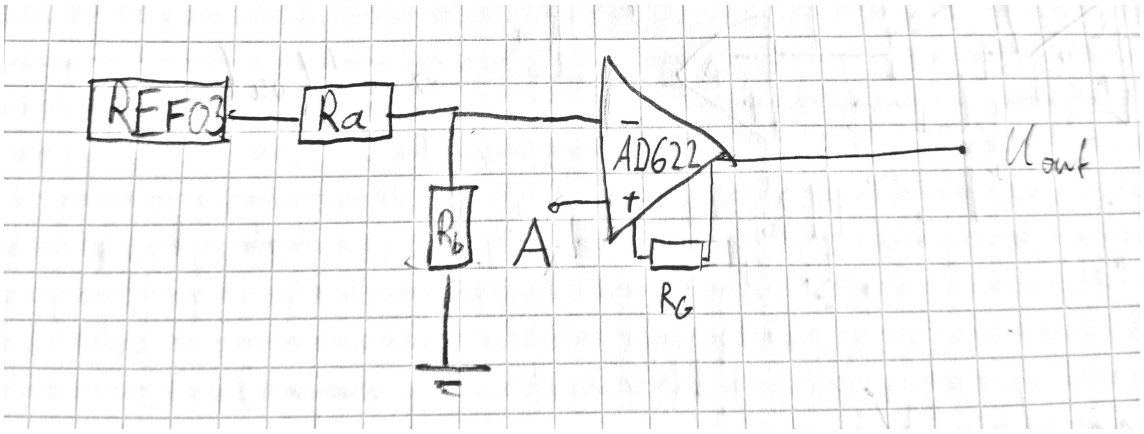


Figure 3: Shifter circuits for the heater and cooler outputs. A reference voltage source REF03 and a voltage dividing circuit are used to subtract the 0.55 V minimum output voltage from the Arduino at the differential amplifier AD622.

The output range from the Arduino's DAC is

$$0.55\,\mathrm{V} < \mathrm{U_{DAC}} < 2.75\,\mathrm{V}$$

The regulation voltage from the heater and cooler goes from 0 V to 10 V. First the voltage from the DAC output needs to be reduced by 0.55 V and then amplified such that the maximum output from the DAC corresponds to the maximum input from the heater/cooler. The amplification is set with the resistance R_G. The values found by Vanessa are:

	R_a	R_b	R_G
R[kΩ]	$3.92 + 5$	1	$27 + 10$

Table 2: Resistances for heater/cooler shifter circuit

The output here can be described by the amplified difference of the two input voltages. The amplification factor was adjusted such that it gives the needed output range. To both circuits many capacitances are added, which are for filtering the input +15 V/-15 V and the reference voltage from AC voltage parts. This is done with 10 nF and 100 nF capacitances in parallel.

2.3 Realization of the circuitboard

All of the circuits were realized using a SMD board. The circuit board was constructed using EAGLE Professional and made with a circuit board cutter. It has the same proportions as the used Arduino Due, such that it fits into the enclosure thats already existing.
The resistors and the capacitances are directly soldered onto the board, while the OpAmp OP07 and the differential amplifier AD622 (not 620 as on the image)

are put onto headers. The complete board is shown here in Figure 5.
Notice that the header for the left AD622 was placed in the opposite direction, such that the marker for the top is actually on the other side than the top is supposed to be.
Also some of the capacitances were not soldered in, on the design of the board they were made as a precaution.
It was found that the heater/cooler cables add a capacitance which lead to oscillations on the temperature measurement. Adding 100Ω resistors on the heater/cooler output resolved the issue.
In Figure 6 the board connected to the Arduino with the Ethernet Shield 2 on top is shown.
After everything on the board is done it is put into the housing in the cabinet.

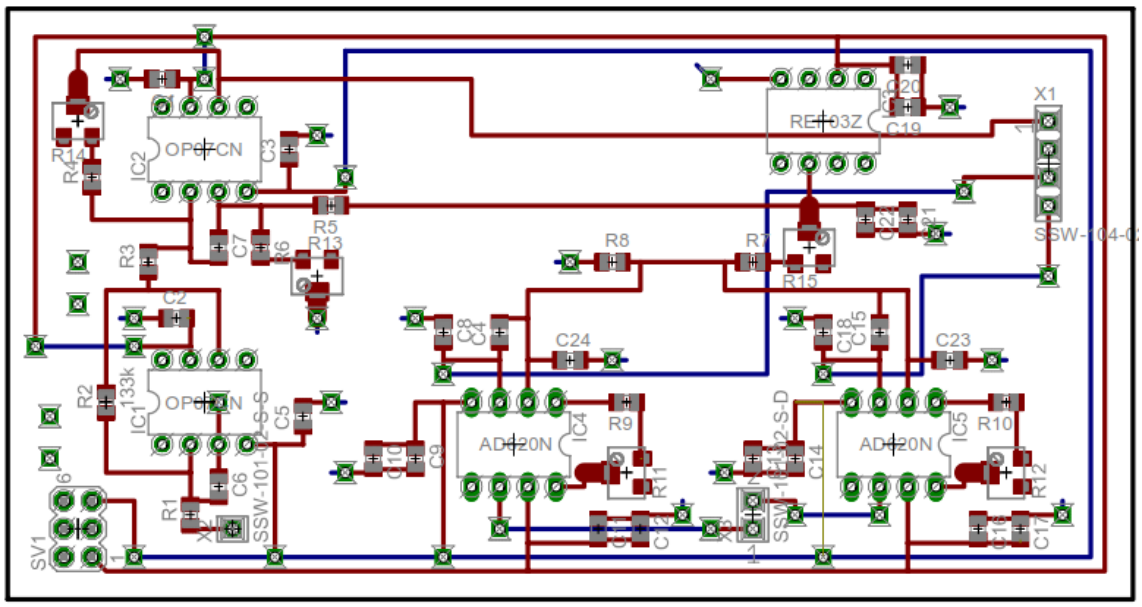


Figure 4: The circuit board layout was done using EAGLE Professional. It consists of two planes.

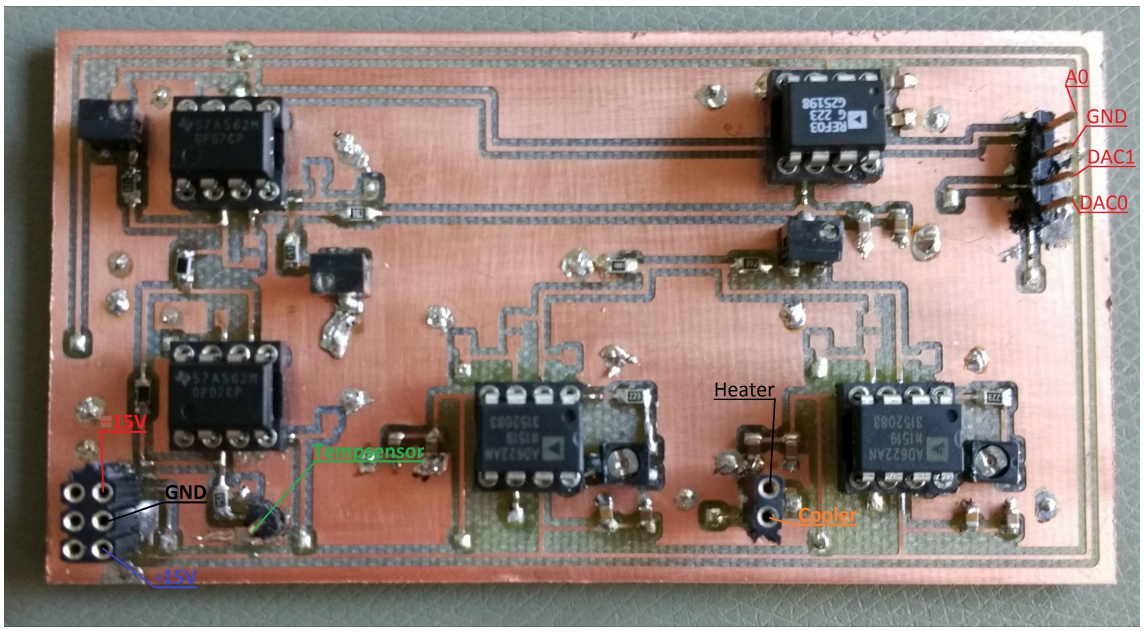


Figure 5: A picture of the circuit board with input / output configuration.

