Institute for Beam Physics and Technology
Institute for Pulsed Power and Microwave Technology

Master Thesis

Development of a Control System to Improve the Stability of the FLUTE Electron Gun

Marvin-Dennis Noll

Supervisors:	Prof. Dr.-Ing. John Jelonnek (IHM)
	Prof. Dr. Anke-Susanne Müller (IBPT)
Advisor:	Dr. Nigel Smale (IBPT)
Period:	15.11.2020 – 07.07.2021

Karlsruhe, 07.07.2021

Institute for Beam Physics and Technology
Hermann-von-Helmholtz-Platz 1
Campus North, Building 348
76344 Eggenstein-Leopoldshafen

Institute for Pulsed Power and Microwave Technology
Hermann-von-Helmholtz-Platz 1
Campus North, Building 421
76344 Eggenstein-Leopoldshafen

Declaration

I hereby declare that I wrote my master thesis on my own and that I have followed the regulations relating to good scientific practice of the Karlsruhe Institute of Technology (KIT) in its latest form. I did not use any unacknowledged sources or means, and I marked all references I used literally or by content.

Karlsruhe, 16.07.2021

Marvin-Dennis Noll

Abstract

The compact accelerator Ferninfrarot Linac- und Test-Experiment (FLUTE) is currently under commission at Karlsruhe Institute of Technology (KIT). Its main purposes are to serve as a technology platform for accelerator research and the generation of strong and ultra short terahertz (THz) pulses.

The Radio Frequency (RF) photo injector, also called the electron gun and the Linear Accelerator (LINAC) are powered by a klystron. It is fed by a pulse forming network, which is driven by a high voltage source connected to mains power. For stable energies of the generated THz pulses, the electron energies have to be stable. To ensure stable energies of the emitted electron bunches, several parameters of the gun, such as temperature and the RF power supply from the klystron, have to stay inside tight tolerance bands.

In this work, instead of passively optimizing the stability of system components, such as the water coolers or power supplies, an active approach with a closed feed-back loop is evaluated. By means of a control system, the amplitude of the low power RF input signal of the klystron is manipulated to mitigate the effects of noise and drifts on the electron energy. As there is currently no non-destructive sensor to measure the electron energies of all the electron bunches, the RF power in the first gun cavity is used instead as an estimator for the electron energy stability.

As part of the development process, first the stability issue is analyzed and metrics for quantifying the stability are defined. Then, an appropriate solution, a linear, discrete time control system, is proposed. In order to implement it, all the necessary building blocks of such a control system are treated in detail. First the necessary sensors and actuators are selected. Then the controller and the measurement filter are designed. To verify the designed system, first an offline simulation on a computer is performed which shows qualitatively a satisfactory disturbance rejection with a measured disturbance signal from FLUTE.

Then the control system is implemented as an algorithm with a fixed-interval control loop using the Python programming language. A graphical user interface, written in Qml, provides the user with plots and status information and allows the fine-tuning of the controller.

The following experiments at FLUTE show results in accordance to the simulation. That is, the stability, when defined as the relative standard deviation, is improved greatly by about a factor of 25.

Finally, ways to refine the control system are regarded. First, by using disturbance feed-forward of the change in waster temperature, the control system is made more robust and achieves the same results. Second, the usage of a Faraday cup, which measures total electron charge provides a potentially better representation of the electron energies, however as the electron beam is lost in the cup, its usages are limited.

Contents

List of Figures

List of Tables

1. Controller Design and Evaluation

In this chapter a control system is designed and evaluated to stabilize Ferninfrarot Linac- und Test-Experiments (FLUTEs) RF system. Referring back to the block diagram of a generic control system in **??**, there are three blocks to determine. First the plant transfer function, which describes the system that is to be controlled. Second, based on the plant, an appropriate controller type is chosen and its parameters are calculated. Third, a measurement filter is used to improve the quality of the feedback signal path. As the choice of the measurement filter influences the controller design, its design is treated before the controller.

1.1 Plant Identification

1.1.1 Principle

Before choosing an appropriate controller, some insight of the system response has to be obtained. Therefore in this section the plant's transfer function is estimated. In the context of this chapter "plant'' refers to everything from the attenuation set at the controllable attenuator to the system output, e.g. the cavity Radio Frequency (RF) power.

In the time domain, a Linear Time Invariant (LTI) system is described by its impulse response $h(t)$, that is the reaction of the plant to an impulse at the input. Using this definition directly, the plant's impulse response $p(t)$ could be measured by applying a short peak in the attenuation setting on the attenuator. The effect on the output is not easy to measure and a single measurement of this kind is very susceptible to noise. Therefore it is more common to measure the step response[Wan00], which is the output of a system, when a step function is applied to its input. As the step function is the time integral of the impulse function, the step response can be converted to the impulse response by differentiation in time.

Instead of measuring a single step response, often several step responses are measured and their average is computed to reduce the variance of the estimation. When measuring a step response the minimum needed measuring time depends on the systems time constants, but they are often not known beforehand.

That is why when there is no prior knowledge of the system, the identification is sometimes done with a Pseudo Random Binary Sequence (PRBS) to excite the system with step functions of different lengths. The PRBS is chosen in a way that some of the steps will probably last longer than a few dominant time constants of the system.

To get the transfer function $P(s) = \mathcal{L}\left\{p(t)\right\}$ of the plant from its step response(s), several methods are common, including correlation based and frequency response based algorithms.

1.1.2 Identifying the Plant Attenuator+RF

The input PRBS signal is generated with the Python script in Listing 1.1. Based on the value of a pseudo random number generator, the sequence toggles the attenuator between

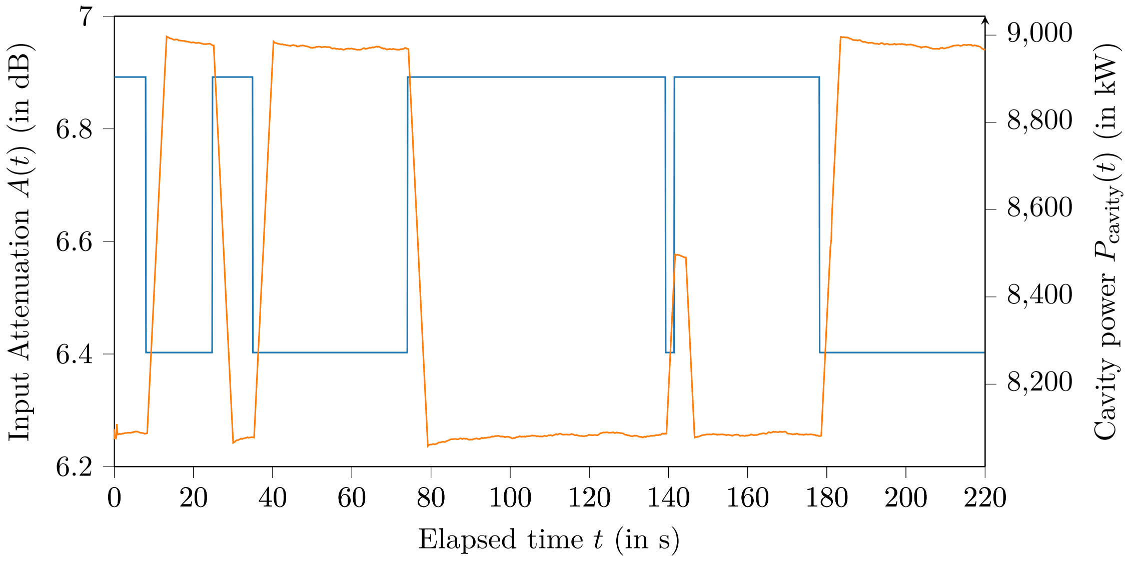


Figure 1.1: Section of the input sequence (blue) and the system response (orange); Note the inverse relation: A higher attenuation A causes a lower cavity power P_{cavity}

$V_{\text{control}} = 7\,\text{V}$ and $V_{\text{control}} = 11\,\text{V}$. Using **??**, this equals a span in attenuation of 6.892 dB to 6.4026 dB. With the parameter `toggleP`, the average length of one constant voltage level can be controlled.

Listing 1.1: Function to get a pseudo random binary sequence

```
def randomBinarySequence(N,toggleP):
    u=[False]*N
    for i in range(1,len(u)):
        if(np.random.binomial(1,toggleP,1)[0]):
            u[i]=not u[i-1]
        else:
            u[i]=u[i-1]
    return list(map(lambda x: 7 if x==False else 11,u))
```

In a test run over six hours (after all FLUTE subsystems had stabilized), the attenuator was driven with such a PRBS. The result is shown in Figure 1.1.

The time signals $A(t)$ and $P_{\text{cavity}}(t)$ are then split into an *estimation* data set (about 80 % of the samples) and a *validation* data set (the remaining $\approx 20\,\%$). This is done so the bulk of the available information is used to estimate the model, but there is data left that the model hasn't seen before. This smaller portion is used to validate the model's performance, hence it is called the validation data set.

The two data sets are then loaded into the MATLAB *System Identification Toolbox*. With the toolboxes pre-processing tools, first the means of both the input and output are removed, which is required by the estimators used. Then, using the "process model" estimator, linear, continuous time transfer functions with different numbers of zeros and poles are estimated (See screenshot of the Graphical User Interface (GUI) in Figure A.1). After that, to check the accuracy of the estimations, the System Identification Toolbox is used to simulate the output of the estimated systems (see Figure 1.2). For that the aforementioned validation data set is used. The measured output is compared with the outputs predicted by the models.

Using the Matlab function `zpk()` the models are converted into the zero-pole-gain representation. In Table 1.1 the estimated models are listed with their zeros, poles, gains and the model fit, as it is computed by the System Identification Toolbox. `P1`, `P2` and `P3` are

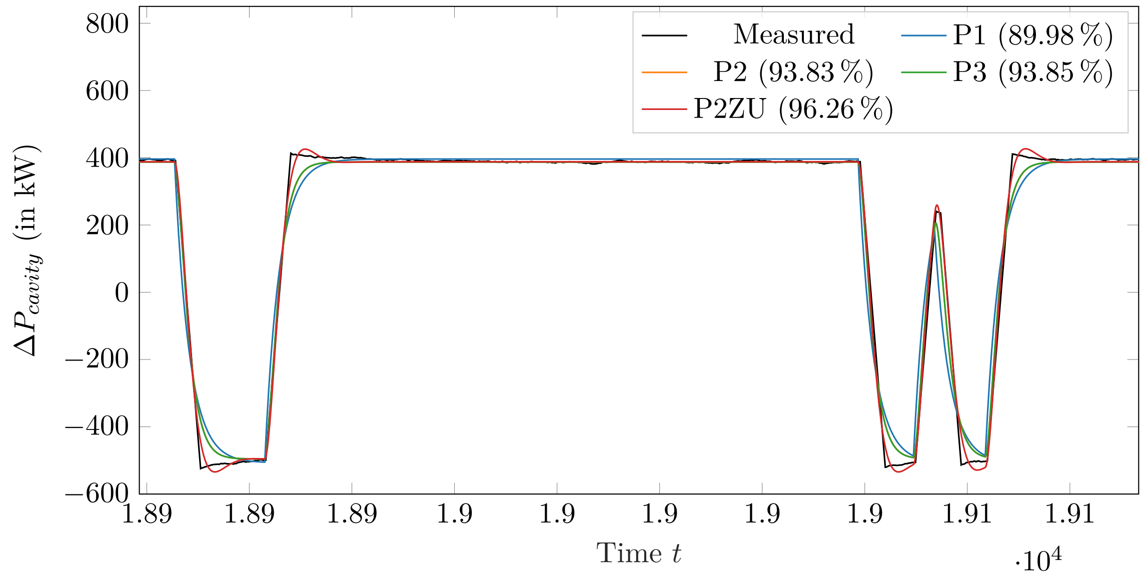


Figure 1.2: Validation of the estimated process models for the plant; the legend also shows the model fits in percent

Table 1.1: Process models of the plant as estimated by the Matlab System Identification Toolboxes process model estimator

Model name	Zeros	Poles	Gain	Model fit
P1		$(s + 0.3505)$	-646.94	89.98 %
P2		$(s + 0.6875)(s + 0.7039)$	-873.18	93.83 %
P3		$(s + 1 \times 10^6)(s + 0.7376)(s + 0.669)$	-8.9019×10^8	93.85 %
P2ZU	$(s - 2846)$	$(s^2 + 0.8014s + 0.3195)$	0.20296	96.26 %

models with one, two and three poles and a gain as free parameters. `P2ZU` consists of a complex pole pair, a zero and a gain.

Figure 1.2 and Table 1.1 show the `P2ZU` model to have the best fit. Therefore it is accepted as the plants transfer function. Using the Matlab function `tf()`, its time continuous transfer function can be stated as

$$P(s) = \frac{0.6352s - 1808}{3.13s^2 + 2.508s + 1}. \tag{1.1}$$

Using the Matlab function `c2d()`[1] and the sample time $T_s = 0.2\,\mathrm{s}$, $G(s)$ can be converted to the time discrete transfer function

$$P[z] = \frac{-10.91z^{-1} - 10.41z^{-2}}{1 - 1.84z^{-1} + 0.8519z^{-2}}. \tag{1.2}$$

The high model fit percentage of 96.26 % justifies the choice of a linear model with few parameters and further estimation attempts using other (non-) linear models are not needed. Considering the accuracy of the model, it is important to note this estimation is only valid at the time the measurements are taken. With FLUTE being a large experimental setup being still under commission, it is always possible that small changes to certain (sub-) systems can lead to minor or major influences to others. For the time being, the estimation is redone a day later and for both measurements days, different parts of the about six hour

[1]Without specifying a different method, `c2d()` discretizes the continuous-time model zero-order hold on the input.

measurements each are used as the estimation and validation data sets. Doing so shows no significant change in the estimated coefficients and the resulting model fits, indicating the estimated models are at least plausible.

To account for errors in the estimated model (and possible other errors), when designing the controller, sufficient gain and phase margins are set to ensure stable operation.

1.2 Measurement Filter

Like all measurements of physical quantities, the measuring of the system output of the control system is subjected to noise. In addition to these disturbances of thermal or electrical origins, also high frequency variations of the system output have detrimental effects on the control systems performance. For example the magnitudes of the bunch-by-bunch changes of the measured cavity power are often in the same order as the long-term drifts. Trying to correct for them, instead of the long term drifts, often leads to overcompensating and can even make the system unstable.

To remove the high frequency components, a low pass filter is used as the measurement block $H(s)$.

In pre-tests the incoming signal was simply filtered with a moving average filter. Commonly, the moving average is defined as the mean of a signal x inside a window of length L, centered around the current time or sample index n, that is shifted along the signal. This smooths out small variations thus the moving average acts as a low pass filter. This *non-causal* version of the moving average can only be used with already measured data as to compute the moving average at n, future values at $n+i$ are needed:

$$\mathrm{MA}_{x,\text{non-causal},L}[n] = \frac{1}{L} \sum_{i=n-\frac{L-1}{2}}^{n+\frac{L-1}{2}} x[i] \tag{1.3}$$

When filtering real-time data, future values are not available and a shifted, *causal* version, of the moving average

$$\mathrm{MA}_{x,\text{causal},L}[n] = \frac{1}{L} \sum_{i=n-(L-1)}^{n} x[i] \tag{1.4}$$

is used.

