DEPARTMENT OF PHYSICS
Institute for Beam Physics and Technology

Stability Improvements at FLUTE
(Verbesserung der Stabilität von FLUTE)

Master thesis
of

Marvin-Dennis Noll

at the Institute for Beam Physics and Technology

Reviewer:	Prof. Dr.-Ing. John Jelonnek (IHM)
Second Reviewer:	Prof. Dr. Anke-Susanne Müller (IBPT)
Advisor:	Dr. Nigel Smale (IBPT)

15.11.2020 – 01.07.2021

Erklärung zur Selbstständigkeit

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde und dass ich die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der gültigen Fassung vom 24.05.2018 beachtet habe.

Karlsruhe, den 01.07.2021, _____

Marvin-Dennis Noll

Als Prüfungsexemplar genehmigt von

Karlsruhe, den 01.07.2021, _____

Prof. Dr.-Ing. John Jelonnek (IHM)

Contents

List of Figures

List of Tables

1. Controller Design and Evaluation

In this chapter stabilizing the power output of FLUTE by means of a control system is examined. This is different than previous attempts in that a feedback control system tries to compensate short term disturbances and long term drifts *actively* by interfering with some part of FLUTE that has influence of the output power.

The next sections describe the chosen architecture of a suitable control system as well as some implementation details and the tuning of necessary controller parameters.

1.1 Concept and Architecture

The general structure of a closed loop control system is shown in Figure 1.1.

$y(t)$ is the physical quantity which should follow a certain time trajectory $x(t)$ or be kept constant (then $x(t) = x = const.$). If there are no disturbances $d(t)$ or the disturbances are deterministic (i.e. $d(t)$ is known $\forall t$), then an open loop system would be possible. In that case the role of the controller would be to merely to set its output $u(t)$ according to the system dynamics of the plant[1] (represented by its impulse response $g(t)$).

In most real world scenarios, $d(t)$ originates from a stochastic process and thus is unknown. Too remedy the negative influences of $d(t)$, the output $y(t)$ is measured and feed back. Based on the error $e(t)$, which is the difference between the set value and the actual value defined by

$$e(t) = x(t) - y(t), \tag{1.1}$$

the controller can react accordingly.

[1] Assuming $g(t)$ is a stable LTI system

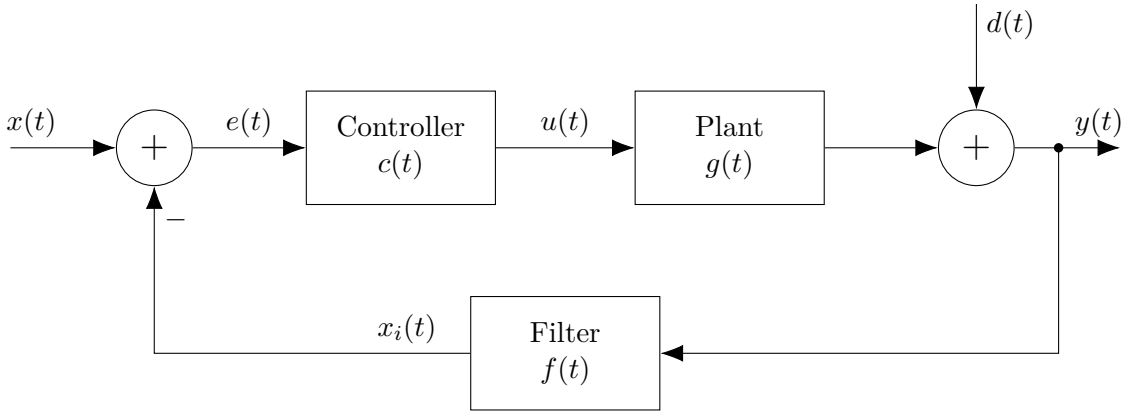


Figure 1.1: General structure of a closed loop feedback control system

1.2 Plant Identification

1.2.1 Principle

Before choosing an appropriate controller, some insight of the system response has to be obtained. For that reason, next the plant transfer function is obtained. In the time domain, the transfer function is the response of the system to an impulse on the input. So per definition, in the special case here, this would mean changing the RF attenuator quickly from a big attenuation to a small attenuation and then back. This is not easy to measure and a single measurement is very susceptible to noise. Therefore it is more common to measure the step response instead and to average over several step responses.

When there is no prior knowledge over the system, the identification is sometimes done with a (pseudo) random binary sequence to excite the system with step functions of different lengths. Then it is necessary that some of the steps last longer than a few dominant time constants of the system. To get the transfer function of the system from its step response, several methods are common, including correlation based and frequency response based algorithms.

1.2.2 Identifying the system response of the FLUTE LLRF

The input sequence is generated by modulating the RF attenuator around a base attenuation. As a trade off between high SNR and driving the LLRF in a "save" region, the control voltage span is chosen to be 4 V in total (7 V to 11 V around the base control voltage of 9 V).

To get a random binary sequence, depending on the outcome of a binomial random process, the voltage is toggled between 7 V and 11 V according to Listing 1.1. With the parameter `toggleP`, the average length of one constant voltage level can be controlled.

Listing 1.1: Function to get a random binary sequence

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

In a test run over 6 hours (after FLUTE had stabilized), the attenuator was driven with such a random sequence. The result is shown in Figure 1.2.

The time signals are then split into a estimation data set (about 80 % of samples) and a validation data set (the remaining ≈ 20 %).