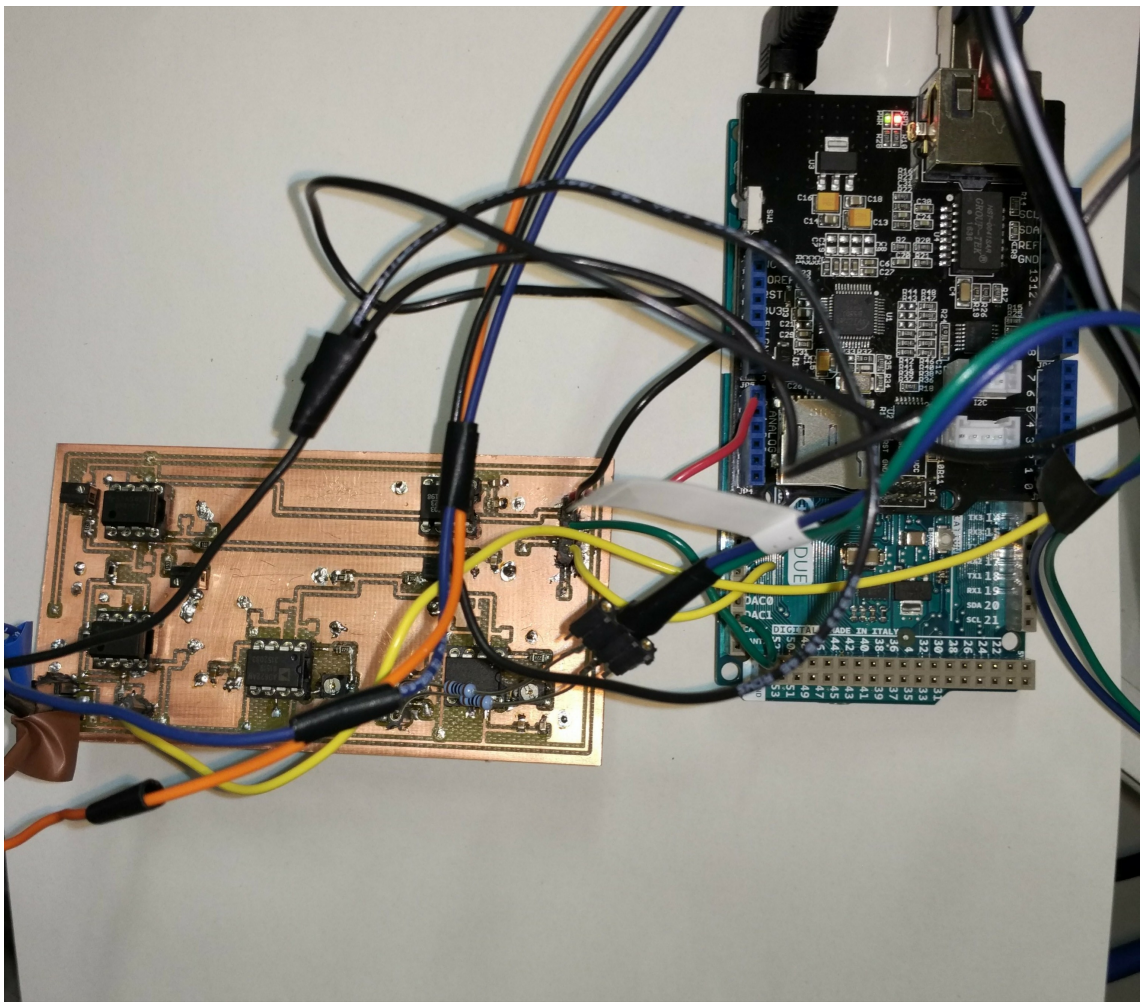


Figure 6: The circuit board connected to the Arduino. On top of the Arduino is the Ethernet Shield 2.

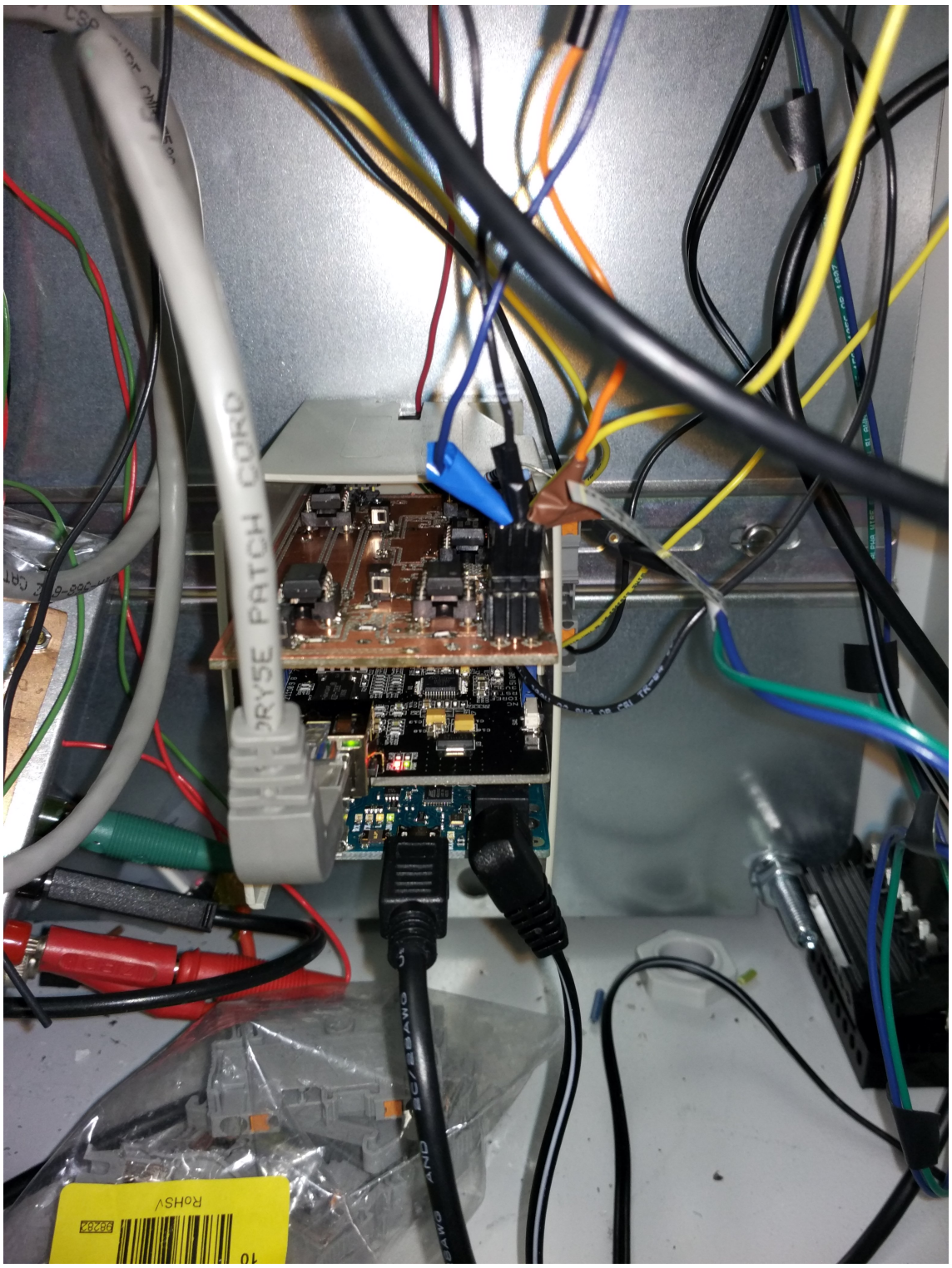
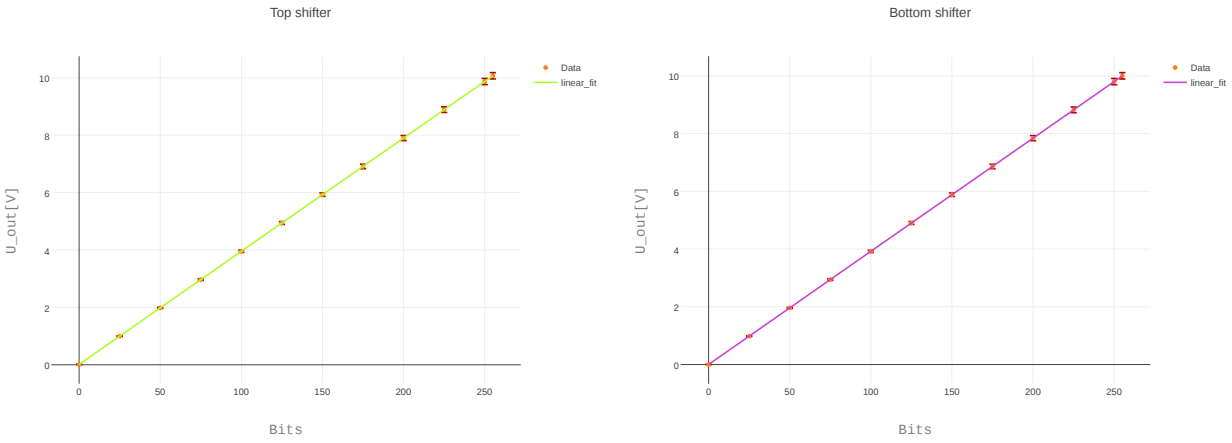


Figure 7: The complete setup in the housing. The USB cable can be used to establish a serial communication with a computer.