In case of the cavity RF power, experiments show a window length of about $L = 100$ or more is necessary to sufficiently smooth the measured power signal. When comparing the original signal with the filtered one, it is apparent that in addition to the desired smoothing effect, the filtered signal also is delayed in time, with the delay being dependent on the window size. To quantify the delay, the alternative definition of the moving average as a digital Finite Impulse Response (FIR) filter is used. One possibility to describe a FIR filter is by giving its impulse response, i.e. the output signal when the input of the filter is an impulse with unity height. In case of the moving average filter, the coefficient sequence of the corresponding FIR filter has the length $N = L$ and is equal to the the impulse response $h[n]$:

$$h[n] = \frac{1}{N}[\underbrace{1,\ 1,\ ...,\ 1}_{N}] \tag{1.5}$$

The delay introduced by a digital filter can be quantified with the filter's group delay

$$\frac{\tau_g(f)}{T_s} = \frac{\mathrm{d}\phi(f)}{\mathrm{d}f} \tag{1.6}$$

which is given normalized to the sampling time T_s [Kam02, p. 70]. In case of a FIR filter with linear phase (with a symmetrical impulse response), the group delay is always constant over all frequencies and is only dependent on the filter length N[Kam02, p. 165]:

$$\frac{\tau_g(f)}{T_s} = \frac{\mathrm{d}\phi(f)}{\mathrm{d}f} = \frac{\mathrm{d}\phi}{\mathrm{d}f} = \frac{N}{2} \tag{1.7}$$

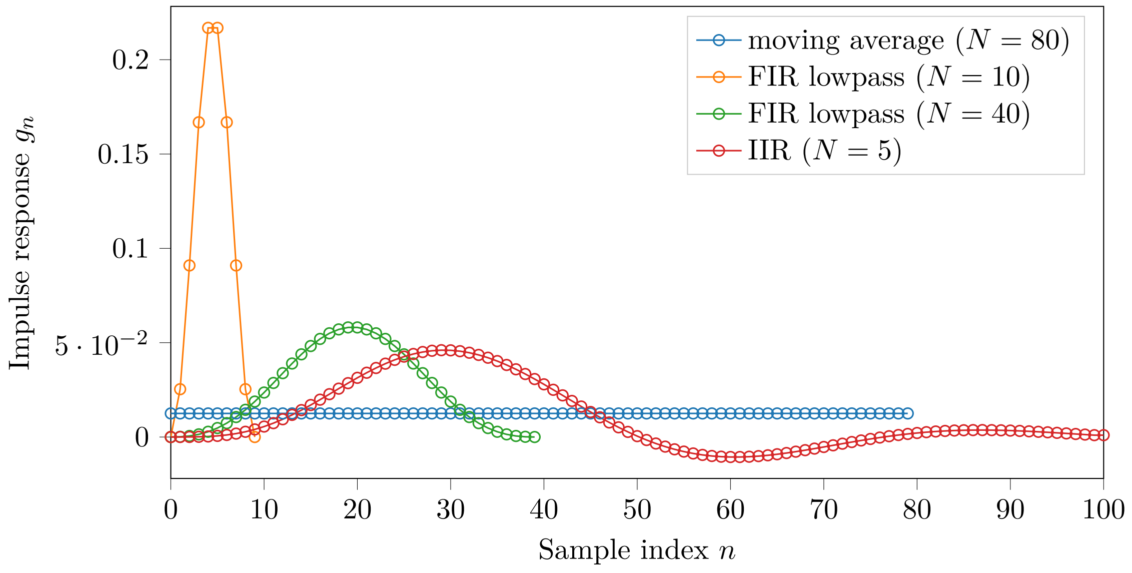


Figure 1.3: Impulse responses of a moving average filter ($N = 100$), a FIR lowpass ($N = 50$, $f_c = 0.1\,\mathrm{Hz}$) and a IIR Butterworth lowpass ($N = 50$, $f_c = 0.1\,\mathrm{Hz}$)

With a sampling time of $T_s = {}^1\!/_{5\,\mathrm{Hz}} = 200\,\mathrm{ms}$ and $N = 100$ the group delay is $10\,\mathrm{s}$. In case of a steady operation this is acceptable, as the disturbances to compensate happen on a timescale in the order of several minutes. But in case of ongoing transients due to user changes to the control system parameters or short error bursts on the measured signal, this long delay causes problems and therefore should be reduced.

Therefore a more sophisticated digital filter is designed to replace the simple moving average.

On the one hand, a FIR filter is designed with the Kaiser window method. This method starts with the desired frequency response, which is usually given piece-wise. In case of the low pass filter it is a step function at a cutoff frequency f_c. Then the IDFT is used to compute the corresponding impulse response $h_{\mathrm{IIR}}[n]$, which is in general infinitely long. Windowing with e.g. a Kaiser window and then truncating the impulse response yields the impulse response of the desired FIR filter $h_{\mathrm{FIR}}[n]$.[Opp10, p. 533]. With SciPy using `b=signal.firwin(N, fc,fs)`, the coefficients of a FIR with this method can be calculated.

On the other hand, an Infinite Impulse Response (IIR) filter is designed with the impulse invariance method and an analog Butterworth filter. This method could be interpreted as sampling the infinitely long analog impulse response.[Opp10, p. 497] In SciPy using `b,a=signal.butter(N,fc,'lowpass',fs,)`, the coefficients of an IIR can be calculated with this method. For an IIR filter, the group delay cannot be calculated with Equation 1.7 and it is in general frequency-dependent.

Figure 1.3 shows the impulse responses of the moving average, the FIR lowpass and the IIR lowpass (truncated to $N = 100$).

In Figure 1.4, the three filter types described above are compared by filtering a ten minute long segment of pre-recorded data. The filtering is done with the SciPy function `signal.lfilter()` which does causal filtering and does not compensate group delay[2], so the results are the same as they would be for real-time data.

[2]In contrast to `signal.filtfilt()`, which applies the filter both forward and backward achieving zero phase/group delay, but this cannot be done for incoming real-time data.

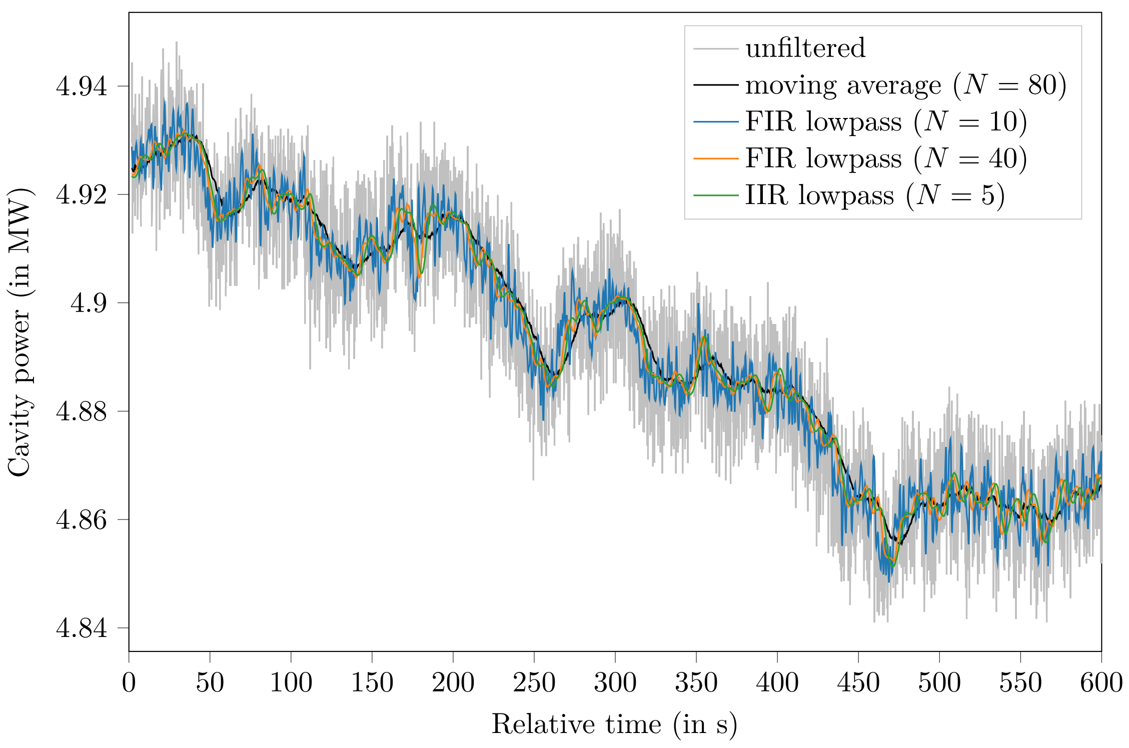


Figure 1.4: Effects of the three different lowpass filters in Figure 1.3 on noisy data

The plot shows the FIR lowpass filter requiring ten times the number of coefficients to achieve about the same result as the IIR lowpass filter. Also the moving average filter has double the number of coefficients as the FIR lowpass filter, but there is still high frequency noise in the output (caused by the sinc(·) shape of its frequency response $H[f] = \text{DFT}\{h[n]\}$).

Compared to the FIR lowpass, the moving average offers no benefit besides its easy implementation. When comparing the FIR with the IIR approach, the IIR has the advantage of needing less coefficients, thus occupying less memory, which is not really an advantage when the control system is implemented on a personal computer, which typically has enough free memory to hold millions of floating point numbers. Also the IIR filter has a non-constant group delay and is not guaranteed to be stable like all FIR filters are.

For these reasons, in the following a FIR lowpass filter is used.

One example filter generated with `signal.firwin()` with a cutoff frequency $f_c = 10\,\text{mHz}$ and order $N = 10$ has the transfer function

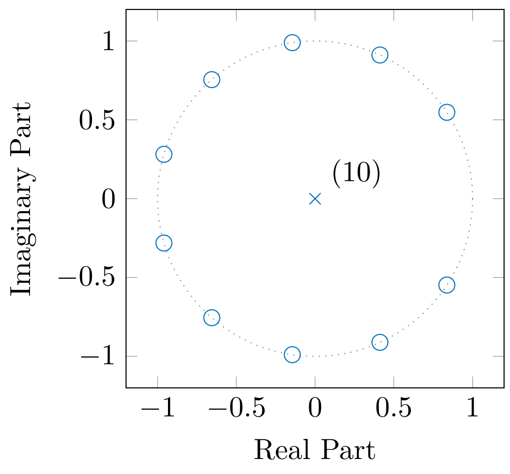
$$H[z] = \frac{1}{b_{10}z^{10} + b_9z^9 + b_8z^8 + b_7z^7 + b_6z^6 + b_5z^5 + b_4z^4 + b_3z^3 + b_2z^2 + b_1z + b_0} \tag{1.8}$$

with the coefficient vector $\vec{b} = [b_{10}, ..., b_0]$, with

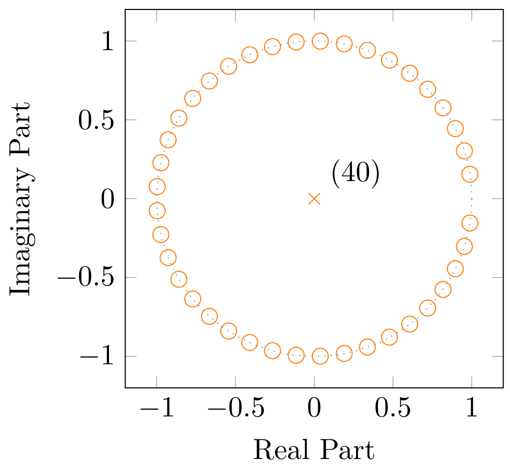
$$\vec{b} = [0.0876, 0.0896, 0.0911, 0.0922, 0.0929, 0.0931, 0.0929, 0.0922, 0.0911, 0.0896, 0.0876]. \tag{1.9}$$

Instead of stating the transfer function, a plot of the poles and zeros is a more intuitive representation. In Figure 1.5, the poles and zeros of the FIR filter with $N = 10$ are compared to one with $N = 40$. In addition, Figure 1.6 shows the magnitude responses of such filters.

For a fixed cutoff frequency and variable order N, the trade-off for choosing N is between the group delay introduced by the filter (see Equation 1.7) and the width of the transition



(a) $N = 10$



(b) $N = 40$

Figure 1.5: Pole-Zero maps for two FIR filters with a common cutoff frequency $f_c = 10\,\text{mHz}$ but different filter orders N;
$\circ$ denotes zeros, $\times$ denotes poles, (k) is a k-times pole

band or in other words the sharpness of the filter. This can be seen by comparing the cases $N = 10$ and $N = 40$ for the FIR filters in Figure 1.4: While the smoothing-effect of the longer filter is obviously better, it also causes a higher group delay, which results in the filter result of the $N = 40$ filter being shifted in time by $\frac{40}{2}T_s = 4\,\text{s}$ compared to $\frac{10}{2}T_s = 1\,\text{s}$ in case of $N = 10$. While a high smoothness is desired for the control system to reject high frequency noise, using a filter with a too high order can introduce a group delay high enough to shift the closed-loop from negative to positive feedback, thus making it unstable.

For the controller design in the next section and the later implemented real-time system, the cutoff frequency and the filter order are kept variable to leave room for improvement.

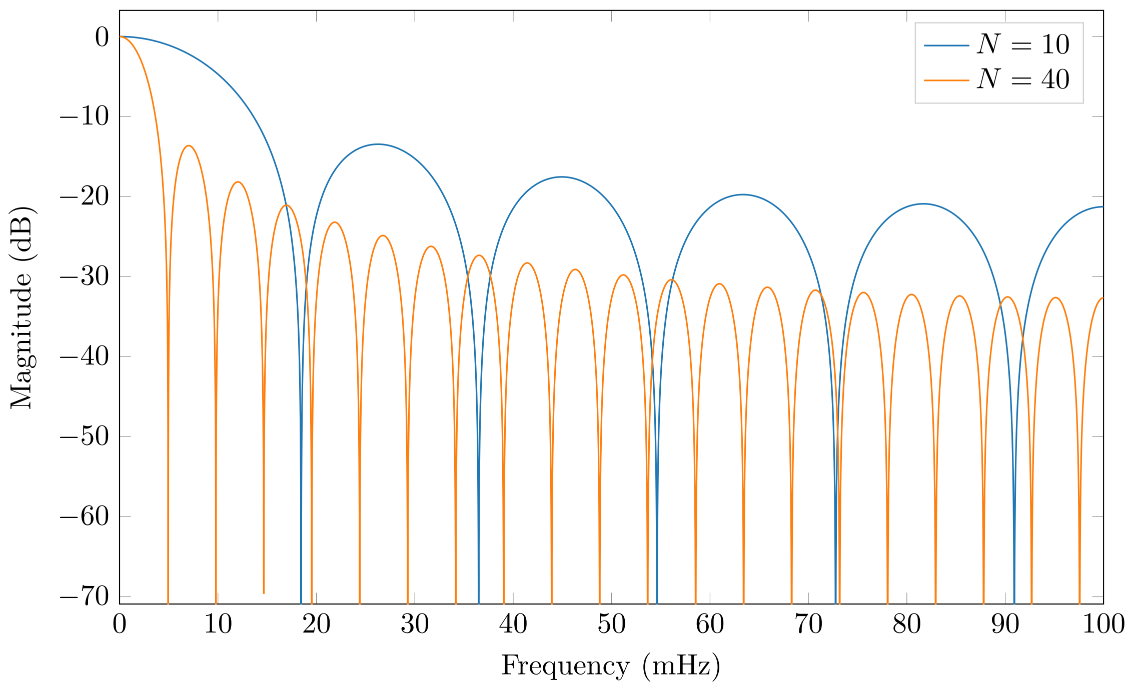


Figure 1.6: Magnitude response of two FIR filters with a common cutoff frequency $f_c = 10\,\mathrm{mHz}$ but different filter orders N

Real-time implementation of a FIR Filter in Python

Similar to Equation 1.4, a FIR filter designed with the SciPy function `sigal.firwin()` can be used in a causal manner to filter real-time data.

Applying the filter on pre-recorded data, like in Figure 1.4 can be done with `signal.lfilter()`. To use `signal.lfilter()` on sample-wise incoming real-time data, the “initial in” input and “final out” output of `signal.lfilter()` can be used to keep the filters state. This is demonstrated in Listing 1.2 by looping through pre-recorded data point-by-point.

Listing 1.2: Demonstration of the `zi` and `zf` variables when using `signal.lfilter()`

```
x=df2["F:RF:LLRF:01:GunCav1:Power:Out Value"].to_numpy()
y=np.array([])
zf=signal. lfilter_zi (b, 1)
for i in range(len(x)):
    y0,zf=signal. lfilter (b, 1, [x[i ]], zi=zf)
    y=np.append(y,y0[0])
```

1.3 Controller Design

Based on the estimated model of the plant $P(s)$ (or $P[z]$) in section 1.1 and the type of measurement filter designed in section 1.2, in this section an appropriate controller to stabilize the plant is designed and its performance is evaluated.