The two data sets are then loaded into the MATLAB *System Identification Toolbox* (SIT). With the SIT, first the means of both sets and both the input and output are removed, which is required by the estimators used. Then with different numbers of poles and zeros, the transfer function is estimated. After trying several settings, three promising candidates

$$\hat{G}_1(s) = \frac{71.37\,s + 0.5966}{s^3 + 0.8208\,s^2 + 0.328\,s + 0.002733} \tag{1.2}$$

$$\hat{G}_2(s) = \frac{-0.2125\,s + 70.85}{s^2 + 0.8022\,s + 0.3202} \tag{1.3}$$

$$\hat{G}_3(s) = \frac{79.12}{s + 0.3502} \tag{1.4}$$

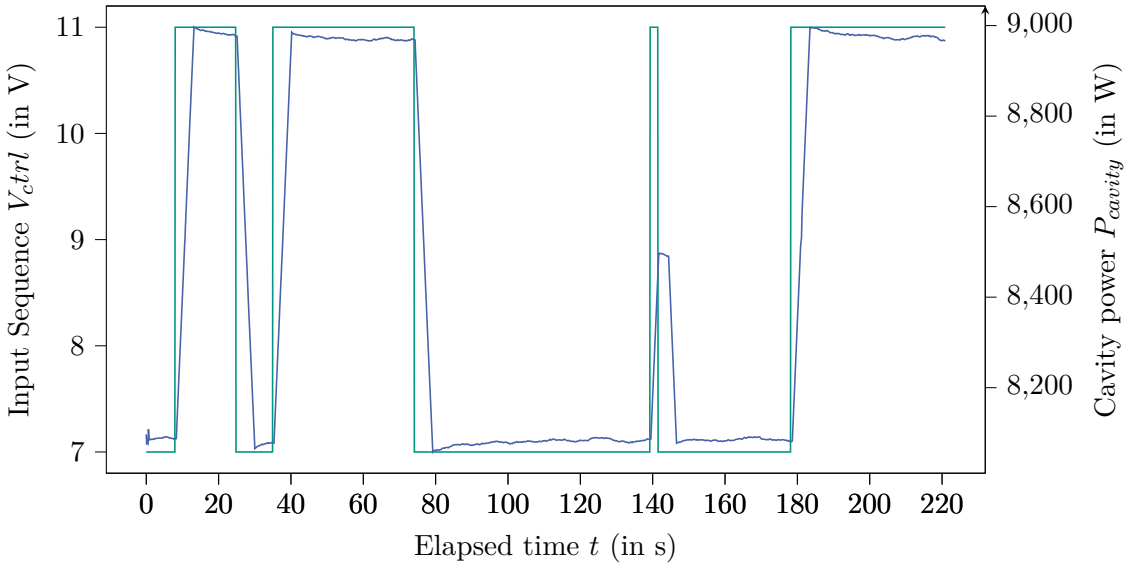


Figure 1.2: Small section of the input sequence (green) and the system response (blue)

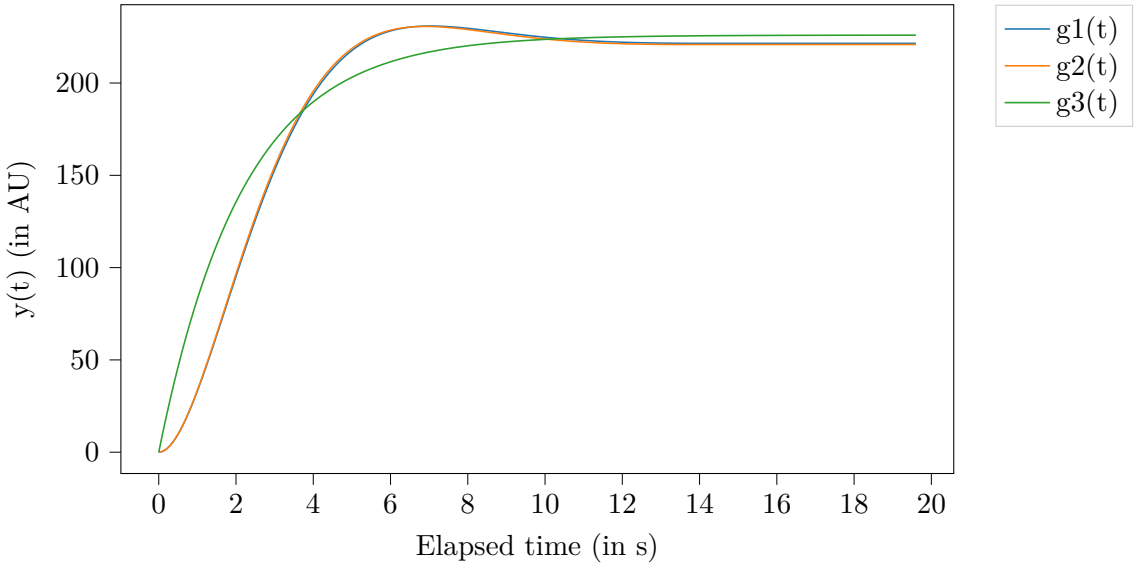


Figure 1.3: Step responses of the systems $\hat{G}_1(s)$, $\hat{G}_2(s)$, $\hat{G}_3(s)$

emerge.

For these transfer functions, the (single) step responses are given in Figure 1.3.

To check the accuracy of the estimations, the SIT is used to do a simulation with the $\hat{G}_i(s)$ and the validation data set (see Figure 1.4).

Figure 1.4 shows that a first order system with one pole and no zeros is not a sufficient approximation as there is no overshoot, since it is not possible in a first order system. The second and third order systems $\hat{G}_2(s)$ and $\hat{G}_3(s)$ show much better fits.

The important conclusion of this section is that the plant can be modeled as a **second order system**. This info is needed when choosing a controller in the next section.

