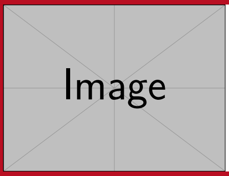
Development of Laser Systems and Spectroscopy of Highly Charged Ions

Zur Erlangung des Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.)
Vorgelegte Dissertation von Patrick Baus aus Mannheim
Tag der Einreichung: 29. September 2022, Tag der Prüfung: 29. September 2022

1. Gutachten: Prof. Dr. Gerhard Birkl
Foobar



Physics Department
Institut für Angewandte Physik
APQ

Development of Laser Systems and Spectroscopy of Highly Charged Ions

Submitted doctoral thesis by Patrick Baus

1. Review: Prof. Dr. Gerhard Birkl

Date of submission: September 29, 2022
Date of thesis defense: September 29, 2022

Foobar

1 Preparation

1.1 Grounding and Shielding

Add parts from ”references
Grounding and Shielding.pdf”

1.2 Laser Current Driver

1.2.1 Design

Simulation

Op Amp Stability

1.2.2 Noise Considerations

1.2.3 Voltage Reference

1.2.4 MOSFET Selection

1.3 LabKraken

1.3.1 Design Goals

LabKraken is a designed to be a asynchronous, resilient data aquisition suite, that scales to thousands of sensors and accross different networks.

1.3.2 Hardware

1.3.3 Software Architecture

LabKraken needs to scale to thousands of sensors, which need to be served concurrently. This problem is commonly referered to as the C10K problem as dubbed by Dan Kegel back in 1999 [3] and refers to serving 10 000 concurrent connections via network sockets. While today millions of concurrent connections can be handled by servers, handling 10 000 can still be challenging, especially, if the data sources are heterogeneous as is typical for sensor networks of different sensors from different manufacturers.

In order to meet the design goals, an asynchronous architecture was chosen and several different architectures were implemented over time. All in all four complete rewrites of the software were made to arrive at the architecture presented here. The reason for the rewrites is mostly historic and can be explained by the history of the programming language Python, which was used to write the code. The first first version was written for Python 2.6 and exclusively supported sensors made Tinkerforge. In 2015, Python 3.5 was released, which supported a new syntax for asynchronous coroutines. The software was rewritten from scratch to support this new syntax, because it made the code a lot more verbose and easier to follow. With the release of Python 3.7 in 2018 asynchronous generator expressions where mature enough to be used in productions and the programm was again rewritten to use the new syntax. In 2021 a new approach was taken and the programm was once more rewritten with a functional programming style. I will discuss each approach in the next sections to highlight the improvements, that were made over time. Each of these sections discusses the same programm, but written in different styles to show the differences.

Threaded Design

The first version of LabKraken used a threaded design approach, because the original libraries of the Tinkerforge sensors are built around threads. The following simplified example shows some code to connect to a temperature sensor over the network and read its data.

```
ipcon = IPConnection()
devices = dict()

# Callback function for temperature callback
def cb_temperature(temperature):
    print("Temperature: " + str(temperature/100.0) + " degC")

def cb_connected(connect_reason)::
    ipcon.enumerate()

def cb_disconnected(disconnect_reason)::
    log_reason(disconnect_reason)

def cb_enumerate(uid, *_args):
    if uid == OUR_KNOWN_DEVICE:
        dev = BrickletTemperatureV2(uid, ipcon)
        # Register temperature callback to function cb_temperature
        dev.register_callback(dev.CALLBACK_TEMPERATURE, cb_temperature)
        dev.set_temperature_callback_configuration(1000, False, "x", 0, 0)
        devives[uid] = dev

if __name__ == "__main__":
    ipcon.connect(HOST, PORT)  # blocking call
    # Register Enumerate Callback
    ipcon.register_callback(IPConnection.CALLBACK_ENUMERATE, cb_enumerate)
    ipcon.register_callback(IPConnection.CALLBACK_CALLBACK_CONNECTED,
                                               cb_connected)
```