3 Measurements

3.1 Measurements for the heater/cooler shifter circuit

First the output voltage from the heater and cooler shifter is measured while changing the input through the Arduino DAC outputs. The DAC outputs use 8Bit resolution, so a measurement is done from 0 to 255 Bits.



(a) (b)

Figure 8: Calibration of the heater and cooler shifter circuits. A good linear behaviour is visible for both, the maximum voltage of 10 V is not exceeded.

3.2 Measurements of temperature sensor circuit

The minimum and maximum values of -1.6 V and 1.6 V should lead to output values of 0 V and 3.3 V. This requirement is used to adjust the potentiometers. The complete voltage curve is shown below.

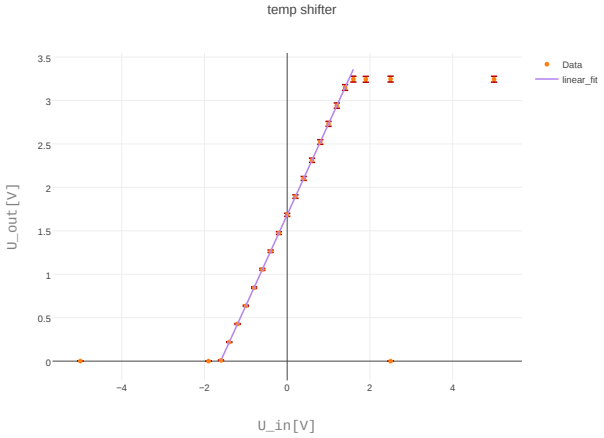


Figure 9: Response of the temperature shifter circuit. It is in good accordance to the theoretical expectation, above and below the input range the output voltage reaches a plateau.

4 The temperature control system

4.1 Controlling the PID

The temperature value gets read out on a Arduino Due platform. It then checks with the temperature reference and calculates control values for the heater / cooler with a PID controller, which was realized as a software running on the Arduino. This was originally developed by Vanessa Scheller.
One of the tasks was to install the Ethernet Shield 2 on the Arduino in order for the PID controller to be able to communicate more easily with a computer. The existing possibility to communicate with the Arduino was to do it over the USB serial interface of the Arduino which allows to read out data and also send data to the Arduino, but requires a computer to be connected to the Arduino at all times. The advantage of a network interface is that it just needs a connected Ethernet cable to the Arduino.
The Ethernet Shield 2 is just connected to the Arduino like any other Arduino shield, by just plugging it on top of the Arduino. The Arduino acts as a server which communicates with a client program running on the tritium-vserver.
The client sends commands directly to the Arduino which get handled there in a defined time interval. The Arduino sends back a response including information.
The client program runs in the background and issues the command to receive the serial output from the Arduino and stores those directly in a file as is, but also creates a THee file with only the time and the temperature values.
The client program can be controlled with the controller script. Running the script with a command line argument makes the client program send the argument as a command to the Arduino. The answer from the Arduino is stored in a log file and also printed on the console. The commands are to be send as follows:

Getting the serial output
Send the command ***g*** to receive the serial output. The values received are:

1. Measurement number (resets every new running day)
2. Mean temperature value over 1 minute. (in 10 bit resolution from 0 to 1023)
3. Difference of current temperature to set value of temperature
4. Difference of mean temperature to set value of temperature
5. Heater set value (in 8 bit resolution from 0 to 255)
6. Cooler set value (in 8 bit resolution from 0 to 255)
7. Days running

Getting the PID parameters and the temperature set value
Send the command ***gpid*** to receive the values K_P, T_N and T_V for both the heater and the cooler as well as the current temperature set value (which is titled sollwert). Also get the state of the PID (1 for turned on, 0 for turned off).

Setting the PID parameters and setting fixed heater/cooler set values
Send the command ***xy###***, where *x* is either *h* or *c* for heater/cooler and *y* is either *p,i,d* for the respective part of the regulator. *y* can also be *f* for a fixed value of the heater/cooler set value (this turns off the PID control). ### corresponds to the wanted value (again in 8 bits from 0 to 255). The Arduino program checks whether the value is in bounds.

Setting the temperature reference
Send the command ***so###*** to set the temperature reference to the value ### (in 10 bits).

Turning PID off and back on
Send the command ***pidon / pidoff*** to turn the PID on or off. If the PID is turned off it will use the fixed values for the set values of the heater and cooler (default is zero for both).

The calibration for the temperature values of the sensor is shown below.

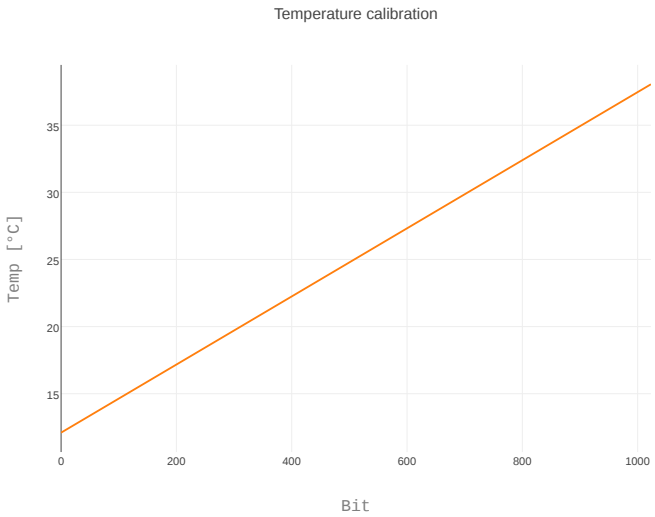


Figure 10: Calibration of temperature sensor done by Vanessa Scheller. The standard value the PID controller regulates to is at 450 Bits, corresponding to 23.5 °C.

4.2 Testing the temperature control system

The board is now connected to the Arduino, the temperature sensor and the heater/cooler. A measurement is done using the already existing sensors at the window and the door.

4.2.1 Holding the temperature constant

To see how well the temperature stabilization works over one day the data from three sensors is plotted. A histogram is used to make an estimate on how much the temperature fluctuates. This is only done for the PID temperature sensor,

because the others show a unexpected behavior. See Figure 11 (a-c) for the temperatures and Figure 11 (d) for the histogram.