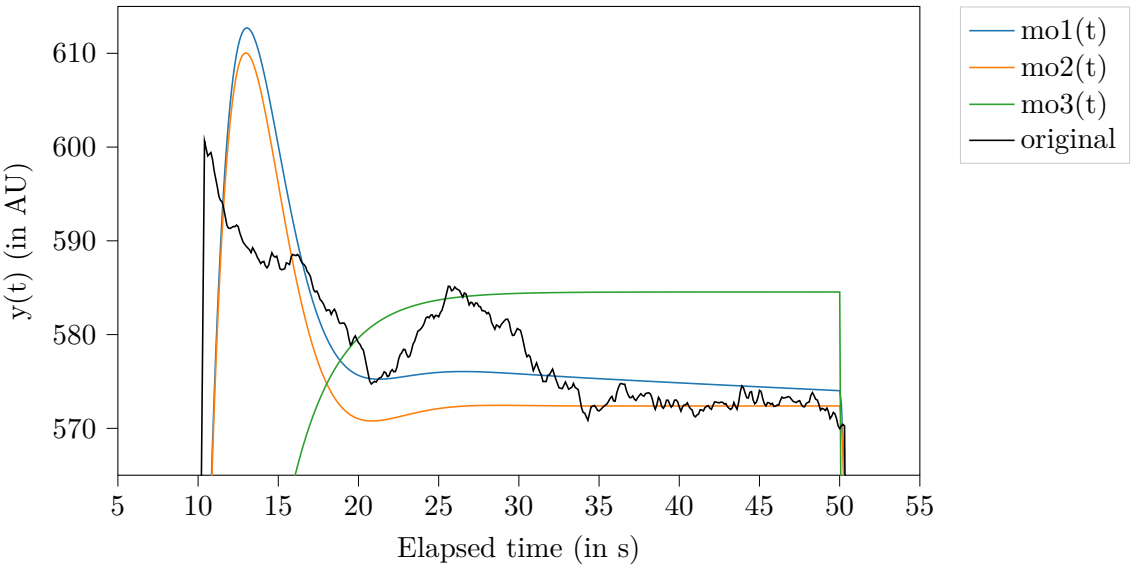


Figure 1.4: Validating the estimations $\hat{G}_1(s)$, $\hat{G}_2(s)$, $\hat{G}_3(s)$ against real data from the validation data set

1.3 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 has 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 XX.

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:

$$\mathrm{MA}_{x,\text{non-causal},L}[n] = \frac{1}{L} \sum_{i=n-\frac{L-1}{2}}^{n+\frac{L-1}{2}} x[i] \tag{1.5}$$

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

$$\mathrm{MA}_{x,\text{causal},L}[n] = \frac{1}{L} \sum_{i=n-(L-1)}^{n} x[i] \tag{1.6}$$

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 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 $h[n]$:

$$h[n] = \frac{1}{N}[\underbrace{1,\ 1,\ ...,\ 1}_{N}] \tag{1.7}$$

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

$$\frac{\tau_g(f)}{T_s} = \frac{\mathrm{d}\phi(f)}{\mathrm{d}f} \tag{1.8}$$

which is given normalized to the sampling time T_s [1, 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[1, p. 165]:

$$\frac{\tau_g(f)}{T_s} = \frac{\mathrm{d}\phi(f)}{\mathrm{d}f} = \frac{\mathrm{d}\phi}{\mathrm{d}f} = \frac{N}{2} \tag{1.9}$$

With a sampling time of $T_s = 1/5\,\mathrm{Hz} = 200\,\mathrm{ms}$ and $N = 100$ the group delay is 10 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 in case of 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 then yields the impulse response of the desired FIR filter $h_{\mathrm{FIR}}[n]$.[2, 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 IIR filter is designed with the impulse invariance method and an analogue Butterworth filter. This method could be interpreted as sampling the infinitely long analogue impulse response.[2, p. 497] With SciPy using `b,a=signal.butter(N,fc,'lowpass',fs,)`, the coefficients of an IIR with this method can be calculated. For an IIR filter, the group delay cannot be calculated with Equation 1.9 and it is in general frequency depend.

Figure 1.5 shows the impulse responses of the moving average, the FIR lowpass and the IIR lowpass (truncated to $N = 100$).

In Figure 1.6, 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.

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

[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.

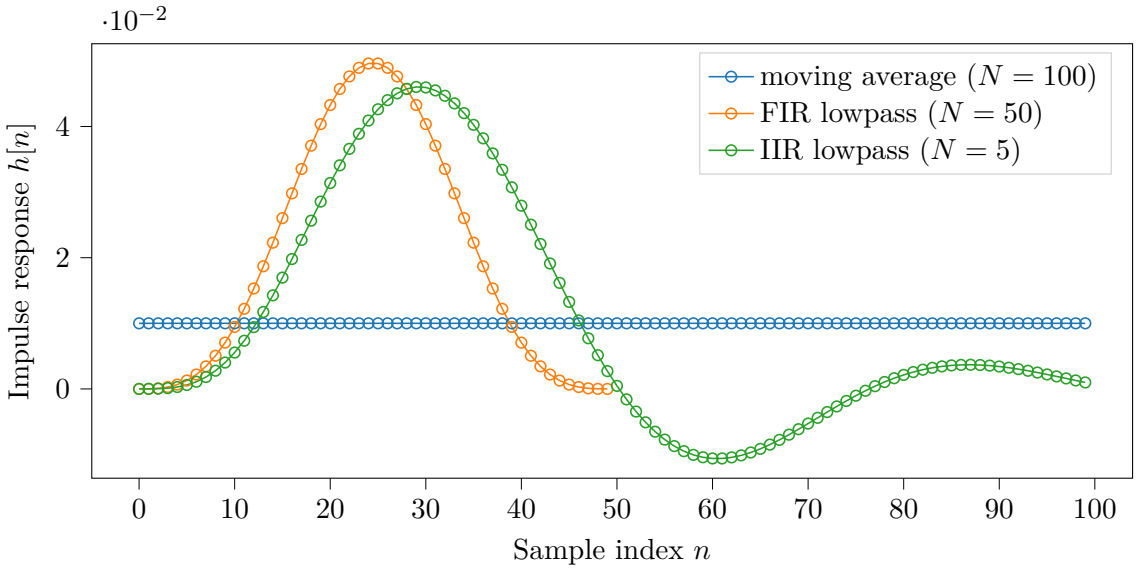


Figure 1.5: Impulse responses of a moving average filter ($N = 100$), a FIR lowpass ($N = 50$, $f_c = 0.1\,\text{Hz}$) and a IIR Butterworth lowpass ($N = 50$, $f_c = 0.1\,\text{Hz}$)

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 an FIR lowpass filter is used.