Device Identifiers

Every sensor network needs device identifiers. Preferably those identifiers should be unique. Typically a device has some kind of internal indetifier. Here are a few examples of the sensors used in our network:

Device Type	Identifiers	Example
GPIB (SCPI)	**IDN?* returns $manufacturer,$name,$serial,$revision	
Tinkerforge	Each sensor has a base58 encoded integer device id	QE9 (163684)
Labnode	Universal Unique Identifier (UUID)	cc2f2159-e2fb-4ed9-8021-7771890b37ad

As it can be seen above, these identifiers do not guarantee to uniquely identify a device within a network. The Tinkerforge id is the weakest, as it is a 32 bit integer (4.294.967.295 options), which might easily collide with another id from a different manufacturer. The tinkerforge id is presented as a base58 encoded string. An encoder/decoder example can be found in the TinkerforgeAsync library [2].

The id string returned by a SCPI device is slightly better, but again does not guarantee uniqueness. As it is shown in the example the same device might return a different id defpending on its settings. This typically done by manufacturers for compatibility reasons.

The only reasonably unique id is the universal unique identifier (UUID) or globally unique identifier (GUID), as dubbed by Microsoft, used in the Labnodes. Their id can be used for networks with participant numbers going into the millions.

Calculating the probability of a collision between two random UUIDs is called the birthday problem [11] in probability theory. A randomly generated version 4 UUID of variant 1 as defined in RFC 4122 [4] has 122 bit of entropy, that is out of 128 bit, 4 bit are reserved for the UUID version and 2 bit for the variant. This gives the probability of at least one collision in n devices out of $M = 2^{122}$ possibilities:

$$\begin{aligned} p(n) &= 1 - 1 \cdot \left(1 - \frac{1}{M}\right) \cdot \left(1 - \frac{2}{M}\right) \ldots \left(1 - \frac{n-1}{M}\right) \\ &= 1 - \prod_{k=1}^{n-1} \left(1 - \frac{k}{M}\right) \end{aligned} \tag{1.1}$$

Using the Taylor series $e^x = 1 + x \ldots$, assuming $n \ll M$ and approximating we can simplify this to:

$$\begin{aligned} p(n) &\approx 1 - \left(e^{\frac{-1}{M}} \cdot e^{\frac{-2}{M}} \ldots e^{\frac{-(n-1)}{M}}\right) \\ &\approx 1 - \left(e^{\frac{-n(n-1)/2}{M}}\right) \\ &\approx 1 - \left(1 - \frac{n^2}{2M}\right) = \frac{n^2}{2M} \end{aligned} \tag{1.2}$$

For one million devices, this gives a probability of about 2×10^{-25}, which is negligible.

In the Kraken implementation, all devices, except for the Labnodes, will be mapped to UUIDs using the underlying configuration database. It is up to the user to ensure the uniqueness of the non-UUID ids reported by the devices to ensure proper mapping.

Limitations

There is one inherent limitation to the ethernet bus for instrumentation. The ethernet bus is inherently asynchronous and multiple controllers can talk to the device at the same time. Not only that, but different processes within the same controller can talk to the same device. This makes deterministic statements about the device state challenging.

While it is impossible to rule out the possibility of multiple controllers on a network, care was taken to synchronize the workers within Kraken.

1.3.4 Databases

Cardinality

- TimescaleDB vs Influx

- Example Sensors vs. Experiment

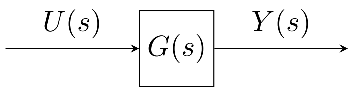
1.4 Short Introduction to Control Theory

This section will give a very brief introduction into some basic concepts of control theory. Many systems require control over one or more process variables. For example, temperature control of a room or a device, or creating a current from a voltage. All of this requires control over a process and is established trough feedback, which allows a controller to sense the state of the system.