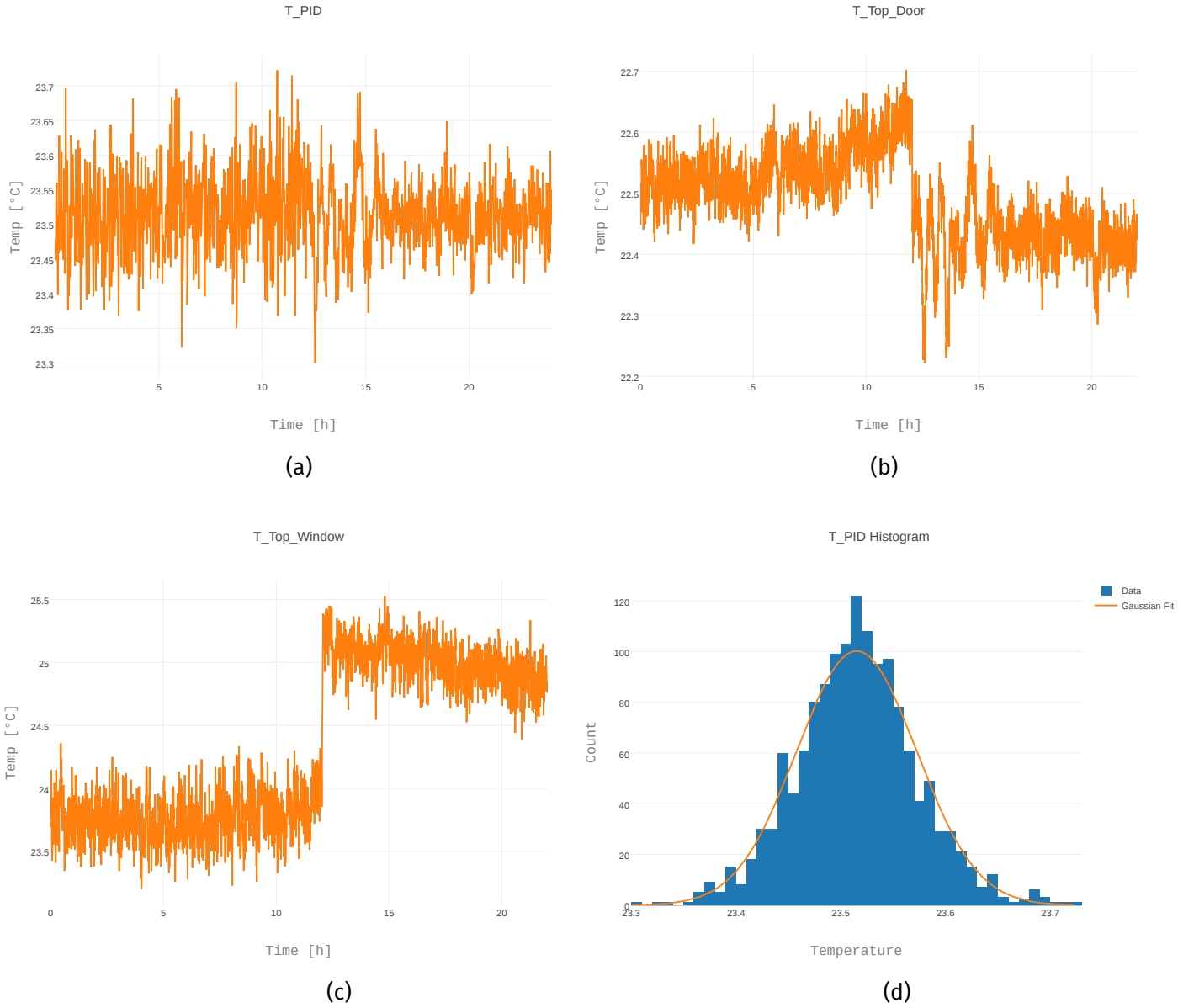


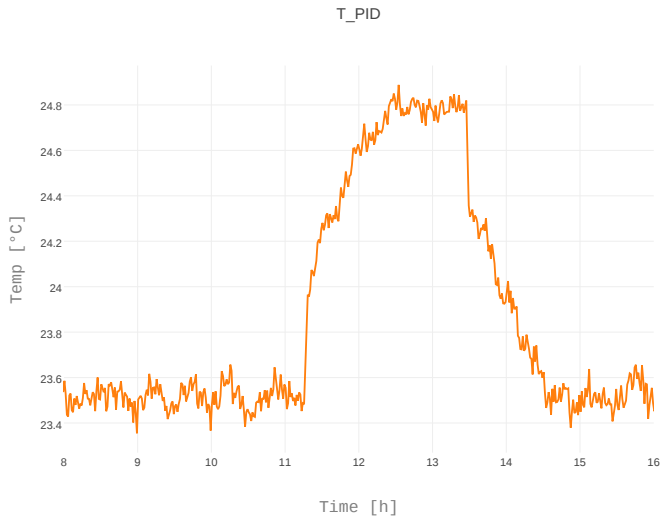
Figure 11: Temperature stabilization over one day with temperature from sensors T_PID, T_Top_Door, T_Top_Window. The PID-sensor is well regulated, while the door and window sensor show a fall/rise in temperature at around 12 o'clock.

The mean of the histogram is at $\mu = 23.515$ °C and the standard deviation at $\sigma = 0.057$ K.
The sensor the PID uses to regulate has a good nearly constant temperature. The fall/rise of the temperature near the door and window can be described as follows. Looking at the cooler/heater values shows that the cooler was shut off up until nearly 12 o'clock and then turned on (probably due to rise in outside temperature), while the heater value decreased.
The heater is located closer to the window and the cooler closer to the door. The rise in outside temperature could be due to sun coming in the windows

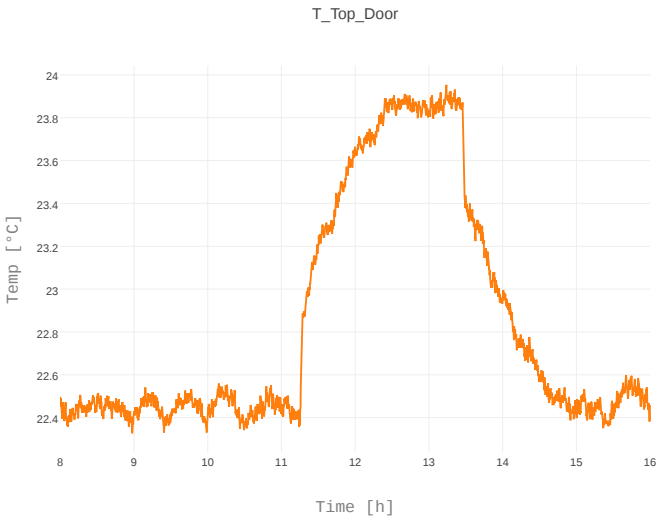
thus raising the temperature there fast. The drop at the door should then be due to the cooler being turned on.

4.2.2 Change in temperature reference

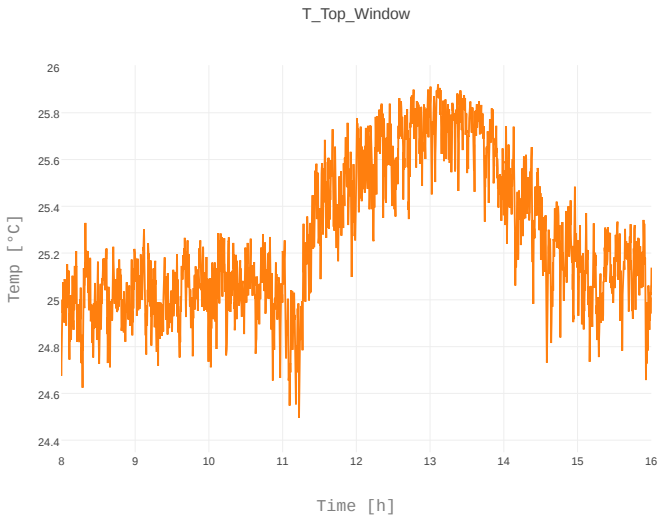
To see how the system reacts to rapid changes in temperature it is much easier to look at a change in the reference temperature. For this the value of the reference temperature is changed from 450 Bits (~ 23.5 °C) to 500 Bits (~ 24.8 °C).



(a)



(b)



(c)

Figure 12: The response in the change of the temperature reference looks much sharper on the door and PID temperature sensors. The window temperature sensor reacts slower and the rise is not as fast.

It's visible that the temperatures at the door and at the PID sensor react very similarly, the temperature at the window seems to react a little slower.
The reaction time can be characterized with the following fit for a first order

system.

$$f(t) = B + A(1 - e^{-t/\tau})$$

for the upwards slope and

$$f(t) = B + Ae^{-t/\tau}$$

for the downwards slope.
For the fit to work errors for the temperature values are needed. A constant error of 0.1 K is assumed. The bounds are set to the time the temperature reference was set.