Real-time implementation of an FIR Filter in Python

Similar to Equation 1.6, 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 1.6 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()`

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

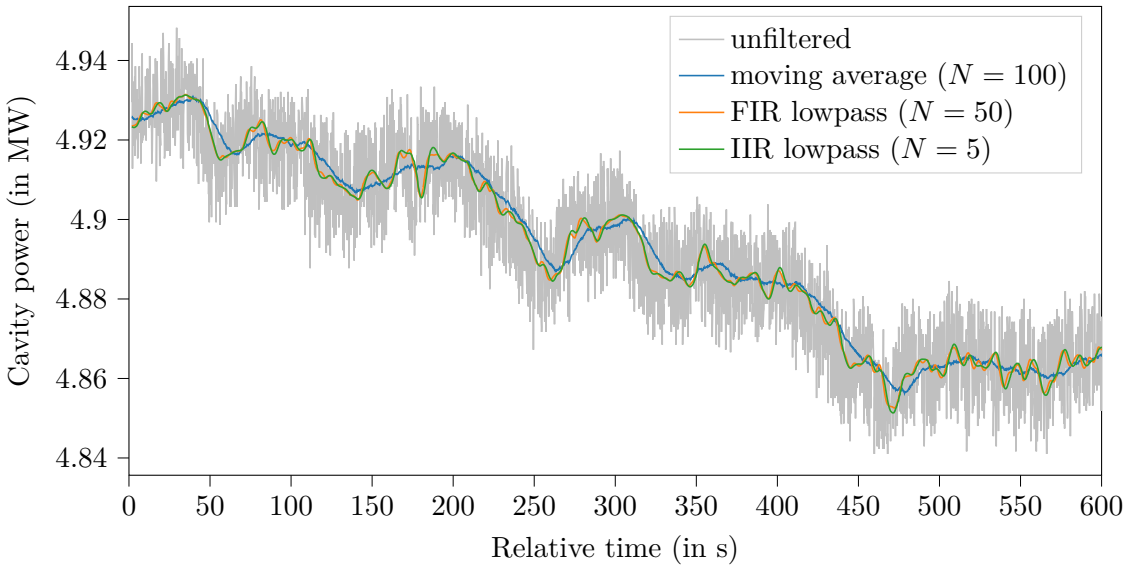


Figure 1.6: Effects of the three different lowpass filters in Figure 1.5 on noisy data

1.4 Controller design

For many control problems, especially if the plant behaves approximately as an LTI system and the system is of low order, a simple PID (proportional, integral and derivative) controller is a good starting point (visualized in Figure 1.7).

A PID controller uses the error $e(t)$, the temporal integral of the error $e_i(t)$ and the temporal derivative of the error $e_d(t)$ as an input and outputs a weighed sum of them. While the unmodified error signal represents the current error, the integrated and derived error signals allow to controller to "see" in the past and predict the future.

Often simplifications, such as a pure P (only $k_p \neq 0$) or a PI (only k_p, $k_i \neq 0$) controller are valid as well. Since the plant has been identified to be a second order system, a simple P controller is not enough to bring the steady state error to zero (see). So at least a PI controller is needed.

As a starting point for software development and parameter tuning, the parameters $k_p = 0.00001$ and $k_i = k_d = 0$ are chosen. This ensures during development the system basically does nothing but still shows changing values at the controller output.

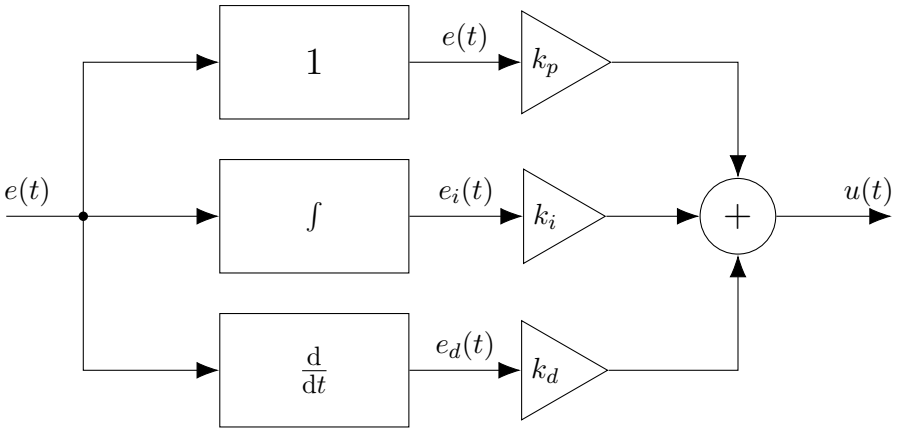


Figure 1.7: Block diagram of a generic PID controller

1.5 Inputs and Outputs

Since the control algorithm should be implemented, tested and used online on the actual accelerator instead of only operate on simulated data, there is the need for fast and reliable interfaces to the machine. Following, "input'' refers to the signal going into the control algorithm (i.e. the measured $y(t)$), while "output'' is the output of the control algorithm $u(t)$.

1.5.1 Input

Depending on which value is chosen to be controlled, filtering of the input signal could be mandatory.