Tuning the controller and testing its capabilities is performed both *offline* with simulations using the building blocks $P[z]$, $H[z]$ and the yet to be defined $G[z]$ and *online* using a software implementation of the control system with the real hardware, i.e. FLUTE and the controllable attenuator.

1.3.1 Choosing a Controller Type

The plant has been identified using a linear model in section 1.1 resulting in the LTI system $P(s)$ (or $P[z]$). The matching controller does not necessarily have to be a LTI system, but choosing a LTI system simplifies design and analysis and is a justifiable choice here, as there are no good reasons against it, yet.

The class of LTI controllers is dominated by the Proportional Integral Derivative (PID) controller and its variants. PID stands for “proportional”, “integral” and “derivative”, which are LTI systems themselves, performing scaling, integration or differentiation. Depending on the application, variants, such as the PI controller, or the PD controller are used as well.

According to [ÅH95, p. 111] PID control is applicable for plants with order two or less. Compared to other types of controllers, using a PID controller has many advantages. It is is quick to design and does not depend on an accurate plant model. Also after the designing step, the few free parameters can easily be presented to an operator and online fine tuning is possible.[3]

In the next section such a PID controller is designed. As it is to be implemented in software later, the design is done in discrete time.

1.3.2 Designing a Discrete Time PID Controller

The output signal $u(t)$ of a generic time continuous PID controller (see Figure 1.7) consists of the weighted sum of three error signals.

[3]In contrast a compensating controller is based on an accurate model of the plant and changes on the fly are difficult. [ZR10]

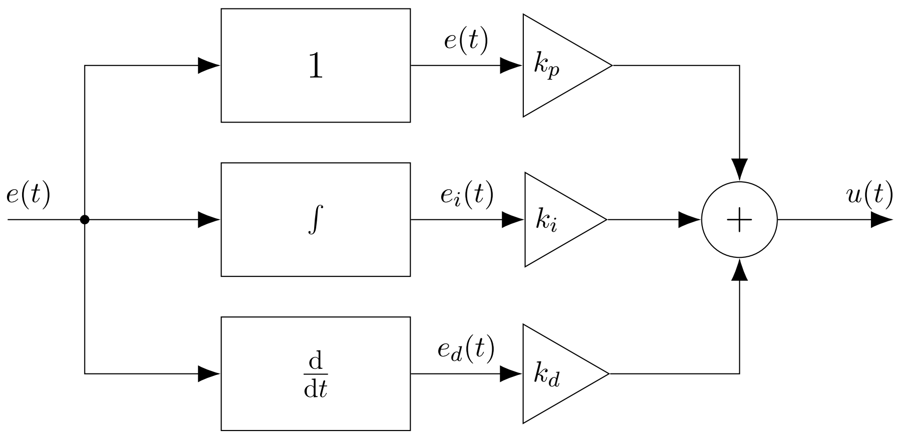


Figure 1.7: Block diagram of a generic PID controller

The first is simply the controller input error signal $e(t)$ scaled by the gain k_p. This is the proportional part of the PID controller. The integral part is calculated by computing the running time integral

$$e_i(t) = \int_0^t e(\tau)\mathrm{d}\tau. \tag{1.10}$$

This is then scaled by the constant k_i. To get the derivative part, the derivative

$$e_d(t) = \frac{\mathrm{d}}{\mathrm{d}t}e(t) \tag{1.11}$$

is calculated and weighted with k_d. All three are then summed to get $u(t)$ in the so called parallel form[Dod15, p. 5]:

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau)\mathrm{d}\tau + k_d \frac{\mathrm{d}}{\mathrm{d}t}e(t) \tag{1.12}$$

Often instead of the parallel form, the PID controller is stated in *standard form*. Instead of using the gains $k_{p,\,i,\,d}$, the parameters proportional gain K, integral time T_i and derivative time T_d are used.[ÅH95, p. 76] With the conversions

$$K = k_i \tag{1.13}$$
$$T_i = \frac{k_p}{k_i} = \frac{K}{k_i} \tag{1.14}$$
$$T_d = \frac{k_d}{k_p} = \frac{k_d}{K} \tag{1.15}$$

the PID controller in standard form is

$$u(t) = K\left[e(t) + \frac{1}{T_i}\int_0^t e(\tau)\mathrm{d}\tau + T_d\frac{\mathrm{d}}{\mathrm{d}t}e(t)\right]. \tag{1.16}$$

The transfer function $G(s)$ is given by the Laplace of $u(t)$ using the computation rules[Leó15]

$$\mathcal{L}\left\{\int_0^t y(\tau)\mathrm{d}\tau\right\} = \frac{1}{s}Y(s) \tag{1.17}$$
$$\mathcal{L}\left\{\frac{\mathrm{d}}{\mathrm{d}t}y(t)\right\} = sY(s) \tag{1.18}$$

as

$$G(s) = K\left[1 + \frac{1}{sT_i} + sT_d\right]. \tag{1.19}$$

To get the discrete transfer function, either the Laplace transform $G(s) = {}^{U(s)}/_{E(s)}$, is discretized or the $\mathcal{Z}$ transform of $u[n]$ is calculated. First, to get $u[n]$, the derivative in Equation 1.11 is approximated by

$$e_d[n] = \frac{e[n] - e[n-1]}{T_s}. \tag{1.20}$$

This assumes $e(t)$ is sampled with a sampling rate of $f_s = {}^1/_{T_s}$ to get $e[n]$. In a similar fashion, the integral in Equation 1.10 is approximated by

$$e_i[n] = e_i[n-1] + e[n]T_s. \tag{1.21}$$

Using the shift rule of the $\mathcal{Z}$ transform[Leó15]

$$y[n-k] \circ\!\!-\!\!\!-\!\!\bullet\, z^{-k}Y[z], \tag{1.22}$$

the discrete transfer function of the PID controller is

$$G[z] = K\left[1 + \frac{T_s}{T_i}\frac{z}{z-1} + \frac{T_d}{T_s}\frac{z-1}{z}\right]. \tag{1.23}$$

In the time domain, this becomes

$$u[n] = k_p e[n] + k_i \left(\underbrace{e_i[n-1] + T_s e[n]}_{e_i[n]}\right) + k_d \left(\underbrace{\frac{1}{T_s}e[n] - e[n-1]}_{e_d[n]}\right) \tag{1.24}$$

1.3.2.1 Controller Tuning

Next the three free parameters $k_{p,\,i,\,d}$ or K, $T_{i,\,d}$ are to be chosen in such a way that the controller has optimal performance. This process is called *tuning*. Tuning of the parameters can be performed online or offline.

Online tuning is done by using the physical[4] plant. The most popular member of this class is the Ziegler-Nichols tuning[ZN42]. The method relies on experiments and tabulated values. The mostly used variant uses one measured step response.

Using this method at FLUTE produced mixed results. While being a very simple and fast process, the resulting controller is often unstable. This is partially due to errors when extracting the tabulated values from the noisy step response, but also the method intrinsically leads to poor stability margins.[ÅH95, p. 142] Nonetheless the Ziegler-Nichols method yields a usable starting point, if the strategy is to fine tune the controller manually by intuition and experience of the user.

As the Ziegler-Nichols method does not yield an acceptable parameter set and the method combined with fine tuning by hand takes a considerable amount of time, next tuning the parameters offline using only the plants transfer function and the measurement filter is done. The offline tuning can be done analytically or using different numerical optimization strategies. In [DHB19] analytical methods, such as the internal model control design or the pole placement design are discussed. Both require the system's transfer function in a closed form.
Tuning by numerical optimization chooses the parameters by a simulation or measured data. Goal of the optimization is to minimize a cost function J like

$$J(\theta) = \sum_{n=0}^{\infty} e_\theta[n]^2 \tag{1.25}$$

with the parameter vector $\theta = [k_p,\, k_i,\, k_d]$ or $\theta = [K,\, T_i,\, T_d]$:

$$\theta_{\text{opt}} = \underset{\theta}{\text{argmin}} J(\theta) \tag{1.26}$$

A convenient way to do such an optimization by simulation using the transfer function of the estimated plant is the Matlab *PID Tuner*.

Tuning the Controller with the Matlab PID Tuner

The Matlab PID Tuner accepts different kinds of system models used by the Matlab/Simulink ecosystem. It is possible to directly use estimated models, models defined via

[4]"Physical" in the sense that one could touch it. (But should one?)

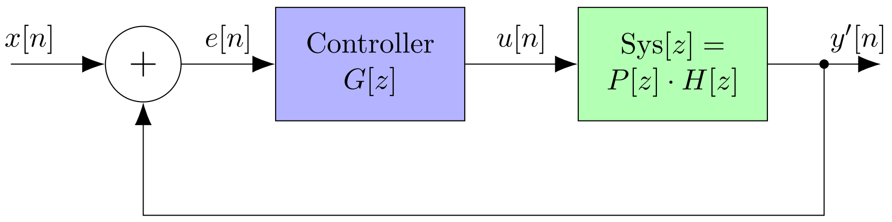


Figure 1.8: Required system architecture for the Matlab PID Tuner; PID Tuner input is the system block $Sys[z]$ (green), generated output is the controller $G[z]$ (blue)

their transfer function or the zero-pole-gain representation, Simulink models or combinations of those[5].

The manual of the PID Tuner[The21a] describes the expected input (see Figure 1.8). The feedback path has to have unity gain, i.e. there is no measurement filter allowed. Therefore the measurement filter is moved before the junction where the output $y[n]$ would normally be measured. Then the measurement filter $H[z]$ is combined with the plant $P[z]$ to form $Sys[z]$. For that reason $y[n]$ becomes an internal signal of $Sys[z]$. However the new system output $y'[n] = h[z] * y[n]$ is not too different from the old one, since $H[z]$ is designed to remove high frequency noise but retain the rest of $y[n]$.

The manufacture's documentation [The21b] does not disclose any internals of the PID Tuner nor state which optimization technique is used, but three tuning objectives are stated:

- Stability: The closed loop should be stable (that is BIBO stable as defined in **??**)
- Performance: The closed loop system tracks the input well and rejects disturbances as rapidly as possible (see **??**)
- Robustness: A gain and phase margin accounts for errors in the system model (see section 1.1)

Using Listing 1.3, the estimated plant $P[z]$ is loaded, a measurement filter $H[z]$ is generated and the combination $Sys[z] = P[z]H[z]$ is fed into the PID Tuner.

Listing 1.3: Matlab script to generate an input system for PID Tuner

```

% design lowpass measurement filter H
N=10;
lpFilt1 = designfilt('lowpassfir', 'FilterOrder', N, 'CutoffFrequency', ...
                     0.01, 'SampleRate', 0.2, 'DesignMethod', ...
                     'window', 'Window', 'kaiser');

% convert filter to a dynamic system
[z1,p1,k1]=zpk(lpFilt1);
H1=zpk(z1,p1,k1,0.2);

% load estimated transfer function and convert it to discrete form
load('P2ZU.mat')
P=c2d(idtf(P2ZU),0.2);

Sys1=tf1*H1; %Combine plant P and measurement filter H
pidTuner(Sys1); %Launch pidTuner
```

Using the PID Tuner GUI (see Figure A.2), the controller is designed by changing the design parameters `Response Time` (RT) and `Transient Behavior` (TB). Changing the

[5]Series connection can be established by multiplying the models transfer functions.

Table 1.2: Parameters of a discrete time PID controller in parallel form calculated with the Matlab PID Tuner; N is the order of the used measurement filter

Name	N	RT (in s)	TB	k_p	k_i	k_d
$G_1[z]$	10	5	0.2	$-0.000\,577$	$-0.000\,197$	$-0.000\,421$
$G_2[z]$	10	10	0.2	$-0.000\,155$	$-0.000\,111$	-5.43×10^{-5}
$G_3[z]$	40	5	0.2	$-0.000\,559$	-2.27×10^{-5}	$-0.000\,396$
$G_4[z]$	40	10	0.2	$-0.000\,35$	-9.68×10^{-5}	$-0.000\,316$

response time (in seconds) influences how fast the controller acts on changes. With the transient behavior setting, the allowed over- and under-shoots, that is, the deviation above and below the final value for $t \to \infty$ can be set qualitatively.

With this tool four different controllers are designed, two for each measurement filter's order of $N = 10$ and $N = 40$. The transient behavior with 0.2 for all controllers is chosen as a compromise between speed and overshoot behavior. The optimized PID parameters (for a discrete time PID controller in parallel form) are listed in Table 1.2.

1.3.3 Analyzing the Input Tracking and Disturbance Rejection

The PID Tuner defaults to show the input tracking step response (as in Figure A.2). The input tracking is calculated based on **??**. To calculate it, the Matlab function `step()` is used, which generates the step response plot of a dynamic system, in this case F_T. As the whole system only consists of linear building blocks, using an arbitrary step height of 1 is possible. The response is then normalized in such a way that the final value is also 1.

In Figure 1.9 and Figure 1.10 the input tracking step responses for all four controllers are plotted. Note the different t scales. As expected, for longer measurement filters, i.e. filters of higher orders, the controller has to act less aggressive. This causes longer settling times. Especially Figure 1.10 shows that setting a shorter response time does not automatically cause the controller to set the output faster to its final value. While for the shorter response time, the set value is reached quicker, there is also a strong oscillation, so choosing the longer response time (see $G_4[z]$) can be beneficial.

As the set-point for FLUTE only changes occasionally, basically a fixed set-point controller is needed. Therefore the disturbance rejection is more important than the input tracking when analyzing the controllers performance. The step responses of the disturbance rejection F_{DR} are calculated in a similar way as the input tracking, using **??** and `step()`. The y axes are normalized so that the initial value is 1 and the final value for $t \to \infty$ is 0.

In Figure 1.11 and Figure 1.12 the results for all controllers in Table 1.2 are plotted.

This shows that even for an optimistically low measurement filter order of $N = 10$, the settling time is in the order of slightly under a minute. This is sufficient for the application, but has more of a practical drawback: when fine-tuning the PID parameters manually on the machine, it is necessary to wait a few minutes after setting new parameters before assessing the change.

Especially when looking a the plots of $G_3[z]$, it is obvious that there can be stability issues when using a too aggressive controller. For that reason in the next section the stability of the four controllers is analyzed.