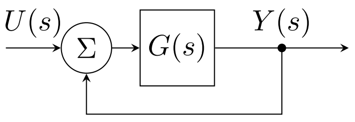
The focus of this section lies on the principels feedback and control and will be detailed in the following sections.

1.4.1 Open and Closed Loop Systems

To understand feedback, one needs to take a look at dynamical systems. There are two types of systems: open and closed loop systems. A system is called open loop, if the output of a system does not influece its input as in figure 1.1a. On the other hand, if the output is connected to the input of the system it is called closed loop system, an example is shown in figure 1.1b. $G(s)$ is called the transfer function of the system, while $R(s)$ is the input, $Y(s)$ is the output and s the Laplace variable.



(a) Open loop system.



(b) Closed loop system.

It is convenient to express the transfer function as its Laplace transform. The unilateral Laplace transform is definded as:

$$\mathscr{L}(f(t)) = F(s) = \int_0^\infty f(t)e^{-st}\,dt. \tag{1.3}$$

with $f : \mathbb{R}^+ \to \mathbb{R}$, that is integrable and grows no faster than $e^{s_0 t}$ for $s_0 \in \mathbb{R}$. The latter property is important for deriving the rules of differentiation and integration.

To understand the benefits of using the Laplace representation for transfer function a few useful properties must be discussed. First of all the Laplace transform is linear:

$$\begin{aligned}\mathscr{L}\left(a\cdot f(t)+b\cdot g(t)\right) &= \int_0^\infty (a\cdot f(t)+b\cdot g(t))e^{-st}\,dt\\ &= a\int_0^\infty f(t)e^{-st}\,dt + b\int_0^\infty g(t)e^{-st}\,dt\\ &= a\mathscr{L}\left(f(t)\right)+b\mathscr{L}\left(g(t)\right) \qquad (1.4)\end{aligned}$$

Another interesting property is the derivative and integral of a function f:

$$\begin{aligned}\mathscr{L}\left(\frac{df}{dt}\right) &= \int_0^\infty \underbrace{f'(t)}_{v'(t)}\,\underbrace{e^{-st}}_{u(t)}\,dt\\ &= \left[e^{-st}f(t)\right]_0^\infty - \int_0^\infty (-s)f'(t)\,dt\\ &= -f(0)+s\int_0^\infty f'(t)\,dt\\ &= sF(s)-f(0) \qquad (1.5)\end{aligned}$$

$$\begin{aligned}\mathscr{L}\left(\int_0^t f(\tau)\,d\tau\right) &= \int_0^\infty \left(\int_0^t f(\tau)\,d\tau e^{-st}\right)\,dt\\ &= \int_0^\infty \underbrace{e^{-st}}_{v'(t)}\underbrace{\int_0^t f(t)\,d\tau}_{u(t)}\;dt\\ &= \left[\frac{-1}{s}e^{-st}\int_0^t f(t)\,d\tau\right]_0^\infty - \int_0^\infty \frac{-1}{s}e^{-s\tau}f(\tau)\,d\tau\\ &= 0+\frac{1}{s}\int_0^\infty e^{-s\tau}f(\tau)\,d\tau\\ &= \frac{1}{s}F(s) \qquad (1.6)\end{aligned}$$

If the initial state $f(0)$ can be chosen to be 0, the differentiation becomes a simple multiplication by s, while the integration becomes a division by s. Finally, the most important aspect is, that a simple relation between the input $r(t)$ and the ouput $y(t)$ of a system can be given. The relation between input and the ouput of a system as shown in figure 1.1a is given by the convolution, see e.g. [1]. Assuming the system has an initial state of 0 for $t < 0$, hence $r(t < 0) = 0$ and $g(t < 0) = 0$, one can calculate:

$$y(t) = (r * g)(t) = \int_0^\infty r(\tau)g(t-\tau)\,d\tau \tag{1.7}$$

Applying the Laplace transformation, greatly simplifies this:

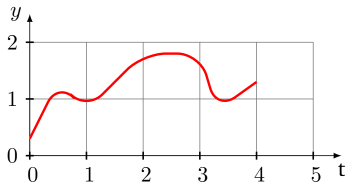
$$\begin{aligned} Y(s) &= \int_0^\infty e^{-st} y(t)\,dt \\ &\overset{1.7}{=} \int_0^\infty \underbrace{e^{-st}}_{e^{-s(t-\tau)}e^{-s\tau}} \int_0^\infty r(\tau)g(t-\tau)\,d\tau\,dt \\ &= \int_0^\infty \int_0^t e^{-s(t-\tau)}e^{-s\tau} g(t-\tau)r(\tau)\,d\tau\,dt \\ &= \int_0^\infty e^{-s\tau} r(\tau)\,d\tau \int_0^\infty e^{-st} g(t)\,dt \\ &= R(s)\cdot G(s) \end{aligned} \tag{1.8}$$

This formula is a lot simpler than the convolution of $r(t)$ and $g(t)$, therefore the use of the Laplace transform has become very popular in control theory.

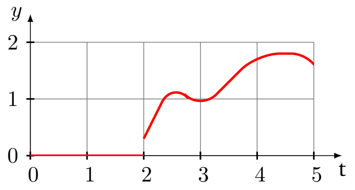
Another property that is heavily used in control theory is the time delay of functions. To show this property, let $f(t-\theta)$ be

$$g(t) := \begin{cases} f(t-\theta), & t \geq \theta \\ 0, & t < \theta \end{cases} \tag{1.9}$$

The reason for this definition is, that the system must be causal. This means, it is impossible to get data from the future ($t < \theta$). An example is shown in figure 1.2a.



(a) Original signal $f(t)$.



(b) Delayed signal $f(t-2)$.

The Laplace transform of a delayed signal can be calculated as follows:

$$\begin{aligned}\mathscr{L}(g(t)) &= \int_0^\infty f(t-\theta)e^{-st}\,dt \\ &\overset{1.9}{=} \int_\theta^\infty f(t-\theta)e^{-st}\,dt \\ &\overset{u:=t-\theta}{=} \int_0^\infty f(u)e^{-s(u+\theta)}\,du \\ &= e^{-s\theta}\int_0^\infty f(u)e^{-su} \\ &= e^{-s\theta}F(s)\end{aligned} \tag{1.10}$$

To satisfy the causaulity requirement, the Heaviside function $H(t)$ can be used:

$$\mathscr{L}(f(t-\theta)H(t-\theta)) = e^{-s\theta}F(s) \tag{1.11}$$

Lastly, the Laplace transform of e^{at}, which is commonly used in differential equations:

$$\mathscr{L}\left(e^{at}\right) = \int_0^\infty e^{(a-s)t}\,dt = \frac{1}{a-s}\left[e^{(a-s)t}\right]_0^\infty = \frac{1}{s-a} \tag{1.12}$$

Using these tools, it is possible calculate the transfer function of a temperature controller. This is done in the next section.

1.4.2 A Model for Temperature Control

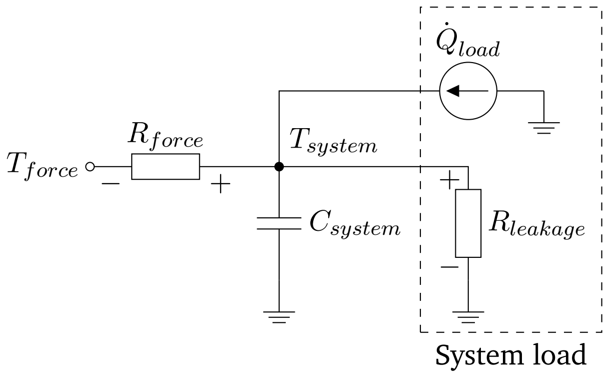
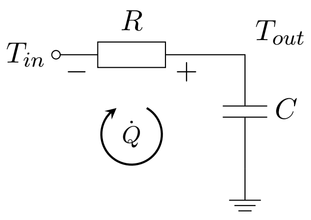


Figure 1.3: Simple temperature model of a generic system.