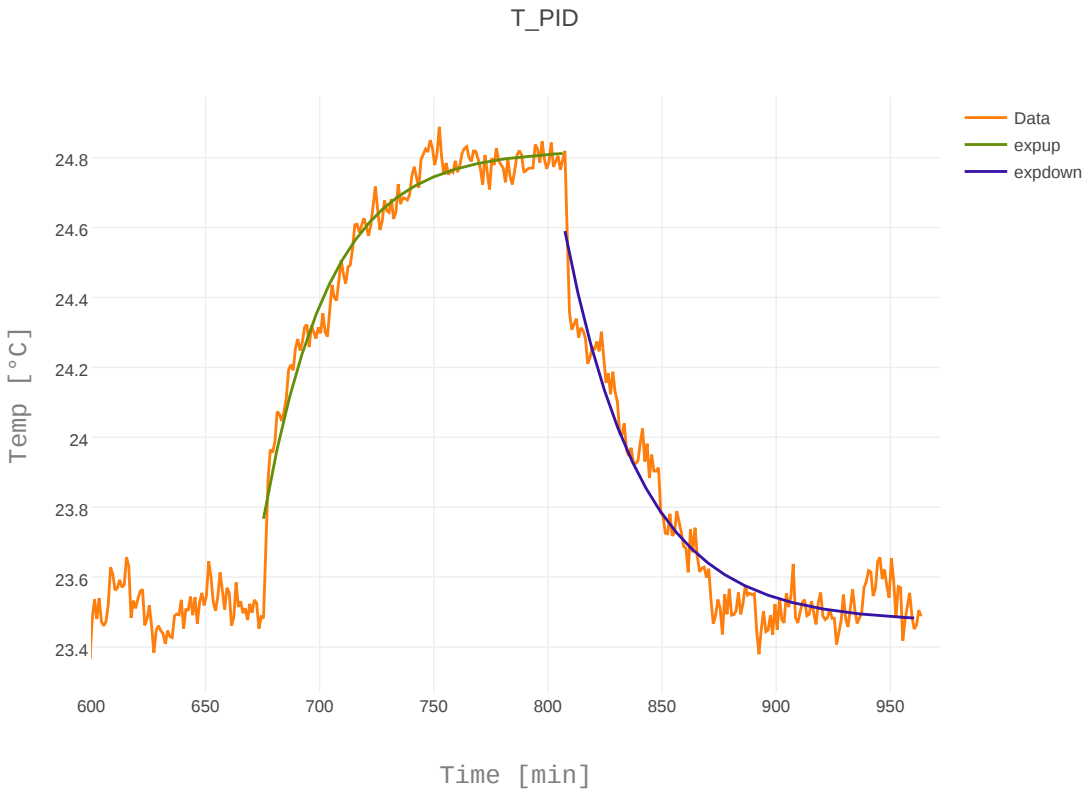


Figure 13: Response of the PID-controller for a change of 50 bits in temperature with fitted exponential rise and falloff. The reduced chi-square values of 0.325 and 0.612 for the rise and falloff make the fit seem plausible.

The time constants are:

$$\tau_{\text{up}} = (28.6 \pm 2.3)\,\text{min}, \quad \tau_{\text{down}} = (33.1 \pm 2.4)\,\text{min}$$

It is also visible that the temperature does not overshoot and does not strongly oscillate.

Another measurement is made to confirm the behavior. This one is done at night (Temperature to 500 Bits at 19:11 and back to 450 Bits at 22:45) (see Figure 14).
The fitting procedure is also repeated (Figure 15). A similar value for the upwards slope is computed. The downwards slope is slightly faster than before,

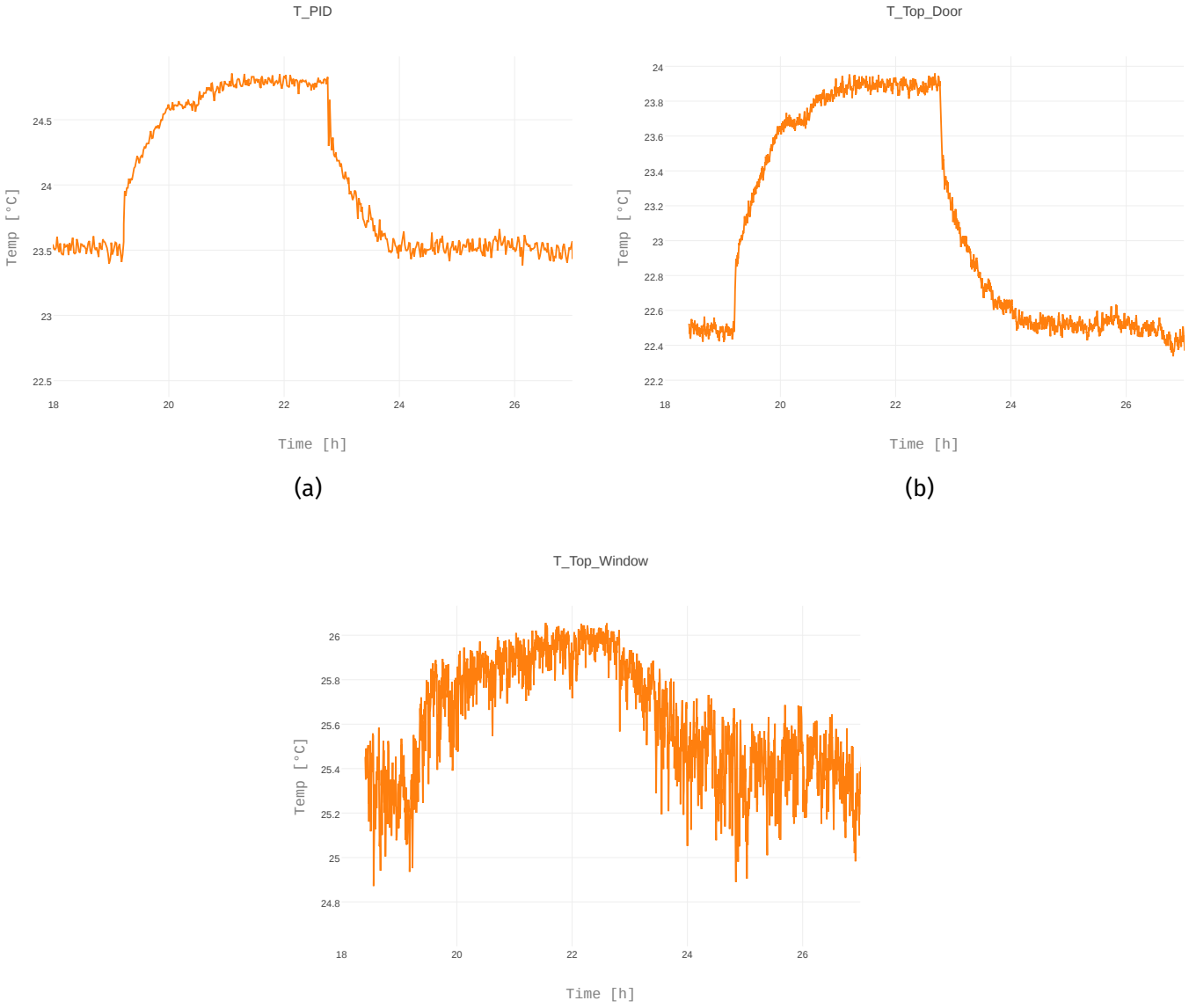
which might be cause by lower outside temperature. Some time after the temperature reaches the lower value again it starts to oscillate heavier. This is also visible for the Door temperature, which is also fitted (Figure 16).
The time constants are

$$\tau_{\text{up}} = (32.5 \pm 2.0)\,\text{min}, \quad \tau_{\text{down}} = (25.4 \pm 1.5)\,\text{min}$$

for the PID sensor and

$$\tau_{\text{up}} = (34.7 \pm 2.0)\,\text{min}, \quad \tau_{\text{down}} = (32.7 \pm 1.6)\,\text{min}$$

for the door sensor.



(a) (b)

(c)

Figure 14: Response to change in in temperature reference. The response again looks much faster at the PID sensor and the door sensor than at the window temperature sensor.