In the case of the cavity RF power the signal jumps to zero each time a breakdown occurs, shortening the RF supply. These outliners are not representative of the average RF power inside the cavity over multiple pulses and thus would greatly impair the controller performance. For that reason, before any further filtering to remove noise etc, a breakdown removal filter is used (Listing 1.3). In principle the new power value is checked to be inside a band which size is determined by the mean deviation of the N_{filt} previous values and a scaling m. The percentile differences are used here as they are robust against outliners (i.e. other breakdowns) in the N_{filt} previous values opposed to a normal standard deviation. The scaling with $(2 * 1.2815)^{-1}$ is used to make the mean deviation comparable to a standard deviation.

Listing 1.3: Breakdown removal

```
#...
if(abs(P[i]-np.median(P[i-3*Nfilt:i-Nfilt]))<
m*(np.percentile(P[i-3*Nfilt:i-Nfilt],90)-np.percentile(P[i-3*Nfilt:i-Nfilt],10)/(2*1.2815))):
    P_filt =np.append(P_filt,P[i])
else:
    breakdown_locations_predicted=np.append(breakdown_locations_predicted,i)
    P_filt =np.append(P_filt,np.median(P[i-3*Nfilt:i-Nfilt]))
#...
```

1.5.2 Output

For the control system to work the controller needs some way of influencing the plant. For that the output of the FLUTE LLRF vector modulator is controlled by a RF attenuator (see **??**).

1.6 Software Design

As integrating a new subsystem into EPICS takes some time and effort and the control system is designated a temporary solution, it is more viable to operate it as an independent system.

Before choosing a programming language, software frameworks, etc. the key requirements for the software are discussed:

- Communication with EPICS to get values and with the RF attenuator
- Efficient and lightweight to achieve clock cycles times of a most 0.1 s
- Easy implementation of a (time discrete) PID controller
- GUI to show input, output and error signals
- Possibility to log signals to file for documentation

With these in mind first programming languages are regarded. As there are EPICS libraries for both C++ and Python, these two languages are examined in more detail.
While C++ as a compile language promises speed, all other requirements are possible but would take much greater effort in C++ compared to Python. For that reason, in the following a small test program is written to evaluate the fastest clock cycle possible with a simple Python program.

shows that retrieving one value of an EPICS channel and setting a new attenuator voltage takes only about 20 ms, thus using C++ is not necessary and Python can be utilized instead.

To create a PID controller in software instead of a continuous time system, only discrete time implementations are possible. Choosing a high clock cycle frequency however approximates the continuous time system. To get the error signals for the integral and derivative part, the integral is replaced with a cumulative recursive sum as

$$e_i[n] = e_i[n-1] + e[n] \cdot dt, \tag{1.10}$$

while the derivative is replaced by a difference

$$e_d[n] = \frac{e_i[n-1] - e[n]}{dt}. \tag{1.11}$$

For the GUI a common framework should be used. In Python Tk and wxwidgets are common ways to build a GUI. Another viable option is PyQt, which, as the name suggests, is a Python port of the Qt framework. One major advantage of using PyQt is the possibility to import `.ui` files describing the GUI directly from the Qt RAD designer called "Qt designer'', removing the need to create the GUI programmatically. Furthermore using PyQt enables the usage of *pyqtgraph*, a highspeed plotting library only compatible with Qt. This ensures plotting live data does not bottleneck performance, which is often the problem with naive *matplotlib* based solutions.

To log all relevant data from RAM to non-volatile memory (hard drive or network share), a simple approach with a line by line CSV writer is used.

1.7 Control Parameter Tuning and Tests

To tune control parameters there are a multiple of analytical, empirical or hybrid approaches. Here the Ziegler-Nichols method is tested and fine tuning is done by hand.

1.8 Improving the Control System

1.8.1 Adding Disturbance Feed Forward

1.9 Results

To evaluate the performance of the control system, 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 1.8. Note, that in this test about 7 h into the experiment there was an unintentional shutdown of FLUTE and the corresponding block of data is removed before further processing.

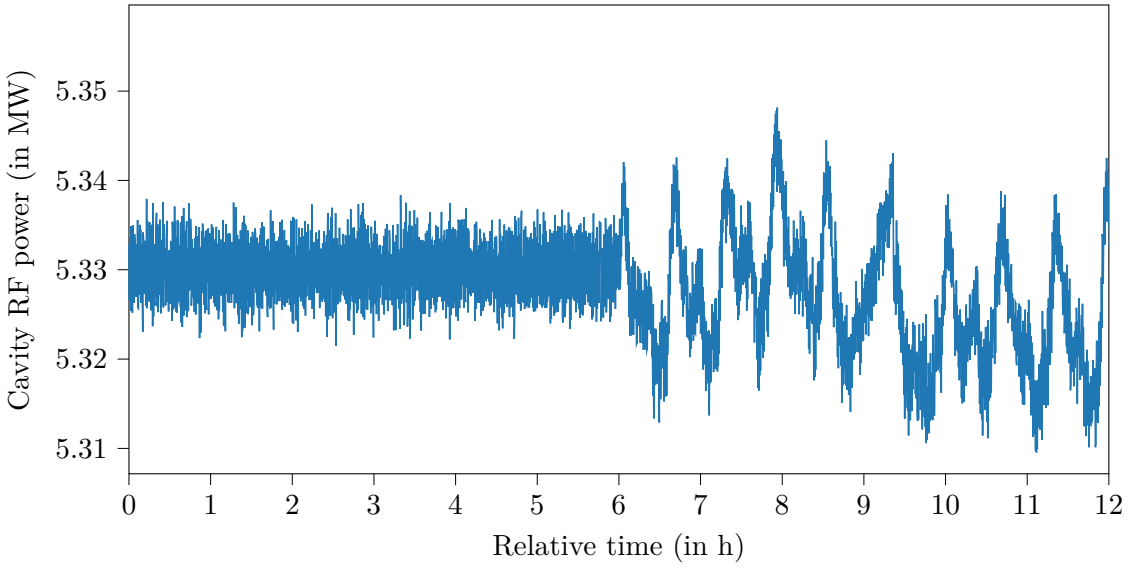


Figure 1.8: Cavity power over about 15 h (about three hours of downtime removed for clarity around the 7 h mark); control system switched off at 6 h (recording started 2021/05/01 20:00)