In order to describe a closed-loop system, one has to first create a model for the process and the controller involved. A simple model can be derived from the idea, that the system at temperature T_{system} has a thermal capacitance C_{system}, an influx of heat $\dot{Q}_{load}$ from a thermal load and a controller removing heat from the system through a heat exchanger with a resistance of R_{force}. Additionally, there is some leakage through the walls of the system to the ambient environment via $R_{leakage}$. The analogy of thermodynamics with electrondynamics allows to create the model in figure 1.3. Since this this model is to be used for a temperature controller, an assumption to simplify it can be made.

The controller will keep T_{system} constant and if the ambient temperature and $\dot{Q}_{load}$ is *reasonably stable*, it is easy to see, that a constant thermal flux must flow through R since it cannot pass through the thermal capacitance C. *Reasonably stable* means that it can be treated as constant with respect to the temperature controller time constants. This will be further discussed in section **??** with regards to system stability. If this assumption holds, the thermal flux from the system load will only cause a constant offset of T_{in}, since the heat must be removed by the controller, and the model can be simplified further:



Neglecting the constant thermal flux from the system load and exploiting the analogy of thermodynamics and electrondynamics again, using Kirchhoff's second law, we find:

$$\sum T_i = 0$$

$$T_{in}(t) - \dot{Q}(t)R - \frac{1}{C}\int \dot{Q}(t)\,dt = 0 \tag{1.13}$$

Taking the Laplace transform, applying equation 1.6 and using $T_{out} = \frac{1}{sC}\dot{Q}(s)$ to replace $\dot{Q}$, equation 1.13 can be written as:

$$T_{in}(s) - \dot{Q}(s)R - \frac{1}{sC}\dot{Q}(s) = 0$$

$$\dot{Q}(s) = \frac{T_{in}(s)}{R - \frac{1}{sC}} = \frac{T_{out}}{\frac{1}{sC}}$$

This allows to calculate the transfer function of the process P:

$$\begin{aligned} P(s) = \frac{T_{out}}{T_{in}} &= \frac{\frac{1}{sC}}{R - \frac{1}{sC}} \\ &= \frac{1}{sRC + 1} \\ &= \frac{K}{1 + s\tau} \end{aligned} \tag{1.14}$$

with the system gain K and the time constant τ. In case of the RC circuit, the gain is 1, but other systems may a gain or attenuation of $K \neq 1$ in the sensor.

Equation 1.14 is called the transfer function of a first-order model, because its origin is a differential equation of first order. This model describes homogeneous systems, like a room, very well, as can be seen in section **??**, but in order to derive the transfer function including the controller and the sensor some more work is required.

Expanding on figure 1.1b and equation 1.7 the closed-loop transfer function becomes:

$$G(s) = P(s) \cdot S(s) \tag{1.15}$$

and the block diagram becomes

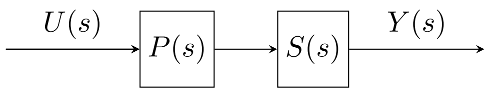


Figure 1.4: Open loop system with sensor.

The transfer funciton of the sensor can, in the most simple case, be modeled as a delay line with delay θ and $f(t-\theta) = H(t-\theta)$. Using equation 1.10 $S(s)$ can be written as

$$S(s) = e^{-\theta s}. \tag{1.16}$$

The full process model including the time delay is:

$$G(s) = \frac{K}{1 + s\tau} e^{-\theta s} \tag{1.17}$$

This is called a first-order plus dead-time model (FOPDT) or first-order plus time-delay model (FOPTD). To fit experimental data to this model it is more convenient to transform the transfer function 1.17 into the time domain. To calculate the output response an input $U(s)$ is required. In principal any function can do, but a step function is typically used, for example by Ziegler and Nichols [12] and many others [5, 6, 10, 9, 7, 8, 1]. It is both simple to calculate and apply to a real system. Using equations 1.10 and 1.12, the Heaviside $H(t)$ step function transforms as

$$\mathscr{L}\left(u(t)\right) = U(s) = \mathscr{L}\left(\Delta u H(t)\right) = \frac{\Delta u}{s} \tag{1.18}$$

with the step size Δu. The output $Y(s)$ can then be calculated analytically.