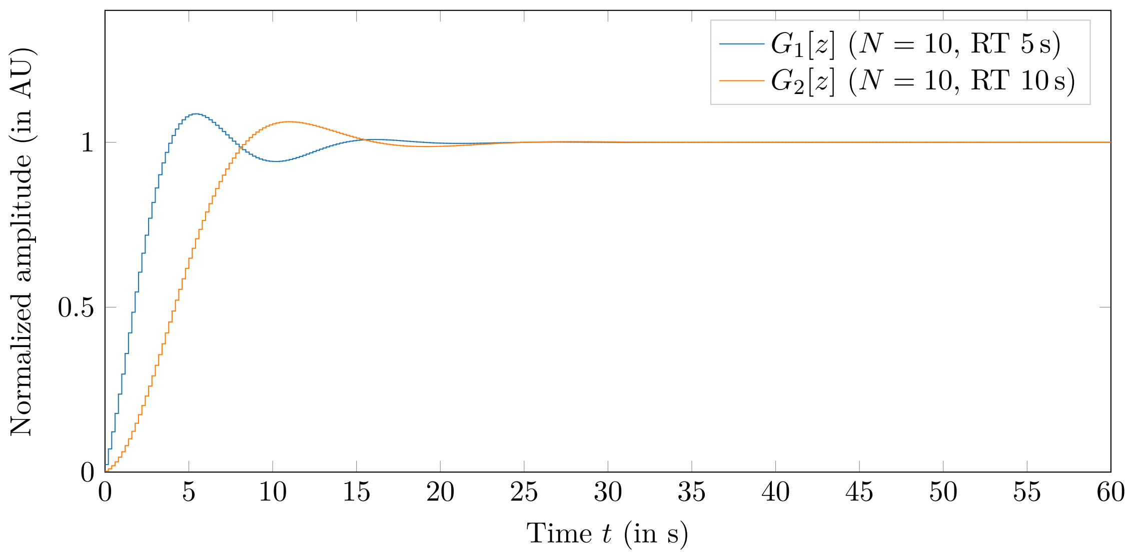


Figure 1.9: Step response of the input tracking F_T for controller $G_1[z]$ and $G_2[z]$, designed with the plant $P[z]$ and a measurement filter of order $N = 10$, response time goal to 5 s and 10 s

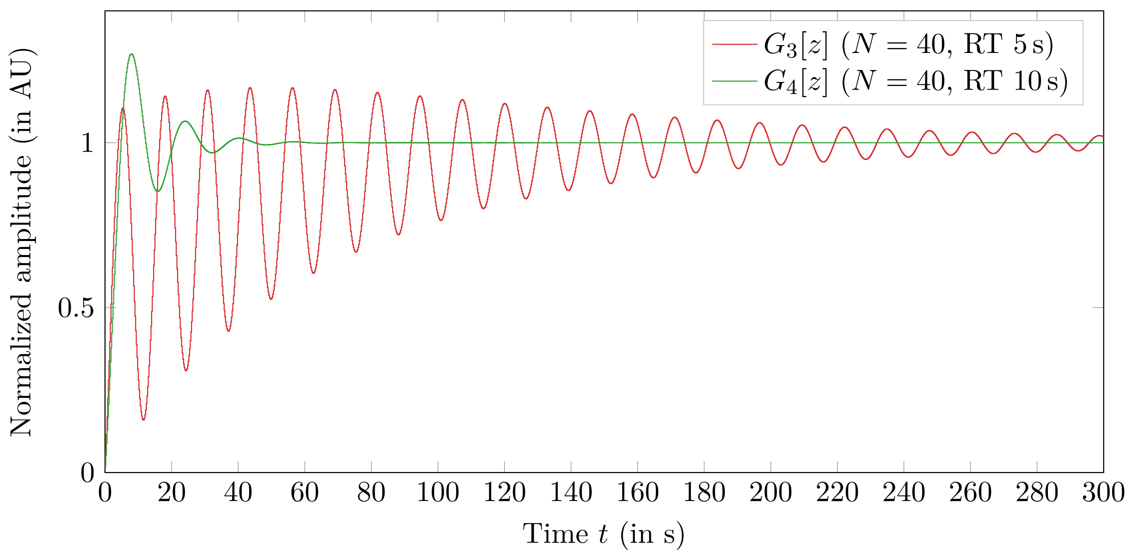


Figure 1.10: Step response of the input tracking F_T for controller $G_3[z]$ and $G_4[z]$, designed with the plant $P[z]$ and a measurement filter of order $N = 40$, response time goal to 5 s and 10 s

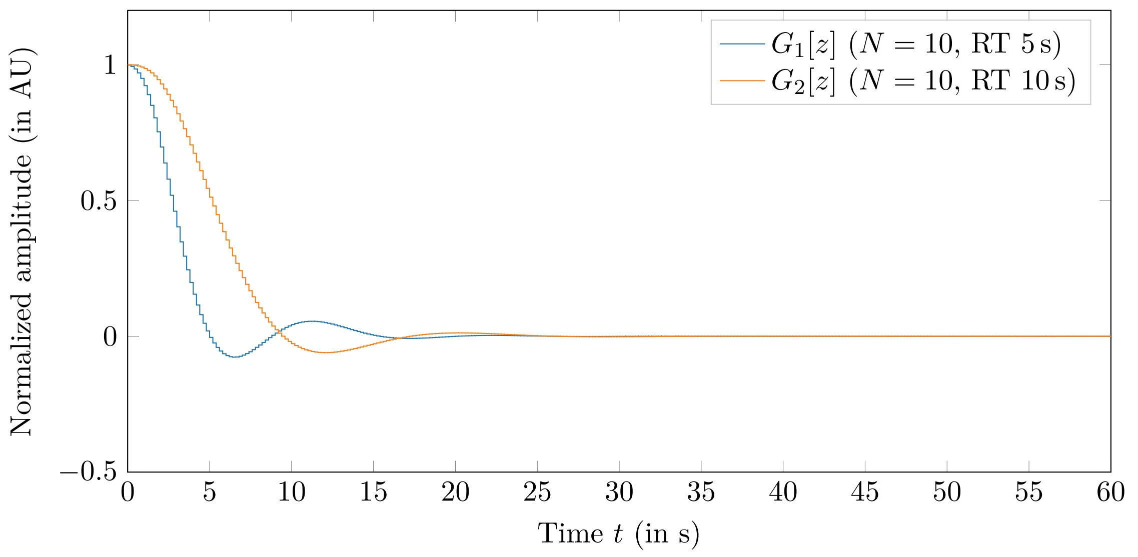


Figure 1.11: Step response of the disturbance rejection F_{DR} for controller $G_1[z]$ and $G_2[z]$, designed with the plant $P[z]$ and a measurement filter of order $N = 10$, response time goal to 5 s and 10 s

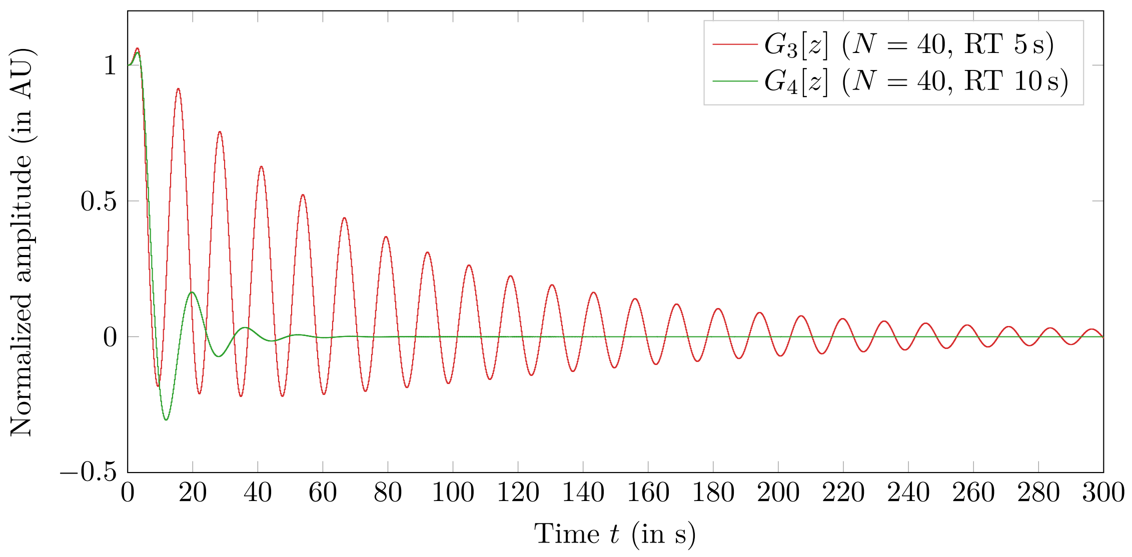


Figure 1.12: Step response of the disturbance rejection F_{DR} for controller $G_3[z]$ and $G_4[z]$, designed with the plant $P[z]$ and a measurement filter of order $N = 40$, response time goal to 5 s and 10 s

1.3.4 Analyzing the Controller Stability

In this section the stability of the controllers $G_i[z]$ from the last section is evaluated.

For that the Nyquist criterion according to **??** is used. With the Matlab function `nyquist()`, the locus $w = F_o[w = j2\pi f]$ of the open loops

$$F_o[z] = G[z]P[z]H[z] \tag{1.27}$$

are calculated. For all controllers designed, they are plotted in Figure 1.13.

This shows the gain margin for $G_3[z]$ (in combination with the $N = 40$ filter), that is the distance to the critical point $w = (-1, 0j)$, is already very small. So in addition to its oscillatory behavior and the longer settling time, there is also the risk that the system can become unstable if there are even small system parameter changes.

In Figure 1.14 another issue is highlighted. If a controller is designed with a certain measurement filter order $N = N_0$ in mind, but is used together with a filter of order $N = N_1 > N_0$, it is very likely the closed loop system becomes unstable. In the figure the controller $G_1[z]$ is used together with the $N = 40$ filter. Both the disturbance rejection and the Nyquist plot show the closed loop to be unstable.

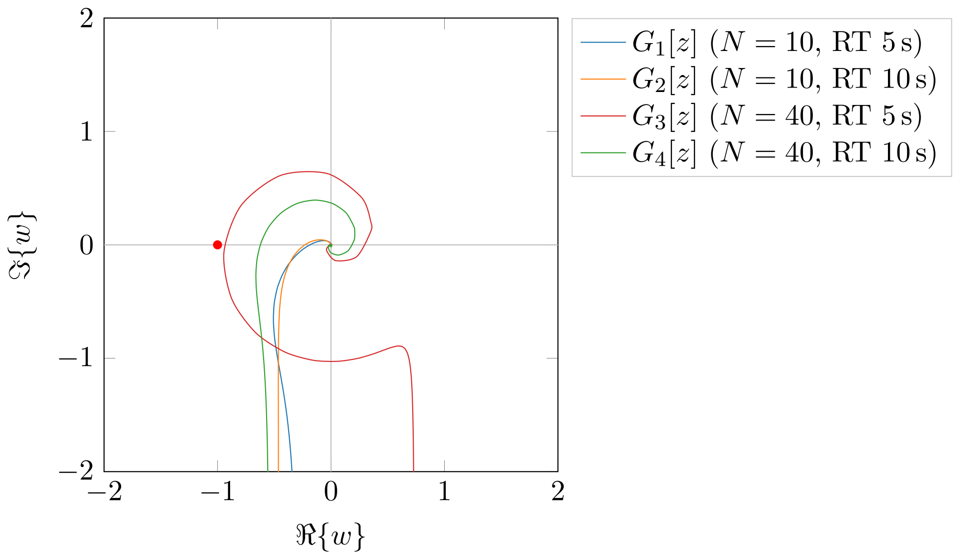
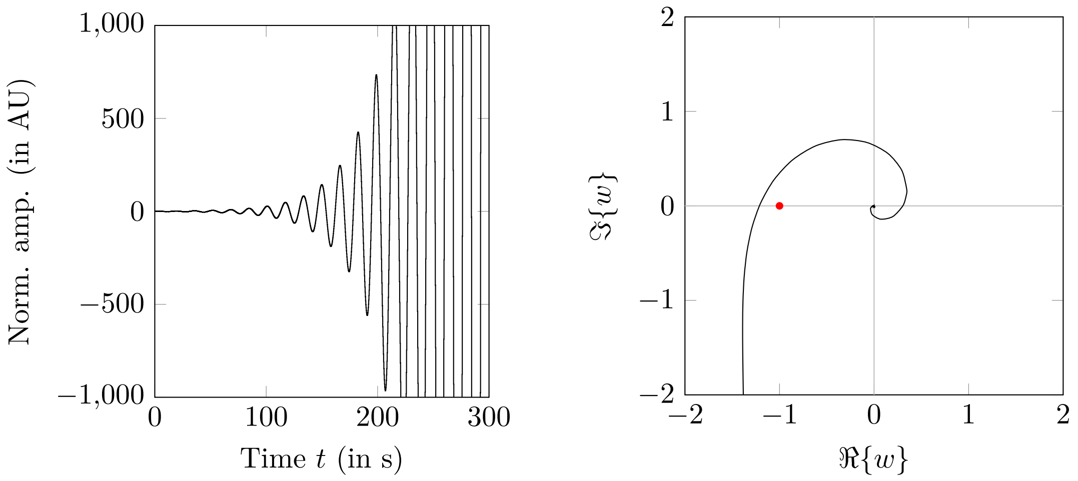


Figure 1.13: Nyquist plot to analyze the stability of the closed-loop control system based on the locus plot of the open-loop system; Note the critical point $(-1, 0j)$ is always on "left'' of the curve indicating stability



(a) Step response of the disturbance rejection; amplitude of oscillations grow over time

(b) Nyquist plot; critical point $(-1, 0j)$ is encircled

Figure 1.14: Two signs of an unstable closed loop system, in this case caused by a measurement filter with too high order and/or a too aggressive controller

1.3.5 Offline Evaluation with Measured Data

Now the performance of the controller $G_1[z]$ together with the filter order $N = 10$ should be evaluated with measured data before testing it with FLUTE. This simulation is called offline evaluation.

Offline evaluation of the control system is potentially less accurate due to an incomplete or erroneous model. But on the upside it requires no access to FLUTE and no potential downtime of the machine. Also with modern computers the simulation time advances much faster than real-time. Another big advantage is repeatability. With a pre-recorded data-set as the disturbance input, simulations are consistent and do not depend on a changing environment or system parameters.

The offline evaluation is done with Simulink, a block diagram development and simulation environment based on Matlab. The Simulink model (see Figure A.3) uses the disturbance

$$d[n] = \Delta P_{\text{cavity}}[n] = P_{\text{cavity}}[n] - \mu_{P_{\text{cavity}}} \tag{1.28}$$

with $\mu_{P_{\text{cavity}}}$ being the time average of $P_{\text{cavity}}[n]$ over one hour. $P_{\text{cavity}}[n]$ is re-sampled to $T_s = 0.2\,\text{s}$. With this calculation of $d[n]$, it is assumed that the disturbance is the deviation of the cavity power from a (theoretical) set-point of $\mu_{P_{\text{cavity}}}$.

With the simulation the effect of adding $d[n]$ to a system without any feedback and a system with the controller $G_1[z]$ and the measurement filter is compared. The result is shown in Figure 1.15

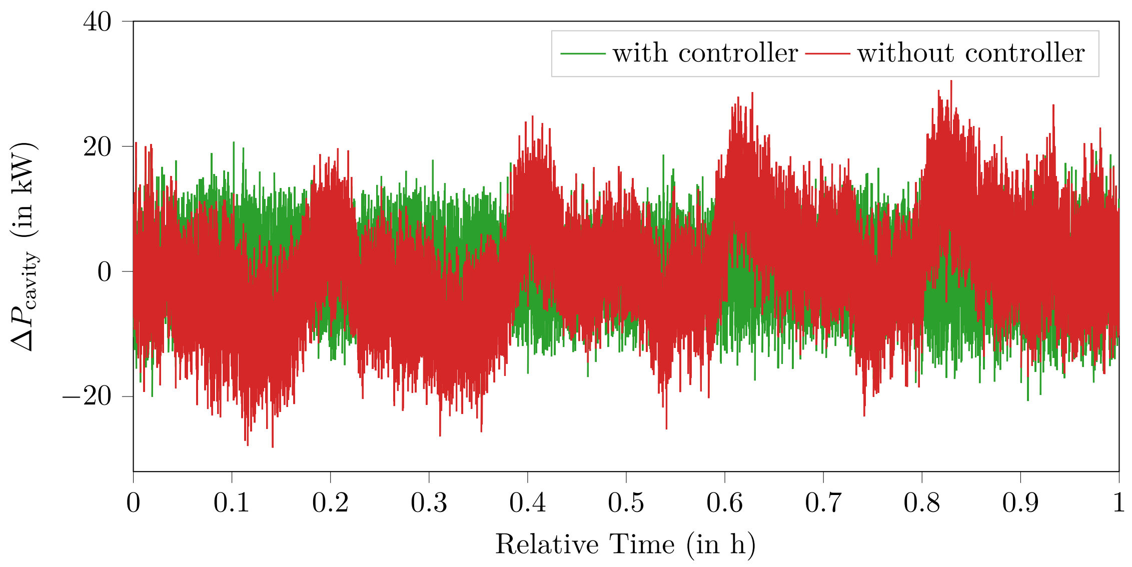


Figure 1.15: Output of the Simulink model in Figure A.3; step size $T = 0.2\,\text{s}$, end time $3600\,\text{s}$. Simulation time on a computer equipped with an Intel i7-3770: $3.5\,\text{s}$

1.4 Implementation of the Control System in Software

In this section the findings from the beginning of this chapter are used to implement a time discrete control system, that is a time discrete PID controller that is used to drive the controllable attenuator based on filtered measurements from the system.