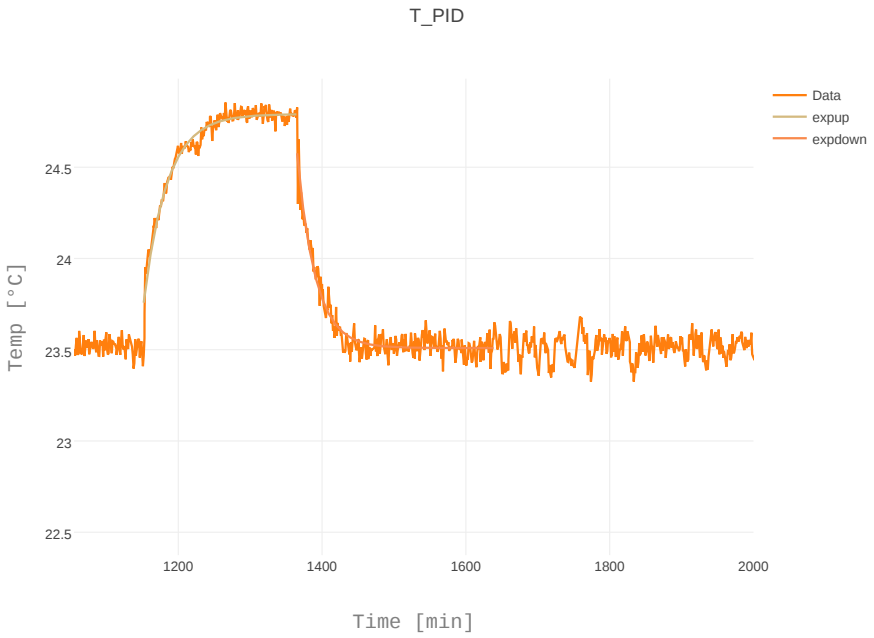


Figure 15: Step response of PID on PID temperature sensor. The response still fits a exponential rise and falloff. After some time the temperature starts oscillating more.

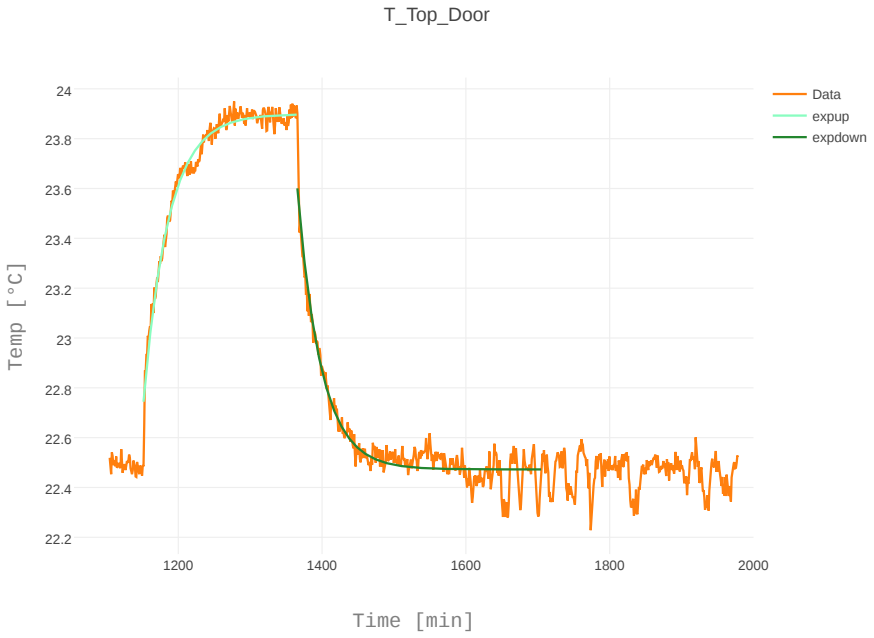
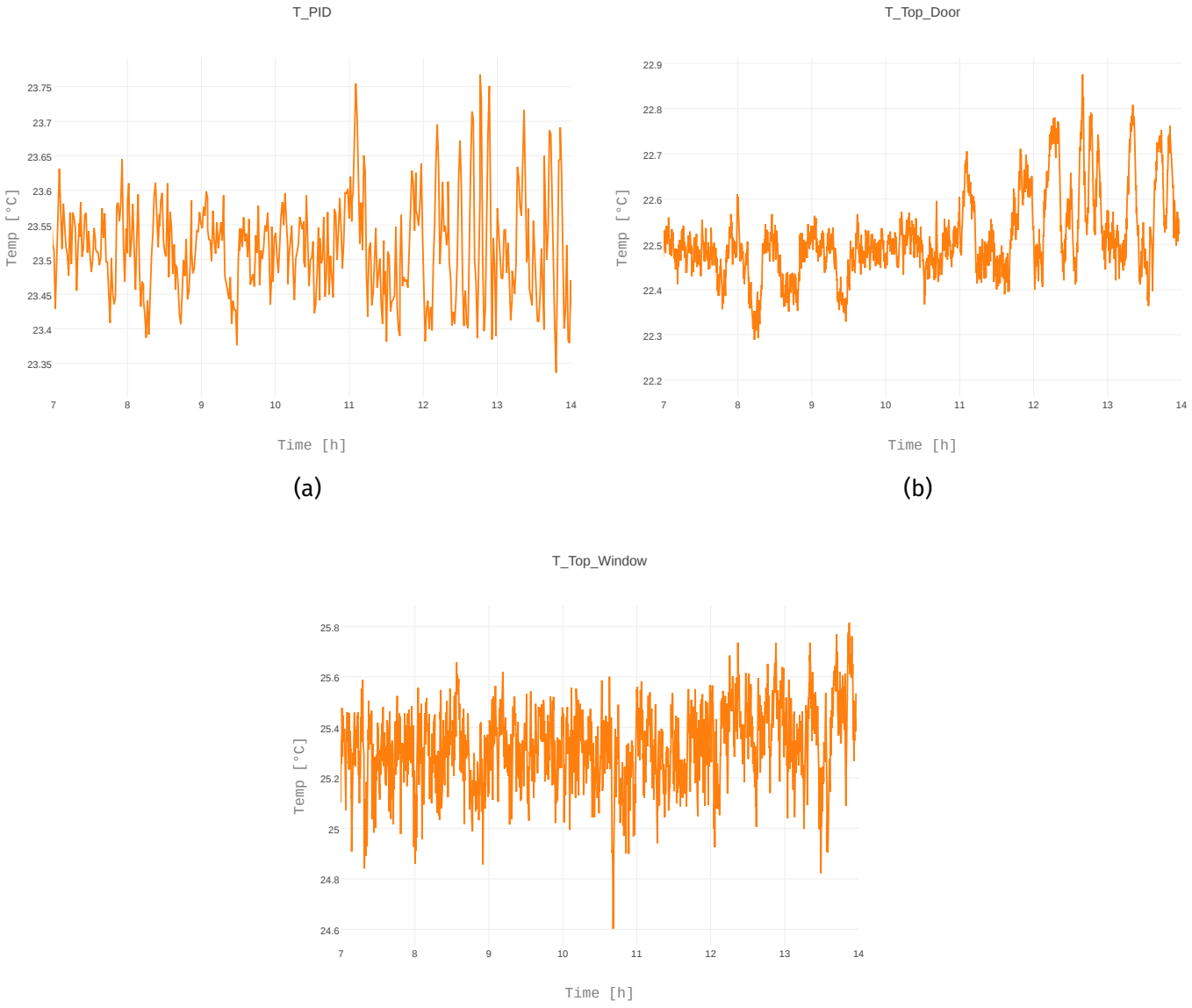


Figure 16: Step response on Top_Door temperature sensor. The response looks mostly like the one on the PID temperature sensor. The oscillations also occur.

4.2.3 Turning the lights on

At 11 o'clock the lights are turned on in the control room. The reaction at the different sensors is shown below (Figure 17), also the set values of the heater/cooler are shown (Figure 18).