$$\begin{aligned} Y(s) &= \frac{\Delta u}{s}\frac{K}{1+s\tau}e^{-\theta s} \\ &= K\Delta u \frac{1}{s(1+s\tau)}e^{-\theta s} \\ &= K\Delta u \left(\frac{1}{s} - \frac{\tau}{s\tau+1}\right)e^{-\theta s} \\ &= K\Delta u \left(\frac{1}{s} - \frac{1}{s+\frac{1}{\tau}}\right)e^{-\theta s} \end{aligned} \tag{1.19}$$

To derive $y(t)$, the inverse Laplace transform of $Y(s)$ is required. Unfortunately, this is not as simple as the Laplace transform. Fortunately, using 1.12 while making sure causaulity is guaranteed as shown in 1.11, the simple first order model can easily be transformed back into the time domain.

$$\begin{aligned} \mathscr{L}^{-1}\left(Y(s)\right) = y(t) &= K\Delta u \mathscr{L}^{-1}\left(\frac{1}{s}e^{-\theta s}\right) - K\mathscr{L}^{-1}\left(\frac{1}{s+\frac{1}{\tau}}e^{-\theta s}\right) \\ &\overset{1.12}{=} K\Delta u \cdot 1 \cdot H(t-\theta) - \left(e^{-\frac{t-\theta}{\tau}}\right)H(t-\theta) \\ &= K\Delta u \left(1 - e^{-\frac{t-\theta}{\tau}}\right)H(t-\theta) \end{aligned} \tag{1.20}$$

The time domain solution of the FOPDT model can now be used extract the parameters τ, θ and K from a real physical system using a fit to the measurement data. The parameter Δu is already known, since it is an input parameter. A simulation of the step response of a first-order model with time delay is shown in figure 1.5. Here it can be clearly seen, that the output does not change until the time θ has passed and the Heaviside function changes from 0 to 1.

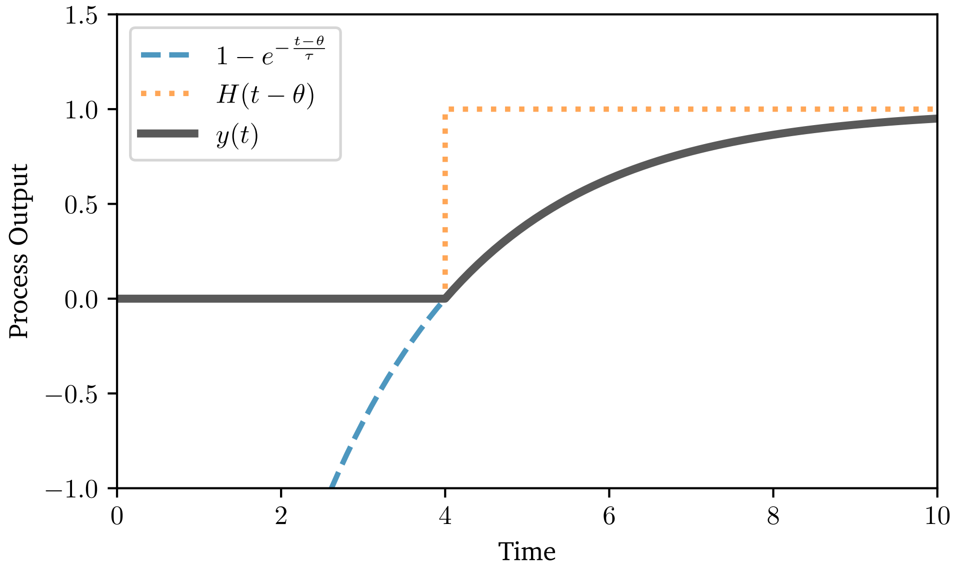


Figure 1.5: Time domain plot of a first-order plus dead time model, showing induvidual components of the model and the composite function $y(t)$. Model parameters: $K = \Delta u = 1$, $\tau = 2$, $\theta = 4$.