The system should be implemented as software that runs on most personal computers and requires few external dependencies to make it as portable as possible. The other core requirements can be summarized to be:

- Communicate with Experimental Physics and Industrial Control System (EPICS) (over an Ethernet connection) to read in data such as $P_{\text{cavity}}[n]$
- Provide means to filter the incoming data with a measurement filter $H[z]$ with variable order N and cutoff frequency f_c
- Calculate the control error $e[n]$ and based on that the controller output $u'[n]$. Then convert the controller output, an attenuation, to the matching control voltage V_{control}
- Communicate with the Keysight 34972A over VME eXtensions for Instrumentation protocol specification 11 (VXI-11) (over an Ethernet Network) to set the control voltage $V_{\text{control}}[n]$ of the attenuator
- The control routine of the program needs to be light-weight enough so a scheduler can call it faster than five times a second to achieve a sampling time of $0.2\,\text{s}$
- Show the input, the output and the error signals to the user on a GUI
- Provide graphical input elements to let the user modify the measurement filter and the controller parameters (k_p, k_i, k_d)
- Log the input, output and parameters to disk for later reference

Since many other choices depend on it, first the programming language has to be picked. As Python was used in earlier chapters and the communicating abilities to both EPICS and the Keysight 34972A were already proven, it is the obvious choice.

From there on the GUI framework is the next decision to be made. For Python popular choices are *Tkinter*, a built in implementation of the Tk/tcl GUI toolkit, *wxPython*, a

Python binding to the cross-platform GUI library wxWidgets, or *PyQt*[Com21], a set of Python bindings for the Qt GUI framework[The21c]. While Tkinter has the advantage of being built into the Python language, wxPython offers more functionality and is more widely adopted. PyQt profits from the large Qt ecosystem, so is it for example possible to develop the GUI separately from the code using *Qt Designer*.

For (live-) plotting of data in the GUI, the standard library in Python is *matplotlib*[Hun07]. It can be used with all three plotting libraries. However the biggest drawback is the possible update speed of the plots. With more than about a hundred points on the screen, the update frequency drops to well below 1 Hz. A much faster alternative is *pyqtgraph*[Cam11], a scientific plotting library written in Python and using the Qt GraphicsView. It is only compatible only with PyQt (or the PySide alternatives).

Therefore PyQt and pyqtgraph are used to build handle the GUI and plot the live data. Also Qt Designer is used to model the GUI graphically.

The control algorithm is implemented by directly using the equations derived in subsection 1.3.2. The time domain representation of the PID controller is (restated from Equation 1.24)

$$u[n] = k_p e[n] + k_i \left(\underbrace{e_i[n-1] + T_s e[n]}_{e_i[n]} \right) + k_d \left(\underbrace{\frac{1}{T_s} e[n] - e[n-1]}_{e_d[n]} \right) \tag{1.29}$$

When translating Equation 1.29 to code, the recursion ($e_i[n]$ depending on $e_i[n-1]$) is solved by introducing a variable that keeps track of $e_i[n]$ over time. The last value of $e[n]$, $e[n-1]$ is kept in memory for the derivative part. An example in pseudo code is listed in Listing 1.4.

Listing 1.4: PID controller implemented in pseudo code

```
while(true) {
    e        := x_set − x_actual
    e_i      := e_i_old + e*T_s
    e_d      := (e−e_old)/T_s
    e_i_old  := e_i
    e_old    := e
    wait_sec(0.2)
}
```

To get the timing right and to allow the GUI to be responsive while the control algorithm runs in the background, a scheduler that activates a callback function every 0.2 s should be used. Since in **??** the Advanced Python Scheduler (AP scheduler) is used successfully, it should also be used for the control system. Unfortunately the AP scheduler is not compatible with PyQt and using it interferes with the internal Qt timing systems. For this reason, a timer is constructed instead with the `QTimer` class.[The21d]

Logging the data to disk is done by using the same `QTimer` as is used for the control algorithm, to write one line of CSV data to disk each time the timer is triggered.

With these building blocks ready, the GUI is build with Qt Designer and based on the `.ui` file generated from Qt Designer the GUI Application is implemented. For a screenshot of the program running see Figure A.4.

1.5 Evaluation of the Control System (Online)

After evaluating the controller offline, it is now time to evaluate the performance of the control system on the machine. To do so, FLUTE is operated with and without the control system switched on for 6 h each. Before the test, FLUTE is allowed to run a few hours for all components to reach operating temperatures. The result of the test is shown in Figure 1.16. Note, that in this test about 7 h into the experiment there was an unexpected shutdown of FLUTE and the corresponding block of data is removed before further processing.

The time plot Figure 1.16 and also the spectogram in Figure 1.17 show the cavity RF power approximately reaches stationarity in the [0, 6] hour interval respectively in the [6, 12] hour interval. This allows the spectrum for these two blocks to be estimated using a periodogram method. In Figure 1.19 the spectra for each case are shown. The corresponding time data is plotted in Figure 1.18

With the periodogram data, the control system success can be measured with the most prominent noise metric.

Next, the cases controller on and controller off should be compared with the relative standard deviation. Measuring and displaying the relative standard deviation $STD\%$ is also supported on the Low Level RF (LLRF) Control System Studio (CSS) panel at FLUTE, so while the experiments run, the values are checked from time to time. It can be observed that different values are shown over time although the plotted time signal looks stationary by eye. This suggests that the relative standard deviation does not only depend on the window size T over which it is calculated, but also heavily on the absolute position in time t. This effect is more visible for small window sizes. The issue is illustrated in Figure 1.20, where $STD\% = STD\%(t, T)$ is plotted as a function of the time t and the window size T.

For that reason, for small window sizes, time averages of $STD\%$ should be used instead of single values. For the measured $STD\%(t, T)$ for the controller off and the controller on cases, $STD\%(T)$ is plotted in Figure 1.21. This plot shows what could already be guessed from Figure 1.20: with the $STD\%$ metric, the system seems *less* stable with the controller on, if the $STD\%$ is calculated over small window sizes e.g. $T = 1\,\text{min}$. For very long windows, the dependency on the window position becomes very small. So for the comparison with the other metrics, $STD\%(T = 4\,\text{h})$ is used.

With the metrics from **??**, the success of the control system is assessed in Table 1.3.

This shows for example the mean squared error is improved by a factor of about 371 by using the control system.

Table 1.3: Quantitative assessment of the controllers performance

Metric	Controller off	Controller on	Controller off/Controller on
$\%STD(4\,\text{h})$	0.115 59	0.005 992 25	19.2891
MSE	37.639	0.101 33	371.44
MPN	487 309.29	14 386.25	33.873

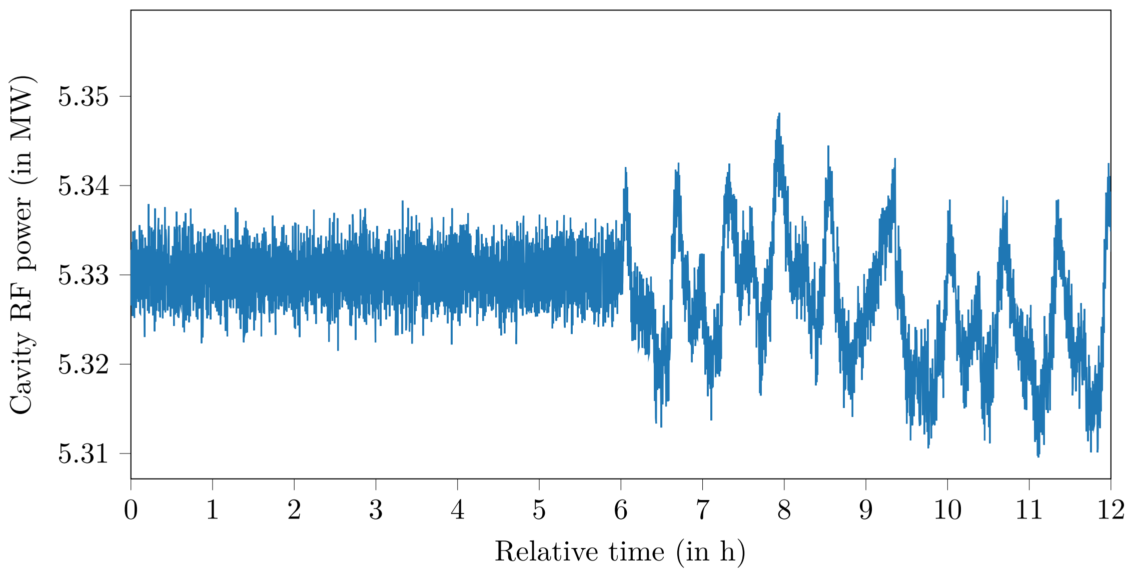


Figure 1.16: Cavity power over about 15 h (about three hours of downtime removed for clarity around the 7 h mark); control system switched off at 6 h (recording started 2021/05/01 20:00)