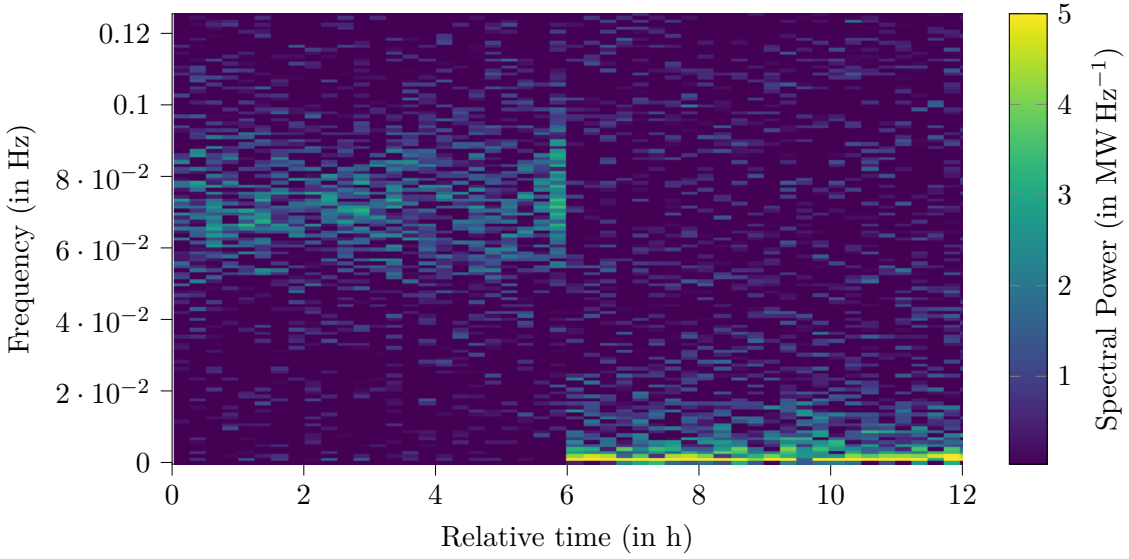


Figure 1.9: Spectrogram of the cavity power in Figure 1.8

The time plot and also the spectogram in Figure 1.9 shows the cavity RF power approximately reaches stationarity in [0, 2]hour respectivaley [6, 12]hour. This allows the spectrum for these two blocks to be estimated using a periodogram method. In Figure 1.9 the spectra for each case are shown.

With the metrics from **??**, the success of the control system is assessed in Table 1.1.

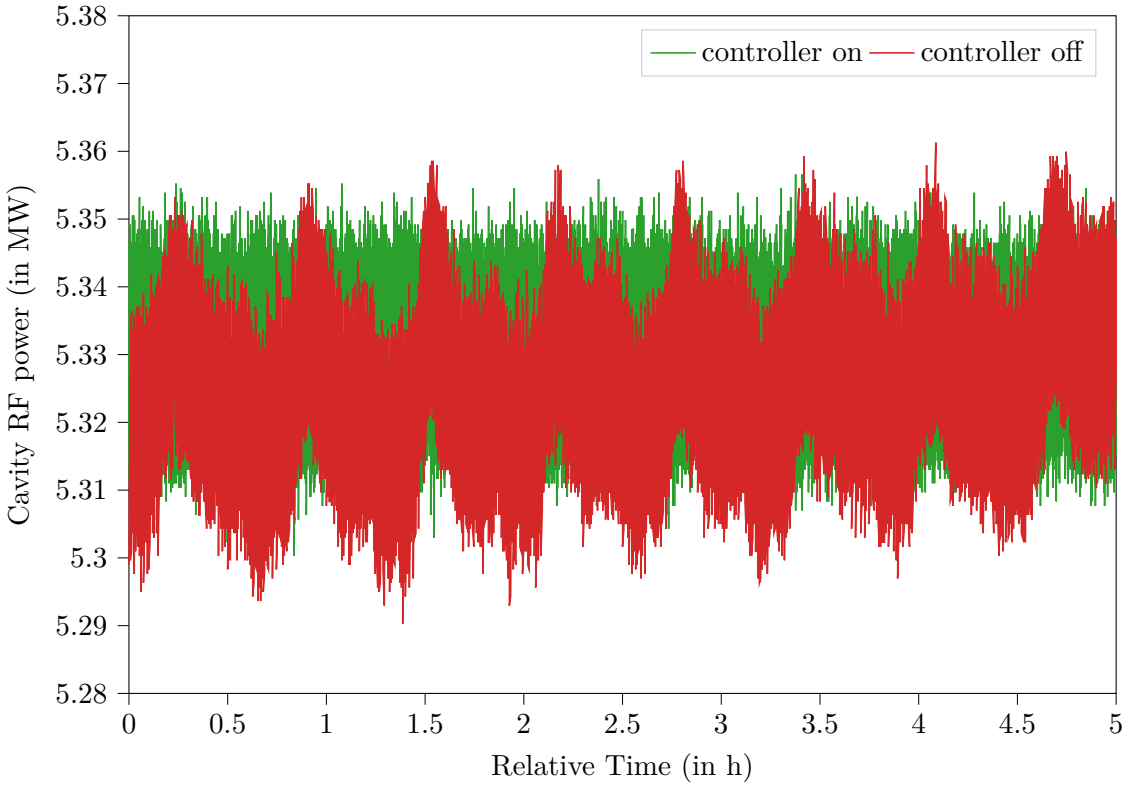
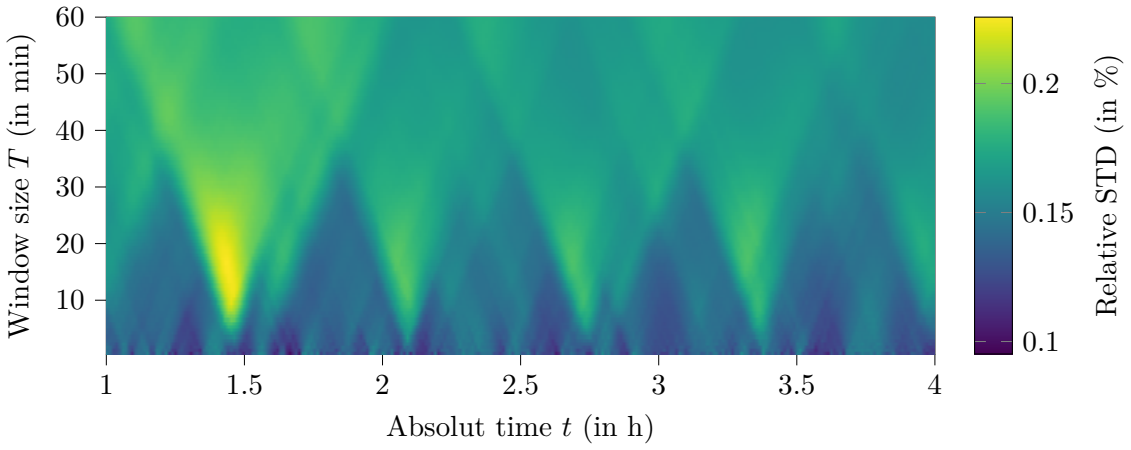