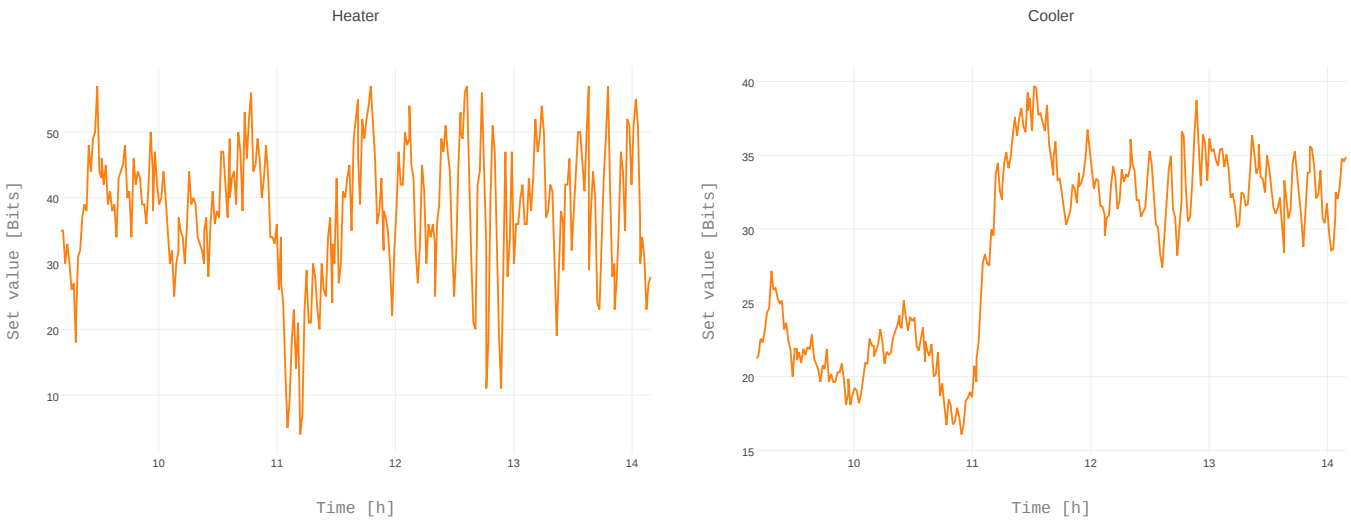


(a) (b)

(c)

Figure 17: Temperature response to light being turned on. Both PID and door temperature sensor show heavier oscillations, while the temperature at the window seems to rise.

Figure 17 shows that the temperature does oscillate more after turning the lights on, especially more to the higher temperatures. Below can be sees that while the heater set value nearly stays the same the cooler starts cooling more after turning the lights on.



(a) (b)

Figure 18: Heater/cooler set values after turning on the lights. Heater value does not change much, cooler set value increases.

4.2.4 Temperature stabilization over several days

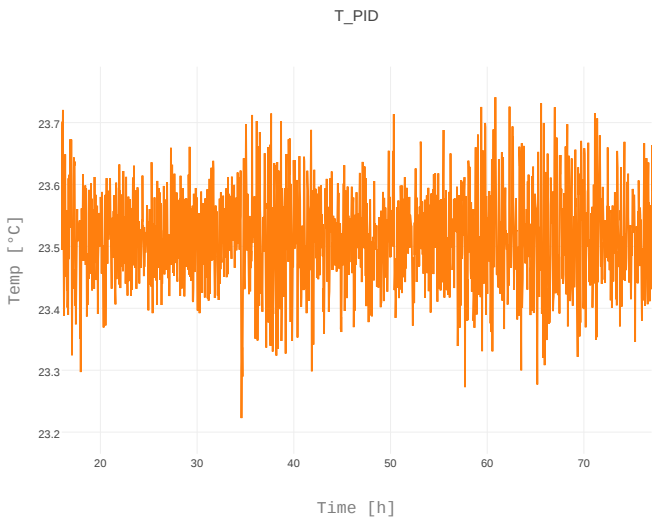
Over the weekend of the 24th to 27th a measurement is made over two and a half days. The measurement starts on the 24th at around 16:30 and ends at the 27th at around 4:30. Here also histograms for the door and window temperatures are plotted, as there is no big jump in them (unlike Figure 11).
Below are the mean and standard deviations for the different sensors, which were also used to plot the gauss curves (Figure 19 (d-f)):

	PID	Door	Window
μ[°C]	23.514	22.652	24.927
σ[K]	0.075	0.073	0.267

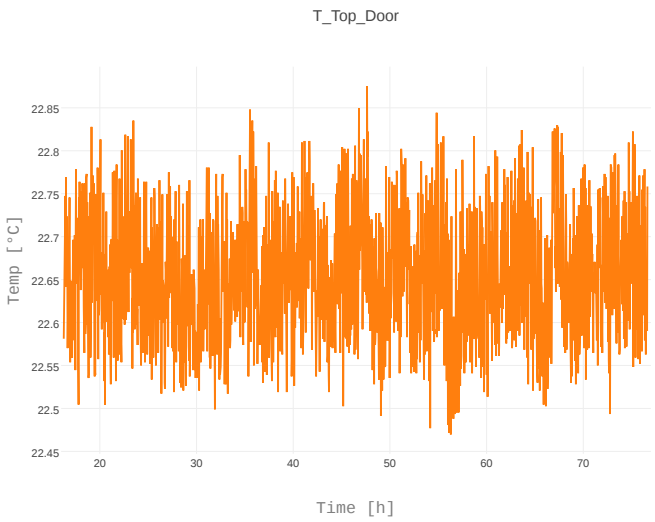
The temperature is fairly stable near the door and the PID temperature sensor, with not even half a degree deviations. The temperature near the window varies more, with a standard deviation of $\sigma = 0.267$ K.
Its also visible that the PID sensor seems to show more oscillation on daytime and at the window there is a clear rise in temperature over the day.
What can also be seen is that the door temperature seems to be rather colder than the mean value, meaning it oscillates more to the cold side and the opposite is the case for the window temperature. This can also be seen by calculating the weighted difference of points to the right to points to the left of the mean. If that number is positive it means that more points are to the left. Calculation yields the following values:

PID	Door	Window
9.67e−04	1.77e−02	−1.05e−02

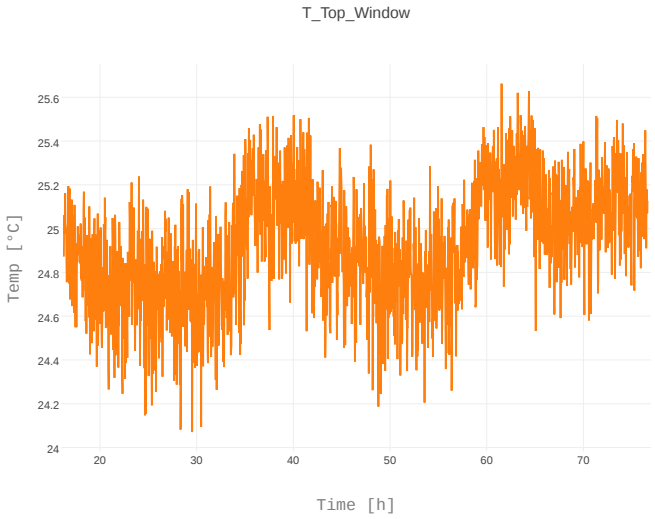
which are consistent with the asumption.



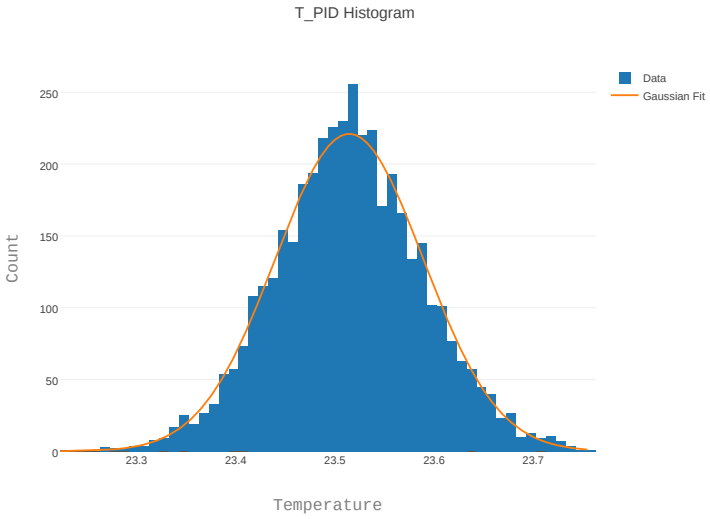
(a)



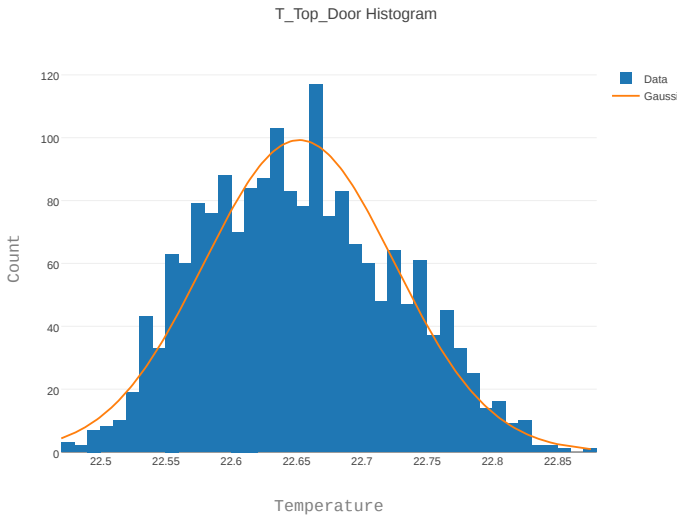
(b)