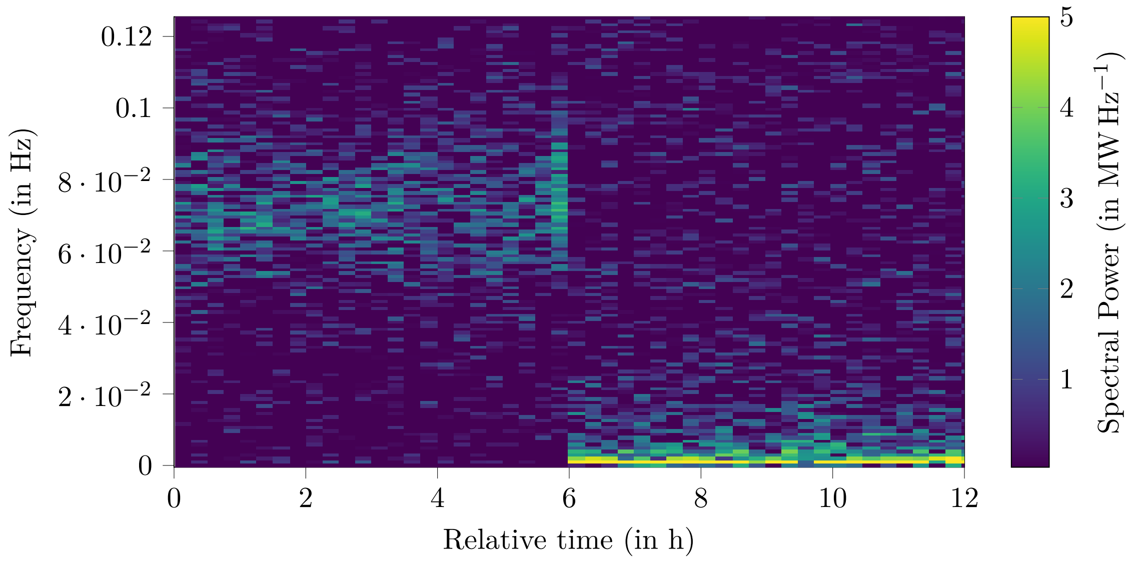


Figure 1.17: Spectrogram of the cavity power in Figure 1.16

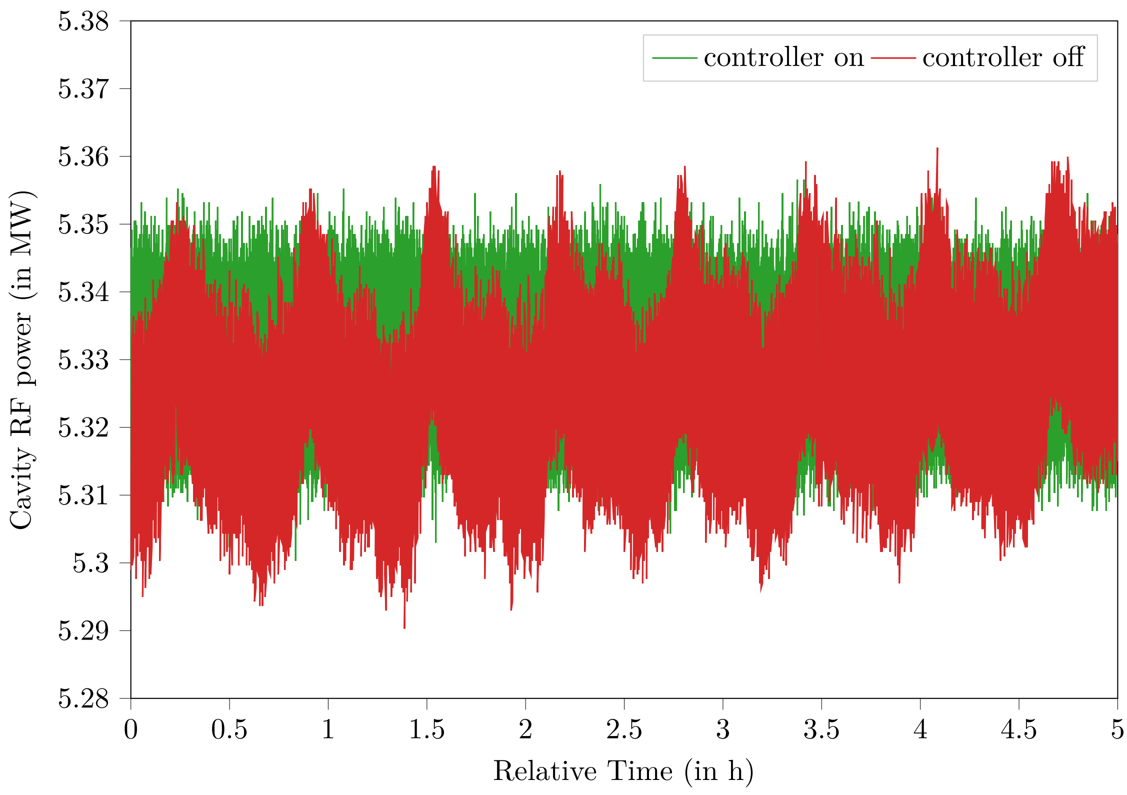


Figure 1.18: Time plots comparing between the control system on and off

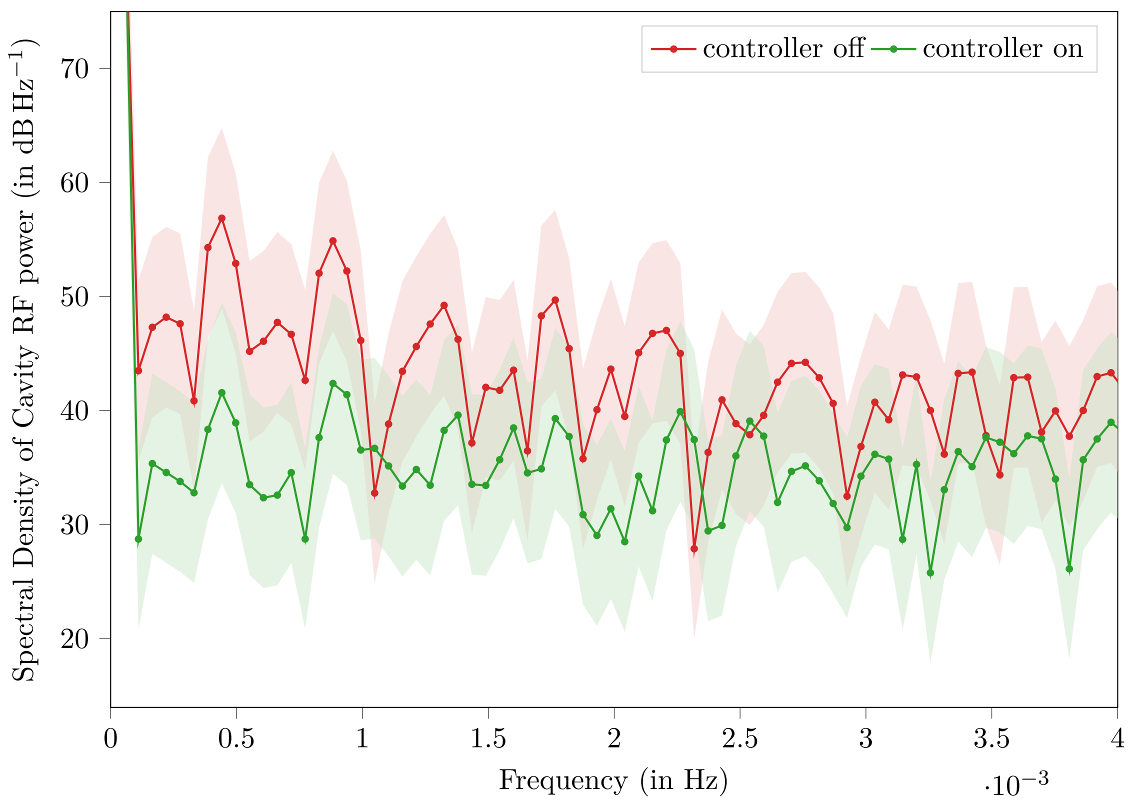
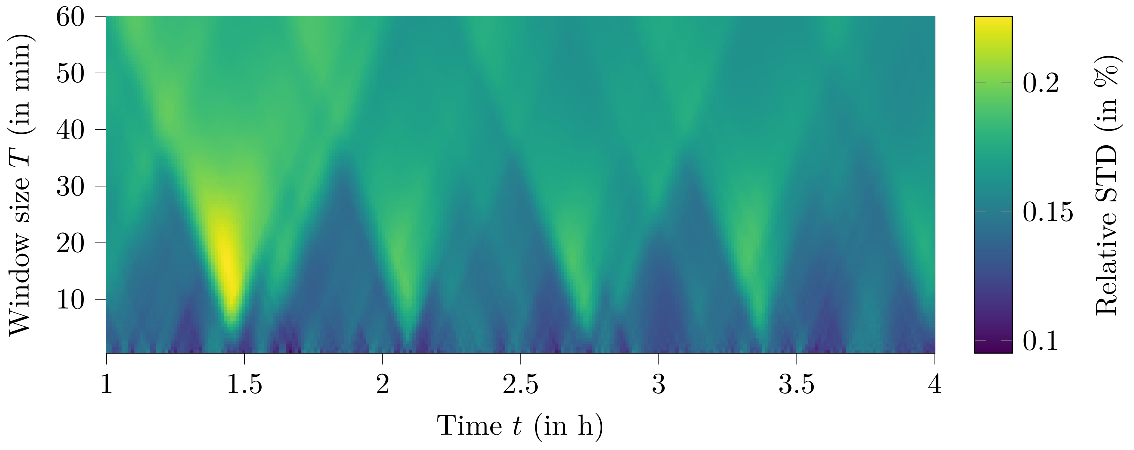
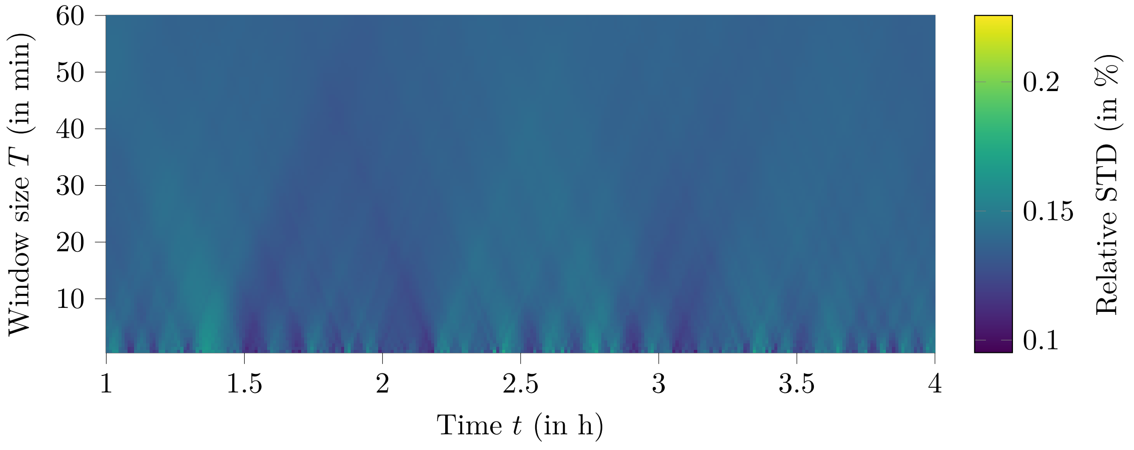


Figure 1.19: Power spectrum of the plots in Figure 1.16 computed with Welch's method; shaded areas show the uncertainty according to **??**



(a) Controller off



(b) Controller on

Figure 1.20: Relative standard deviation $STD\%(t, T)$

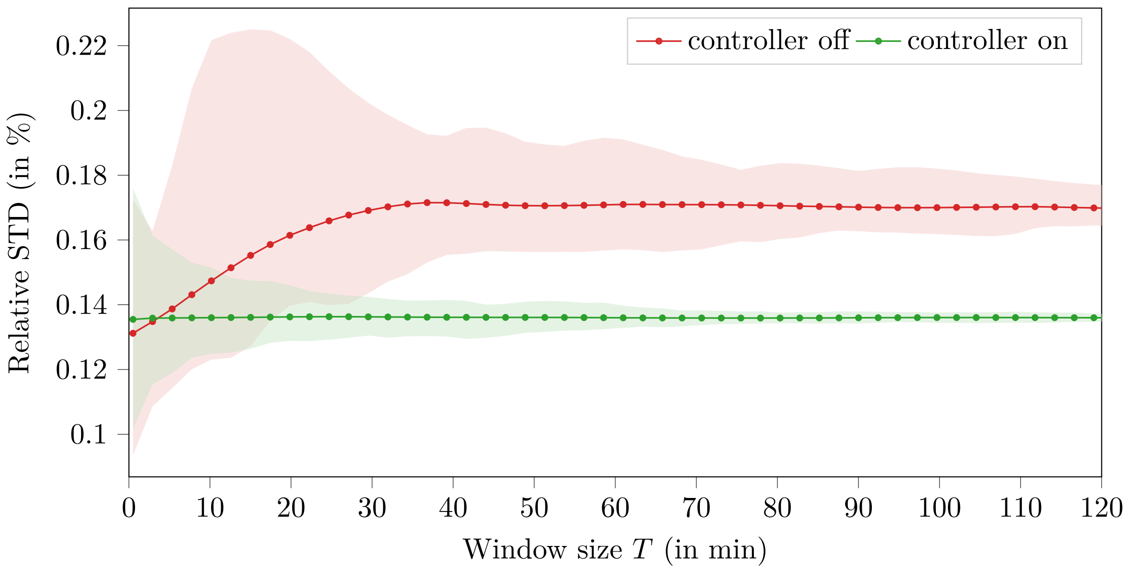


Figure 1.21: Relative standard deviation $STD\%(T)$, shaded areas show $\min\{STD\%(t, T_0)\}$ and $\max\{STD\%(t, T_0)\}$, solid lines show $\text{mean}\{STD\%(t, T_0)\}$ for a fixed window size $T = T_0$

1.6 Further Improvements to the Control System

In this section, two method for improving the control system's performance are examined. First the closed-feedback loop is supplemented with an additional input for one disturbance source, the gun's body temperature. A different approach tested is switching from the gun cavity RF power to the electron charge, as measured by the Faraday cup at the end of the low-energy section, as the controlled signal.

1.6.1 Changing the Controller Architecture: Disturbance Feed-Forward of the Gun Temperature

Instead of changing only the parameter of the PID controller or switching to another type of controller, the strategy presented here relies on changing the architecture over the standard form (see **??**) to include a feed-forward path[6].

To use the disturbance $d(t)$ ($|d(t)| < \infty$) explicitly in the control system, it has to be measurable separately from the control system's output $y(t)$. This can be only done practically if either all disturbance sources or the dominant ones can be identified and measured with some physical sensor. If the signal $d(t)$ is obtainable, it can be fed into the signal path near the controller. In literature, two locations for the disturbance signal to be injected into are described. The method used decides, among other things, which (if any) filtering of $d(t)$ has to be done.

One technique is to regard the disturbance to be similar to the error signal $e(t)$, i.e. the controller input. [Bro02] In this case, the pre-processing, in general, has to include a scaling operation, because the units and magnitudes of $e(t)$ and $d(t)$ are different. If $d(t)$ has non-zero mean, it needs to be subtracted to achieve linear behavior. This means, the needed disturbance filter is

$$D_1(s) = k_{\text{disturbance},1} D'(s), \qquad \text{with } D'(s) = \mathcal{L}\left\{d(t) - \mu_{d(t)}\right\}. \tag{1.30}$$

The second method is to add the filtered disturbance signal to the controller output. [Föl16; Luo+18] In general, this also requires the same pre-processing as before,

$$D_2(s) = k_{\text{disturbance},2} D'(s), \qquad \text{with } D'(s) = \mathcal{L}\left\{d(t) - \mu_{d(t)}\right\}, \tag{1.31}$$

but with a different scaling factor $k_{\text{disturbance},2} \neq k_{\text{disturbance},1}$. This method has the advantage of being potentially faster, because the controller does not occur in the signal path for the disturbance feed-forward, so its dynamics add no additional delay to $d_2(t)$.

In both cases it is common to add another filter to $D_{1,2}$. This is often a lowpass filter $H_d()$ to remove noise. The disturbance filter than becomes

$$D(s) = k_{\text{disturbance}} \cdot H_d(s). \tag{1.32}$$

The new controller architecture is shown in Figure 1.22. Drawn there is the plant $P(s)$ being split into $P_1(s)$ and $P_2(s)$. This is neither necessary nor possible under all circumstances, but feeding forward the disturbance that acts earlier on the plant would lessen the delay of $d(t)$[7]. Besides its main purpose of quickly reacting to a disturbance change, the feed-forward also has the inherent benefit of having no influence on the systems stability. Since $d(t)$ is assumed to be finite, the output has to react to $d(t)$ also with a finite response, as the filtered $d(t)$ is only added to $y(t)$.

[6]The calculations here are done in continuous time to be comparable with the control system model in **??**

[7]A disturbance that acts on the middle of the plant is different from the one that acts on the end.

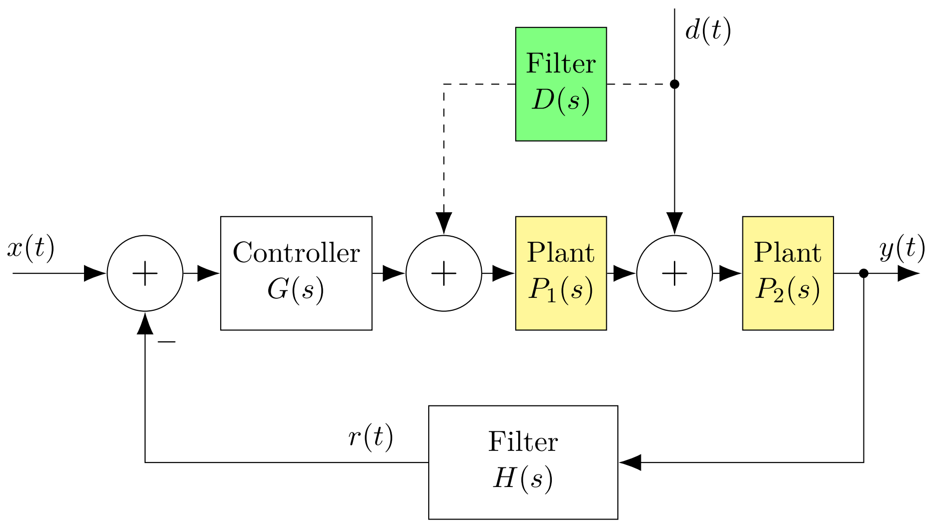


Figure 1.22: Schematic for a control system that uses disturbance feed-forward of the measurable disturbance $d(t)$ in addition to the feed-back loop; changes to the classical control system architecture in **??** highlighted in yellow(changed) and green(added) (based on [Föl16, p. 221])

When testing the newly improved system, there are two observations to be made. First, the stability over a few hours is not improved compared to the system that uses only feedback. But the stability also stays about the same when the control parameters are changed to achieve a slightly "less aggressive" controller. Decreasing k_p by a factor of 2 and k_i by a factor of 4 does not change stability (measured with the MSE over 1 h each) over the feedback-only case.

1.6.2 Feedback on Faraday-Cup Charge Measurements

In the previous part of this chapter, the control input and the measured output was always the cavity power $P_{\text{cavity}} = P_{\text{cavity}}(t)$. This power reading is measured with a RF power probe in the first half-cell of the electron gun.

But as the electrons are accelerated not only by the half-cell but also by the second and third full-cells, regarding only $P_{\text{cavity}}(t)$ as a representative for the electron energy can be misleading. Also, the conversion from RF power to electron energy depends on the (time-varying) properties of the cavity, so there is no simple linear relation between them fixed in time.

An approach that tries to circumvent these issues uses the dark current. The dark current is normally unwanted, as it is the electron emission from the gun, that is not stimulated by the laser. Without a laser pulse, the electric field, generated by the RF inside the first half-cell, accelerates electrons from the vicinity of the cathode that are not freed by the ultraviolet (UV) laser pulse and the photo effect, but rather spontaneous by thermionic emission.

For this dark current, the relation

$$P_B = \frac{I \Delta W}{q}, \tag{1.33}$$

holds. It describes the relationship between the power transferred to the beam P_B by the cavity, the current of the beam I and the energy gain ΔW (with the electron charge q). [Wan08, p. 43]

That means by measuring the beam current $I(t)$ or the charge during one pulse

$$Q_\mathrm{B} = \int_0^{T_\mathrm{pulse}} I(t)\mathrm{d}t = \int_0^{4.5\,\mathrm{\mu s}} I(t)\mathrm{d}t, \tag{1.34}$$

it is possible to calculate at least $^{P_B}/_{\Delta W}$.

This charge reading can then be used for the control system input/output as an alternative to the cavity RF power.

To measure the charge Q_B, a Faraday cup is used. Faraday cups are hollow metal cups designed to catch charged particles, such as electrons, in vacuum. [Rad]

The Faraday cup does not discriminate between the origin of the electrons in the $[0, T_\mathrm{pulse}]$ interval. This is different from the the measurements taken with the Integrating Current Transformer (ICT) and the charge output of one of the Beam Position Monitors (BPMs). They are only sensitive to the laser-generated electron beam. [Nas+19]

The charge is captured as a current over some time. This charge then needs to be transformed into a voltage readable by the FLUTE data acquisition system, which takes voltages as an input. This is done with a Charge Sensitive Amplifier (CSA).

It is similar to an analog integrator (see Figure 1.23). In principle it can be constructed using an operational amplifier as depicted in Figure 1.23.[HHH15, p. 230]

By using one of the fundamental rules of the ideal model of an operational amplifier, that is the voltage difference between the + and − inputs vanishes, the input impedance is

$$Z_\mathrm{in} = \frac{V_\mathrm{in}}{I_\mathrm{in}} = R. \tag{1.35}$$

The transfer function in the Laplace space can be directly stated using Kirchhoff's voltage law[8] and the complex impedance of a capacitor $Z_C = \frac{1}{sC}$ (with $s = j\omega$) as

$$\frac{V_\mathrm{out}(s)}{V_\mathrm{in}(s)} = -\frac{R + \frac{1}{sC}}{R}. \tag{1.36}$$

Substituting $I_\mathrm{in}(s) = \frac{V_\mathrm{in}(s)}{Z_\mathrm{in}} = \frac{V_\mathrm{in}(s)}{R}$ yields

$$\frac{V_\mathrm{out}(s)}{I_\mathrm{in}(s)} = -\left(R + \frac{1}{sC}\right). \tag{1.37}$$

$$\frac{V_\mathrm{out}(s)}{I_\mathrm{in}(s)} = -\frac{1}{sC}. \tag{1.38}$$

With $\mathscr{L}\left\{\int_0^t x(\tau)\mathrm{d}\tau\right\} = \frac{1}{s}X(s)$, the output in the time domain becomes

$$\begin{aligned} V_\mathrm{out}(t) &= \mathscr{L}^{-1}\left\{-I_\mathrm{in}(s)\frac{1}{sC}\right\} && (1.39)\\ &= -\frac{1}{C}\frac{1}{s}I_\mathrm{in}(s) && (1.40)\\ &= -\frac{1}{C}\int_0^t I_\mathrm{in}(\tau)\mathrm{d}\tau. && (1.41) \end{aligned}$$

[8]The sum of all signed voltages around a closed loop is zero.