Figure 1.10: Comparison between the control system on and off

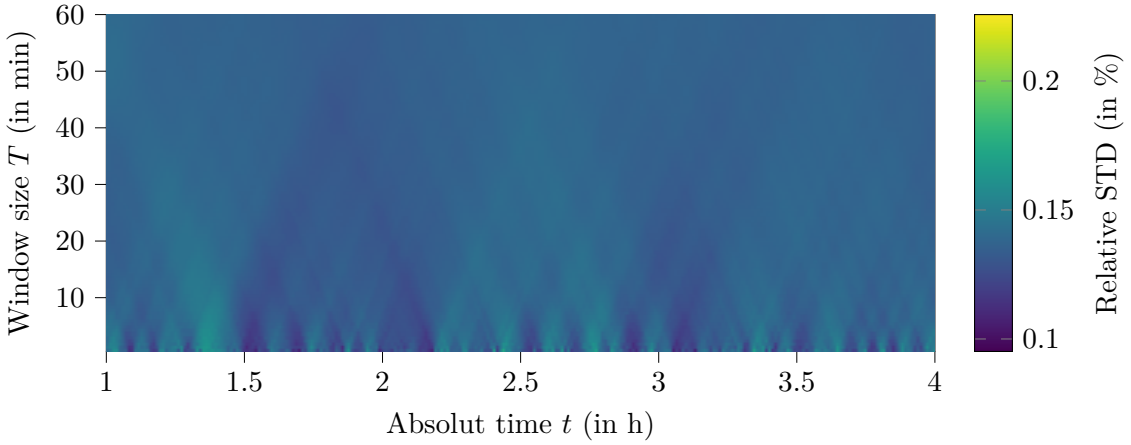
This shows for example the mean squared error is improved by a factor of about 371 by using the control system.

Table 1.1: Quantitative assessment of the controllers performance

Metric	Controller off	Controller on	Controller off/Controller on
$\%STD$	0.001 155 9	$5.992\,25 \times 10^{-5}$	9.2891
MSE	37.639	0.101 33	371.44
MPN	487 309.29	14 386.25	33.873



(a) Controller off



(b) Controller on

Figure 1.11: Relative standard deviation $STD\%(t, T)$

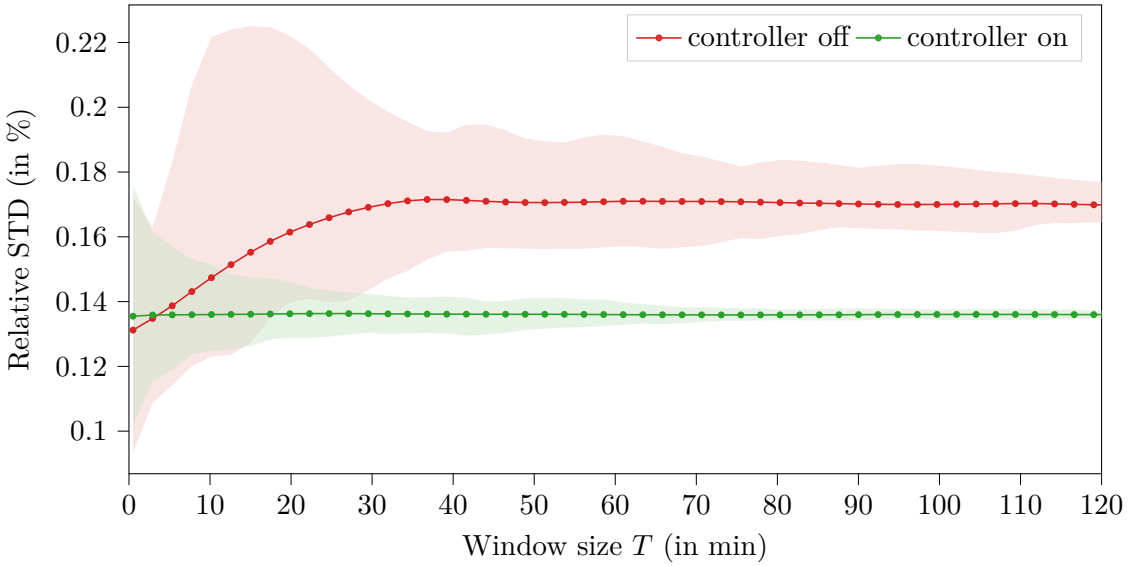


Figure 1.12: Relative standard deviation $STD\%(T)$, shaded areas show $\min\{STD\%(t, T_0)\}$ and $\max\{STD\%(t, T_0)\}$, solid lines show $\text{mean}\{STD\%(t, T_0)\}$ for a fixed window size $T = T_0$