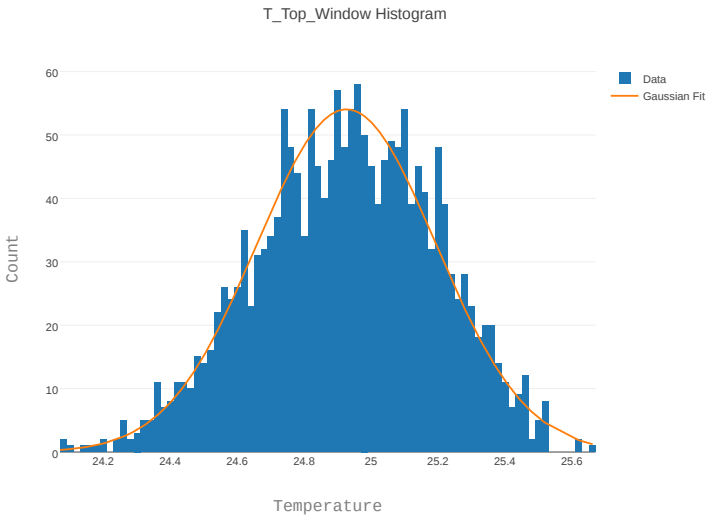
(c)



(d)



(e)



(f)

Figure 19: Temperature stabilization over several days with temp. from sensors T_PID, T_Top_Door, T_Top_Window.

5 Overview and discussion

I started this project work developing a network interface for the PID temperature control system in the control room of the THe-Trap experiment at the Max-Planck für Kernphysik.
The temperature control system was originally developed by Vanessa Scheller using a Arduino Due platform. I worked with her log book and report.
With the original setup it was possible to communicate with the Arduino over the serial interface, but only with a computer connected over the USB-port. To ensure a connection over the local network the most straightforward attempt was installing a Ethernet Shield on top of the Arduino (Figure 6). With this it is possible to get information from the Arduino using a socket communication over a local port. It was not however possible to directly write the data from the Arduino on a network device. The data is send from the Arduino upon request to a handling script on a local computer. The controlling script is described in Controlling the PID.
The network communication could also have been realized using a small single-board computer such as the Raspberry Pi. With this it would have been possible to directly save to data on the network drive and control the PID with commands send to the single-board computer. The disadvantage is that this would make the system more complex and probably more prone to errors.
I continued with reworking the level shifting circuits for the temperature input and heater/cooler set value output. I worked with the schematics from the previous setup and designed a circuit board which could then be realized with a circuit board mill (Figure 4, 5). The new level shifter board should be more failsafe and it also leads to a much tidier setup, because it also fits into the same enclosure as the Arduino board (Figure 7). The complete schematics from Vanessa Scheller are shown in Figure 20. I also used the same resistance values for the resistors for the temperature and the heater/cooler shifter circuit.
With the network interface I was also able to do a more thorough testing of the temperature control than before. The temperature control system should of course hold the temperature as constant as possible under influences that would change the temperature, such as change in daytime temperature. I used three different sensor which were placed at different spots in the control room. The temperature near the door showed a very good constant temperature behavior ($\sigma = 0.073$ K), the temperature near the window does oscillate a little bit more with $\sigma = 0.267$ K (Figure 19).
With changing the temperature reference I could examine the behavior for a rapid change in temperature. With this one can get a good estimate for the time it takes for the system to regulate back to the reference temperature. I calculated time constants which are circa 30 minutes for a change of about 1.3 K , depending on the time of the day and the sensor placement (Figure 12-16).
I also tested the influence of turning the lights on in the control room. The influence on the temperature is visible as bigger oscillation on the sensors (Figure 17) and as a higher cooler set value from the PID (Figure 18). The change in temperature is still rather small.
One thing which the temperature control system could not prevent is a temperature gradient over the room, as it only controls towards the temperature at one fixed place in the room. This can be seen in Figure 11. Here the temperature near the door and the window make a fast change probably due to local change

in temperature (i.e. light from the window) and are not regulated back, but stay at the higher/lower level. This effect was probably increased by the placement of the heater and cooler systems, with the heater being near the window and the cooler being near the door.
Concluding it can be said, that the PID temperature control system works fairly well and can also withstand influences such as change in outside temperature, lights being turned on and a rapid change in temperature, but it can happen that it builds up a temperature gradient. Whether or not this drawback does actually influence the results of the experiment (or whether the temperature control is at all necessary) remains to be seen.

6 Appendix

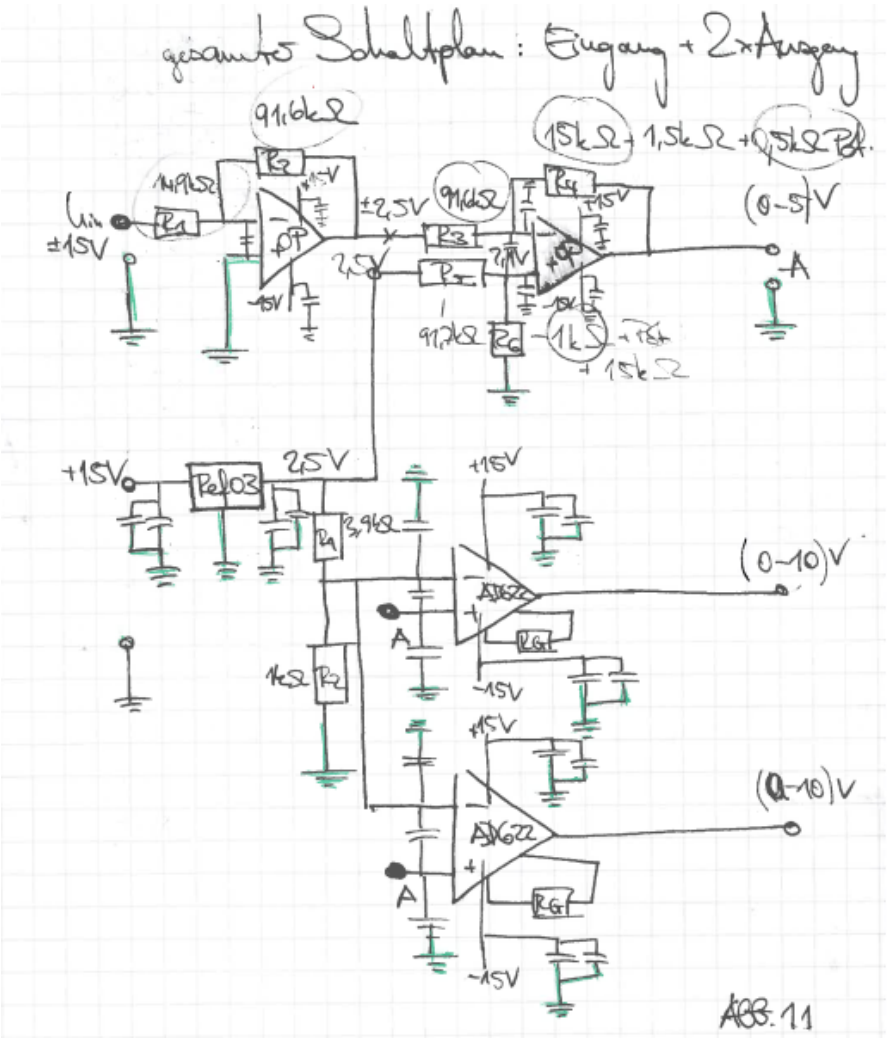


Figure 20: Circuit board schematics from Vanessa Scheller found in her lab book. The resistance values are not the ones finally used. These are found in Table 1 and Table 2.

6.1 Miscellaneous

- The client script, controller script and the Arduino program are located in the tritium drive in the folder Stefan.
- In the file howto.txt in the tritium drive (same folder) a small explanation on how to start the control system can be found.
- The current IP-Adress of the Arduino is 149.217.90.48
- Most of the materials for the temperature stabilization are also found on the THee-Wiki:
  `https://www.mpi-hd.mpg.de/THeWiki/index.php/Temperature_stabilization`