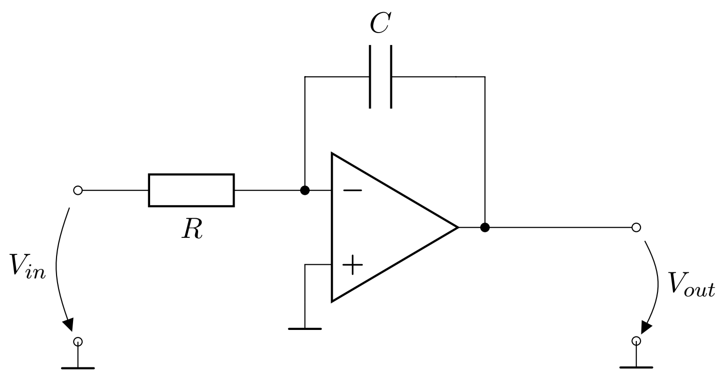


Figure 1.23: Schematic of an inverting integrator using an integrated operational amplifier

The CSA uses an adjustable capacity (to set the range of expected charges) and an analog switch to clear the accumulated charge on the capacitor.

At the moment, the Faraday cup is mounted at the end of the low energy section. The capacity in the CSA is resetted at the start of every pulse and outputs a voltage proportional to the range setting and the measured charge.

To set the range of the CSA, its serial interface together with a serial to Ethernet adapter is used (See an example of the command structure in Listing 5).

Figure 1.24 and Figure 1.25 show the cavity power and the (un-calibrated) charge reading of the Faraday cup for three different cases: the control system switched off, the control system on with the cavity power used as control input and the control system with the Faraday cup's readout (via the CSA) as an input.

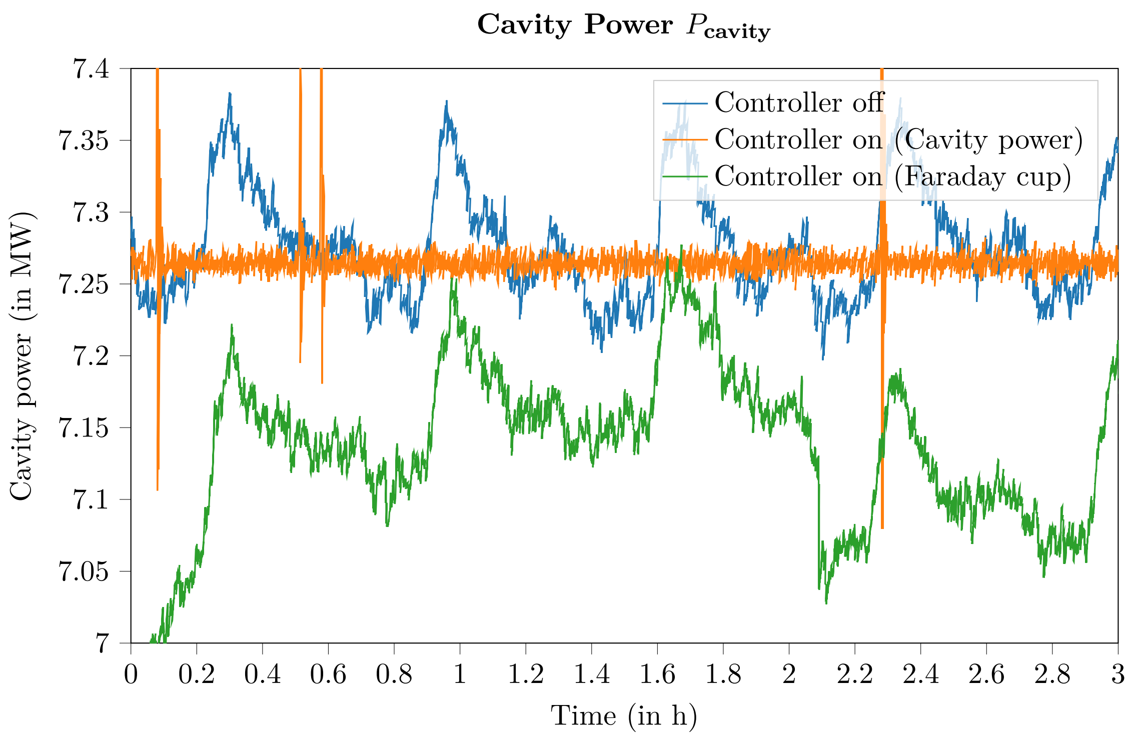


Figure 1.24: Cavity power without control interaction, with the controller acting on the cavity power and with the controller acting on charge; Note the three curves are measured at different times and have no relation to one another

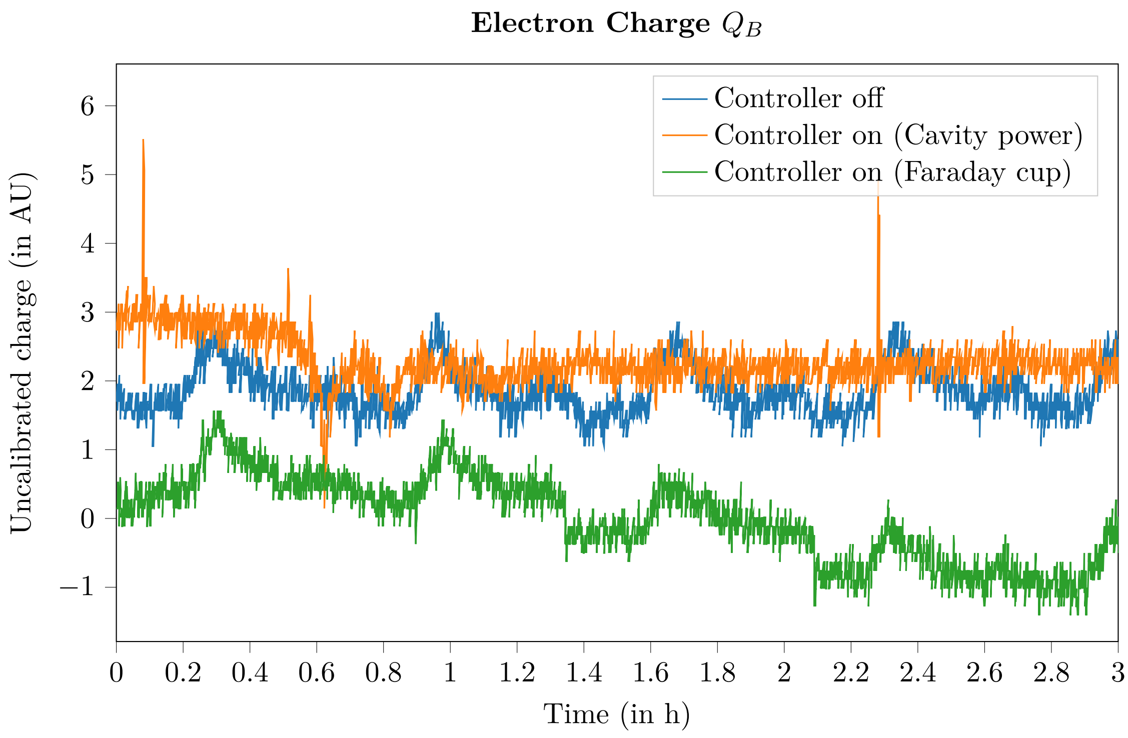
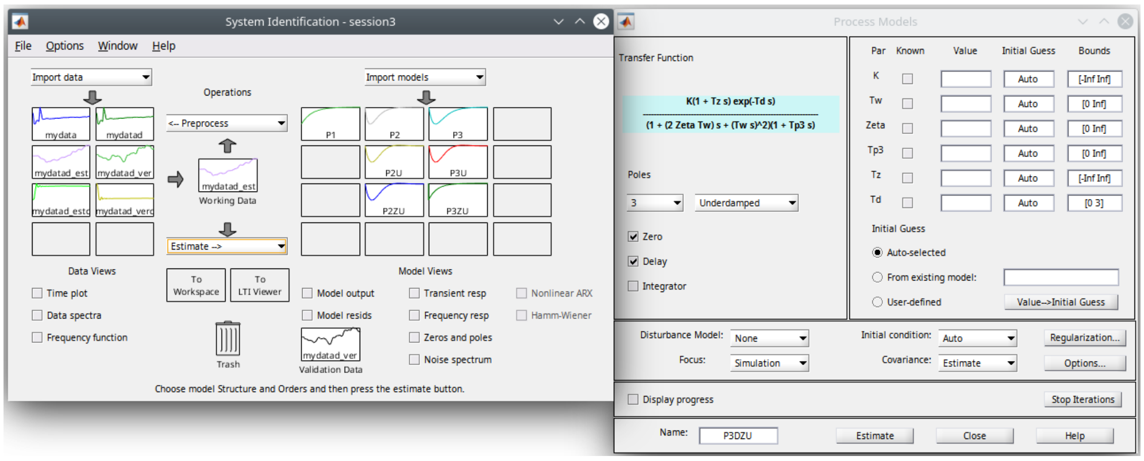


Figure 1.25: Electron charge without control interaction, with the controller acting on the cavity power and with the controller acting on charge; Note the three curves are measured at different times and have no relation to one another

Appendix

A Complementary Material Controller Design

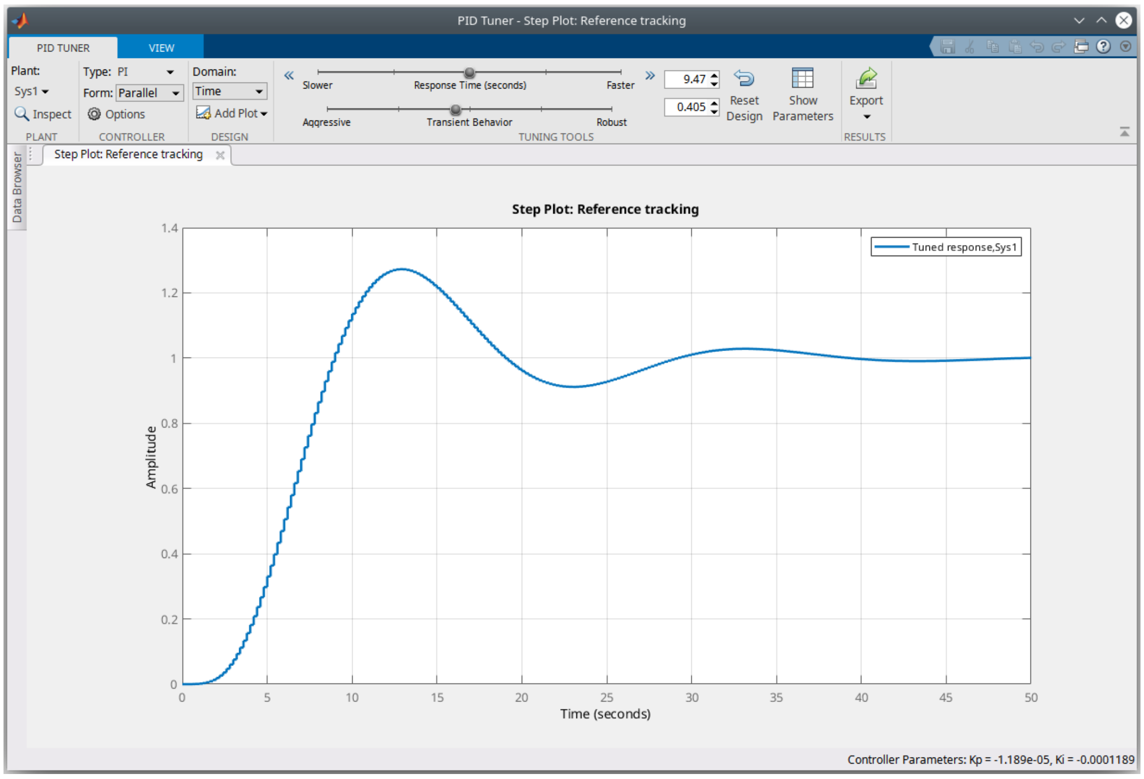


$$\frac{K(1 + Tz\ s)\exp(-Td\ s)}{(1 + (2\ Zeta\ Tw)\ s + (Tw\ s)^2)(1 + Tp3\ s)}$$

Par	Known	Value	Initial Guess	Bounds
K	☐		Auto	[-Inf Inf]
Tw	☐		Auto	[0 Inf]
Zeta	☐		Auto	[0 Inf]
Tp3	☐		Auto	[0 Inf]
Tz	☐		Auto	[-Inf Inf]
Td	☐		Auto	[0 3]

Poles: 3, Underdamped
Zero ✓
Delay ✓
Disturbance Model: None
Focus: Simulation
Initial condition: Auto
Covariance: Estimate
Name: P3DZU

Figure A.1: Screenshot of the Matlab System Identification Toolbox; to the right the process models estimator window