1.4.3 PID tuning rules

We use $\tau_c = \tau$ as suggested by [10, 9] for "*tightest possible subject to maintaining smooth control*".

1.5 Temperature Controller

1.5.1 Tuning of a PID controller

The number of emperical algorithms to determine a set of PID parameters (K_p, K_i, K_d) are numerous. In this work only the most common algorithms and a few notable exceptions will be presented.

1.5.2 Design

2 Outlook

Bibliography

[1] K.J. Åström and R.M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2010. ISBN: 9781400828739. URL: https://fbswiki.org/wiki/index.php/Feedback_Systems:_An_Introduction_for_Scientists_and_Engineers.

[2] Patrick Baus. *TinkerforgeAsync*. Version 1.2.0. July 2021. URL: %7Bhttps://github.com/PatrickBaus/TinkerforgeAsync%7D.

[3] Dan Kegel. *The C10K problem*. May 1999. URL: %7Bhttps://web.archive.org/web/19990508164301/http://www.kegel.com/c10k.html%7D.

[4] Paul J Leach, Michael Mealling, and Rich Salz. "A Universally Unique IDentifier (UUID) URN Namespace". In: *RFC* 4122 (2005), pp. 1–32. URL: %7Bhttps://datatracker.ietf.org/doc/html/rfc4122%7D.

[5] A.S. McCormack and K.R. Godfrey. "Rule-based autotuning based on frequency domain identification". In: *IEEE Transactions on Control Systems Technology* 6.1 (1998), pp. 43–61. DOI: 10.1109/87.654876.

[6] David W. Pessen. "A new look at PID-controller tuning". In: *Journal of dynamic systems, measurement, and control* 116.3 (1994), pp. 553–557.

[7] G.J. Silva, A. Datta, and S.P. Bhattacharyya. *PID Controllers for Time-Delay Systems*. Control Engineering. Birkhäuser Boston, 2007. ISBN: 9780817644239. URL: https://link.springer.com/book/10.1007/b138796.

[8] Guillermo J. Silva, Aniruddha Datta, and S.P. Bhattacharyya. "PI stabilization of first-order systems with time delay". In: *Automatica* 37.12 (2001), pp. 2025–2031. ISSN: 0005-1098. DOI: 10.1016/S0005-1098(01)00165-0. URL: https://www.sciencedirect.com/science/article/pii/S0005109801001650.

[9] Sigurd Skogestad. "Simple analytic rules for model reduction and PID controller tuning". In: *Journal of Process Control* 13.4 (2003), pp. 291–309. ISSN: 0959-1524. DOI: 10.1016/S0959-1524(02)00062-8. URL: https://www.sciencedirect.com/science/article/pii/S0959152402000628.

[10] R. Vilanova and A. Visioli, eds. *PID Control in the Third Millennium: Lessons Learned and New Approaches*. Advances in Industrial Control. Springer London, 2012. ISBN: 9781447124245. DOI: 10.1007/978-1-4471-2425-2_5.

[11] Richard Von Mises. *Über Aufteilungs-und Besetzungswahrscheinlichkeiten*. na, 1939.

[12] J. G. Ziegler and N. B. Nichols. "Optimum Settings for Automatic Controllers". In: *Journal of Dynamic Systems, Measurement, and Control* 115.2B (June 1993), pp. 220–222. ISSN: 0022-0434. DOI: 10.1115/1.2899060. eprint: https://asmedigitalcollection.asme.org/dynamicsystems/article-pdf/115/2B/220/5546571/220_1.pdf. URL: https://doi.org/10.1115/1.2899060.