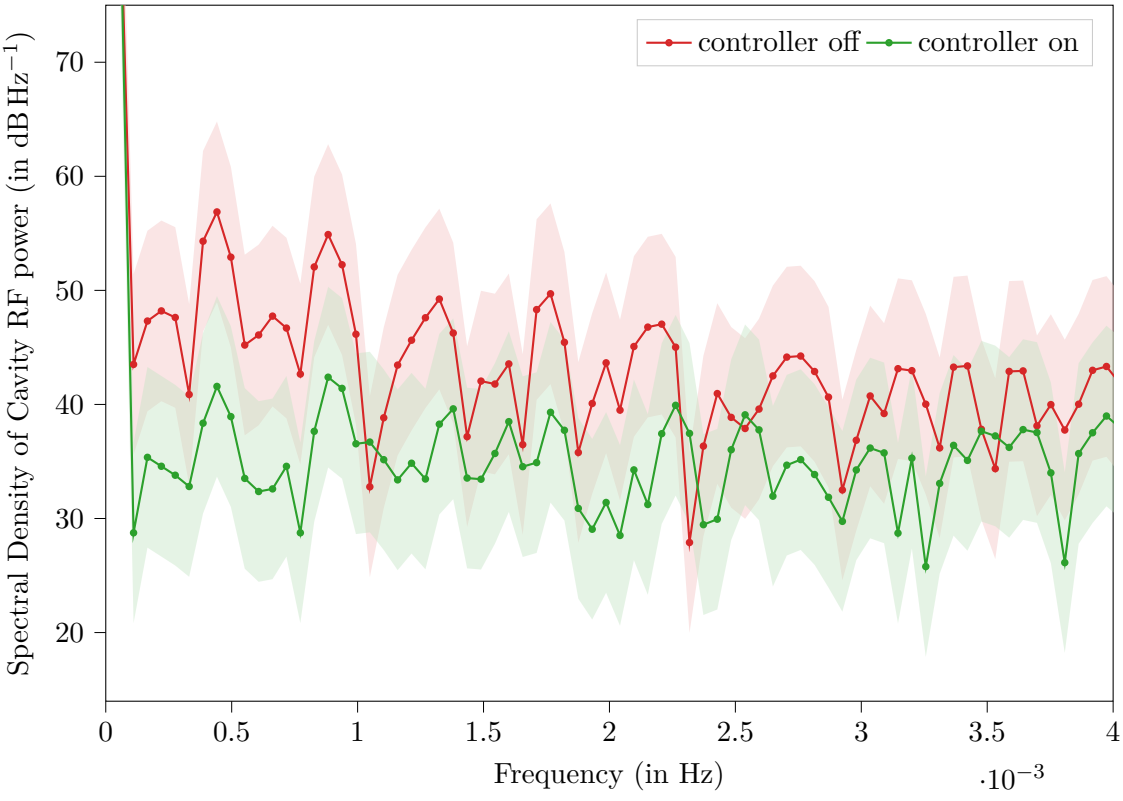


Figure 1.13: Power spectrum of the plots in Figure 1.8 computed with Welch's method; shaded areas show the uncertainty according to **??**

Appendix

A Lab Test and Measurement Equipment

A.1 Benchtop multimeters

A.1.1 Agilent 34411A

Table A.2: Agilent 34411A specifications

Specification	Value
	DC volt
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 A.3: Agilent 34411A some SCPI commands

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

A.1.2 Keysight 34470A

Table A.4: Keysight 34470A specifications

Specification	Value
	DC volt
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 A.5: Keysight 34470A some SCPI commands

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

A.2 Data Acquisition/Switch Unit

A.2.1 Keysight 34972A

Table A.6: Keysight 34972A specifications

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

Table A.7: Keysight 34972A some 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)`	

A.3 Oscilloscopes

A.3.1 Tektronix MSO64

Table A.8: Tektronix MSO64 specifications

Specification	Value
Bandwidth	6 GHz
Sample rate	25 GS/s
ADC resolution	12 bit
DC gain accuracy (@ 50 Ω, >2 mV/div)	±2 %

Table A.9: Tektronix MSO64 some SCPI commands

Description	Example command	Example return
Read mean of measurement 1 (current acq.)	`MEASUrement:MEAS1:RESUlts:CURR:MEAN?`	`3.0685821787408`

A.4 RF signal generator

A.4.1 Rohde and Schwarz SMC100A

Table A.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, $f_o = 20$ kHz, $BW = 1$ Hz)	−111 dBc
Level error	<0.9 dB

Table A.11: Rohde and Schwarz SMC100A some 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`	

A.5 RF power meter

A.5.1 HP E4419B

Table A.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 A.13: HP E4419B some SCPI commands

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

A.6 Vector Network Analyzer

A.6.1 Agilent E5071C

Table A.14: Agilent E5071C specifications

Specification	Value
Frequency range	9 kHz to 8.5 GHz

A.7 Phase noise analyzer

A.7.1 Holzworth HA7062C

Table A.15: Holzworth HA7062C specifications

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

Acknowledgments

A

Bibliography

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

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