Controller Parameters: Kp = -1.189e-05, Ki = -0.0001189

Figure A.2: Screenshot of the Matlab PID Tuner from the Control Systems Toolbox

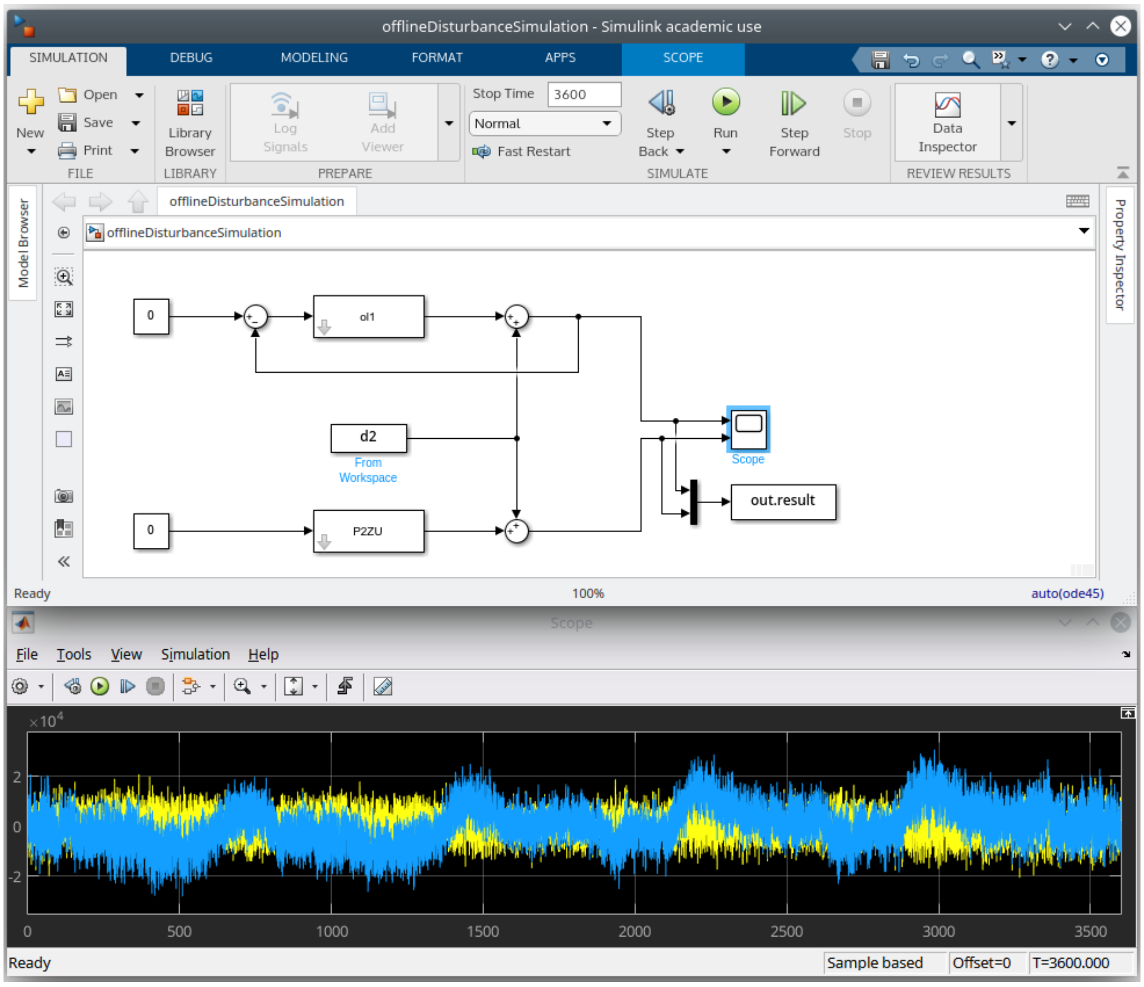
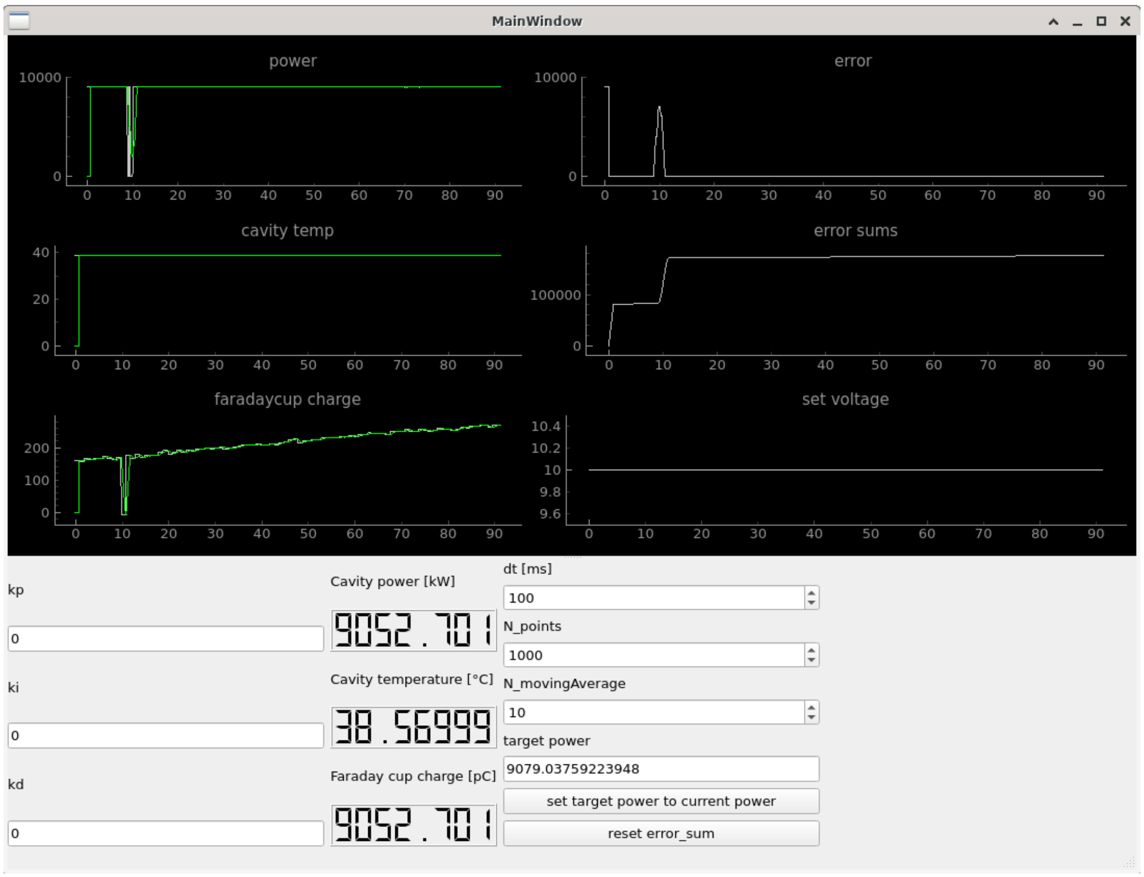


Figure A.3: Simulink model to evaluate the designed controller together with the measurement filter (`ol1`) compared to the uncontrolled system (in `P2ZU`) using measured disturbance data (in the vector `d2`); below a view of the scope data



kp: 0

ki: 0

kd: 0

Cavity power [kW]: 9052.701

Cavity temperature [°C]: 38.56999

Faraday cup charge [pC]: 9052.701

dt [ms]: 100

N_points: 1000

N_movingAverage: 10

target power: 9079.03759223948

Figure A.4: Screenshot of the control system's GUI application

Listing 5: Java class of the PCB421A25 charge amplifier demonstrating the command structure and checksum calculation for integration of the amplifier into Control System Studio (CSS)

```
class PCB421A25 {
  final static char STX = '\u0002';
  static enum FixedRange {
    RANGE_1000000("01"),
    RANGE_500000("02"),
    RANGE_200000("03"),
    RANGE_100000("04"),
    RANGE_50000("05"),
    RANGE_20000("06"),
    RANGE_10000("07"),
    RANGE_5000("08"),
    RANGE_2000("09"),
    RANGE_1000("10"),
    RANGE_500("11"),
    RANGE_200("12"),
    RANGE_100("13");
    public final String command;
    private FixedRange(String command) {
      this.command = command;
    }
  }

  public PCB421A25() {};

  public void setFixedRange(FixedRange fixedRange) {
    String command=STX+"c"+"0"+fixedRange.command;
    command += calculateChecksum(command);
    sendCommand(command);
  }

  public boolean setVariableRange(int variableRange) {
    if (!( variableRange>100 && variableRange<1000000)) return false;
    String command=STX+"d"+"0"+String.format("%06d", variableRange);
    command += calculateChecksum(command);
    sendCommand(command);
    return true;
  }

  private char calculateChecksum(String command) {
    int checksum=0;
    for(int i=0;i<command.length();i++)
      checksum+=(int)command.charAt(i);
    String checksum_hexstr=Integer.toHexString(checksum).toUpperCase();
    return checksum_hexstr.charAt(checksum_hexstr.length()−1);
  }

  private void sendCommand(String command){
    System.out.println("Command to send: "+command+" (length: "+command.length()+")");
    // [...]
  }

  public static void main(String[] args) {
    PCB421A25 chargeSensitiveAmplifier = new PCB421A25();

    //Test fixed ranges
    System.out.print("Fixed range 1000000;\t");
    chargeSensitiveAmplifier.setFixedRange(FixedRange.RANGE_1000000);
        System.out.print("Fixed range 100;\t");
        chargeSensitiveAmplifier.setFixedRange(FixedRange.RANGE_100);

        //Test variable ranges
    System.out.print(”Variable range 123456;\t”);
    chargeSensitiveAmplifier.setVariableRange(23500);
    System.out.print(”Variable range 999;\t”);
    chargeSensitiveAmplifier.setVariableRange(999);
    }
}
```

B Lab Test and Measurement Equipment

B.1 Benchtop multimeters

B.1.1 Agilent 34411A

Table B.4: Agilent 34411A - Specifications

Specification	Value
Digits	6 1/2
Measurement method	cont integrating multi-slope IV A/D converter
Accuracy (10 V range, 24 hours)	0.0015 %+0.0004 % (% of reading + % of range)
Bandwidth	15 kHz (typ.)

Table B.5: Agilent 34411A - Relevant SCPI commands

Description	Example command	Example return
Read current measurement	`READ?`	`+2.84829881E+00` (2.848 V)

B.1.2 Keysight 34470A

Table B.6: Keysight 34470A - Specifications

Specification	Value
Digits	7 1/2
Measurement method	cont integrating multi-slope IV A/D converter
Accuracy (10 V range, 24 hours)	0.0008 %+0.0002 % (% of reading + % of range)
Bandwidth (10 V range)	15 kHz (typ.)

Table B.7: Keysight 34470A - Relevant SCPI commands

Description	Example command	Example return
Read current measurement	`READ?`	`+9.99710196E+00` (9.997 V)

B.2 Data Acquisition/Switch Unit

B.2.1 Keysight 34972A

Table B.8: Keysight 34972A - Specifications

Specification	Value
34907A (Multifunction module)	
DAC range	±12 V
DAC resolution	16 bit ($24\,\mathrm{V}/2^{16}$ = 366.21 µV per bit)
DAC maximum current	10 mA
34901A (20 channel multiplexer)	

Table B.9: Keysight 34972A - Relevant SCPI commands

Description	Example command	Example return
Read current measurement	`READ?`	`+2.00200000E+01` (20.02 °C)
Set DAC voltage of ch 204 to 3.1 V	`SOUR:VOLT 3.1,(@204)`	

B.3 RF signal generator

B.3.1 Rohde and Schwarz SMC100A

Table B.10: Rohde and Schwarz SMC100A - Specifications

Specification	Value
Frequency range	9 kHz to 3.2 GHz
Maximum power level	17 dBm
SSB phase noise (@ 1 GHz, $f_o = 20\,\text{kHz}$, $BW = 1\,\text{Hz}$)	-111 dBc
Level error	<0.9 dB

Table B.11: Rohde and Schwarz SMC100A - Relevant SCPI commands

Description	Example command	Example return
Set RF power level to 10.5 dBm	`SOUR:POW 10.5`	
Set RF frequency to 3.1 GHz	`SOUR:FREQ:FIX 3.1e9`	
Enable the RF output	`OUTP on`	

B.4 RF power meter

B.4.1 HP E4419B

Table B.12: HP E4419B - Specifications

Specification	Value
Digits	4
Accuracy (abs. without power sensor)	±0.02 dB
Power probe: E4412A	
Frequency range	10 MHz to 18 GHz
Power range	-70 dBm to 20 dBm

Table B.13: HP E4419B - Relevant SCPI commands

Description	Example command	Example return
Measure power on input 1	`MEAS1?`	`+2.89435802E+000` (2.894 dBm)

B.5 Vector Network Analyzer

B.5.1 Agilent E5071C

Table B.14: Agilent E5071C - Specifications

Specification	Value
Frequency range	9 kHz to 8.5 GHz

B.6 Phase noise analyzer

B.6.1 Holzworth HA7062C

Table B.15: Holzworth HA7062C - Specifications

Specification	Value
DUT input frequency	10 MHz to 6 GHz
Measurement bandwidth	0.1 Hz to 40 MHz offsets

Acknowledgments

First I would like to thank Prof. Anke-Susanne Müller and the whole FLUTE team for providing me with the opportunity to write my thesis at their research facility. FLUTE proved to be a one-of-a-kind machine and I really appreciate the work that has already been put into it.

I am very grateful to Prof. Dr.-Ing. John Jelonnek from the Institute for Pulsed Power and Microwave Technology for taking the responsibility of being my first reviewer. Also many thanks to his PhD students Benjamin Ell and Alexander Marek for listening patiently to my presentations and providing great support in optimizing them.

I'm especially in debt to my excellent advisor Dr. Nigel John Smale, for always having the time to go over both fundamentals and nifty details of electronics and physics alike. Without his support I wouldn't even be able to switch on FLUTE by now and much less a successful thesis would've been possible.

Valuable inputs from Andreas Böhm regarding the RF attenuator and electronics in general were a great help.

Thanks to Igor Križnar for constructive discussions and programming the graphical user interface and its back-end to control the charge sensitive amplifier.

Construction of the charge sensitive amplifiers connection box by Jürgen Schmid is greatly appreciated.

Thanks to Olena Manzhura from KIT-IPE for her support and for prove-reading the thesis multiple times.

Finally but maybe most importantly, a big thanks goes out to my family, that is my mom Sonja, Stefan and my grandparents Ernst and Gretel. I owe them my deepest appreciation for their support over the last years.

Acronyms

CERN Conseil Européen pour la Recherche Nucléaire. 2

CSS Control System Studio. 10

cSTART compact Storage ring for Accelerator Research and Technology. 4

EPICS Experimental Physics and Industrial Control System. 4

FLUTE Ferninfrarot Linac- und Test-Experiment. v, vii, 1, 2, 5

KIT Karlsruhe Institute of Technology. v

LINAC Linear Accelerator. v

LLRF Low Level RF. 5

RF Radio Frequency. v, vii, 1, 2, 4–6

THz terahertz. v, 4, 5

Bibliography

[ÅH95] Åström and Hägglund. *PID controllers*. Research Triangle Park, N.C: International Society for Measurement and Control, 1995. ISBN: 1556175167.

[Bro02] Coleman Brosilow. *Techniques of model-based control*. Upper Saddle River, N.J: Prentice Hall, 2002. ISBN: 013028078X.

[Cam11] Luke Campagnola. *PyQtGraph - Scientific Graphics and GUI Library for Python*. 2011. URL: `https://www.pyqtgraph.org/` (visited on 06/14/2021).

[Com21] Riverbank Computing. *PyQt*. 2021. URL: `https://riverbankcomputing.com/software/pyqt/` (visited on 06/17/2021).

[DHB19] Iván D. Díaz-Rodríguez, Sangjin Han, and Shankar P. Bhattacharyya. *Analytical Design of PID Controllers*. Springer International Publishing, May 29, 2019. 316 pp. ISBN: 3030182274.

[Dod15] Stephen J. Dodds. *Feedback Control*. Springer-Verlag GmbH, July 1, 2015. ISBN: 1447166744.

[Föl16] Otto Föllinger. *Regelungstechnik*. Vde Verlag GmbH, June 2016. ISBN: 3800742012.

[HHH15] Paul Horowitz, Winfield Hill, and J. C. Holt. *The Art of Electronics*. 2nd ed. Cambridge University Pr., Apr. 2, 2015. ISBN: 0521809266.

[Hun07] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: `10.1109/MCSE.2007.55`.

[Kam02] Kammeyer. *Digitale Signalverarbeitung Filterung und Spektralanalyse*. StuttgartLeipzig-Wiesbaden: Teubner, 2002. ISBN: 9783519461227.

[León15] Fernando León. *Signale und Systeme*. BerlinBoston, Mass: De Gruyter Oldenbourg, 2015. ISBN: 9783110403855.

[Luo+18] Yong Luo et al. "Feedforward Control Based on Error and Disturbance Observation for the CCD and Fiber-Optic Gyroscope-Based Mobile Optoelectronic Tracking System". In: *Electronics* 7.10 (2018). ISSN: 2079-9292. DOI: `10.3390/electronics7100223`. URL: `https://www.mdpi.com/2079-9292/7/10/223`.

[Nas+19] Michael Nasse et al. "First Electron Beam at the Linear Accelerator FLUTE at KIT". en. In: *Proceedings of the 10th Int. Particle Accelerator Conf.* IPAC2019 (2019), Australia. DOI: `10.18429/JACOW-IPAC2019-MOPTS018`.

[Opp10] Alan Oppenheim. *Discrete-time signal processing*. Upper Saddle River, NJ: Pearson, 2010. ISBN: 9780131988422.

[Rad] RadiaBeam. *Faraday Cups*. URL: `http://www.radiabeam.com/upload/catalog/pdf/14272334342015-03-24_faraday-cups.pdf`.

[The21a] The Mathworks. *Open PID Tuner for PID tuning - MATLAB pidTuner*. June 16, 2021. URL: `https://de.mathworks.com/help/control/ref/pidtuner.html` (visited on 06/16/2021).

[The21b] The Mathworks. *PID Tuning Algorithm - MATLAB & Simulink*. June 16, 2021. URL: `https://de.mathworks.com/help/control/getstart/pid-tuning-algorithm.html` (visited on 06/16/2021).

[The21c] The Qt Company. *Qt Framework*. 2021. URL: `https://doc.qt.io/` (visited on 06/15/2021).

[The21d] The Qt Company. *QTimer Class*. 2021. URL: `https://doc.qt.io/qt-5/qtimer.html` (visited on 06/12/2021).

[Wan00] Liuping Wang. *From plant data to process control : ideas for process identification and PID design*. London New York: Taylor & Francis, 2000. ISBN: 0748407014.

[Wan08] Thomas Wangler. *RF linear accelerators*. Weinheim: Wiley-VCH, 2008. ISBN: 9783527623426.

[ZN42] J. G. Ziegler and N. B. Nichols. “Optimum Settings for Automatic Controllers”. In: *Transactions of the ASME* 64 (1942), pp. 759–768.

[ZR10] Serge Zacher and Manfred Reuter. *Regelungstechnik für Ingenieure*. Vieweg+Teubner Verlag, Dec. 7, 2010. 514 pp. ISBN: 9783834898371.