Karlsruhe Institute of Technology

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
Second Reviewer:	Prof. Dr.-Ing. Thomas Zwick
Advisor:	Dr. Nigel Smale

15.11.2020 – 17.05.2021

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 17.05.2021, _____

Marvin-Dennis Noll

Als Prüfungsexemplar genehmigt von

Karlsruhe, den 17.05.2021, _____

Prof. Dr.-Ing. John Jelonnek

Contents

List of Figures

List of Tables

1. Own Work - Control System

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 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. 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

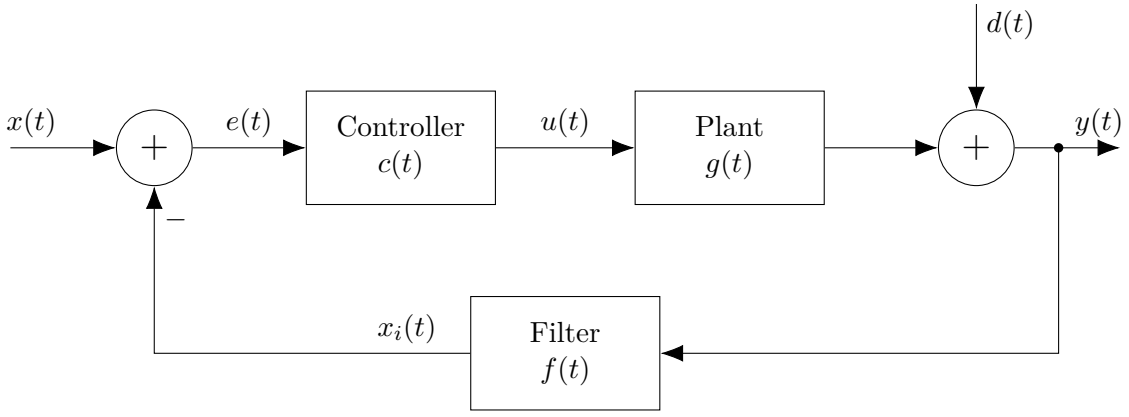

Figure 1.1.: General structure of a closed loop feedback control system

1.2. Interfacing with FLUTE: 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 $u(t)$.

1.2.1. Input

Most signals to be useful as controlled variable ($y(t)$ and its set point $x(t)$) are available in EPICS. Retrieving their current value is therefore trivial if the control PC is on the machine network.

For example using the command line tool `caget` the last value of a PV can be obtained independent of the programming language used, however a syscall is needed which comes with a (slight) overhead and is inconvenient to use.

At least for Python and C++ there are EPICS libraries directly providing useful EPICS commands. A Python example of how to get the current cavity power is shown in page 2.

Listing 1.1: Get an EPICS PV with python

```
1 from epics import caget
2 cavity_power=caget("F:RF:LLRF:01:GunCav1:Power:Out")
```

1.2.2. 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).

1.3. Plant Identification

1.3.1. 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.

1.3.2. 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).

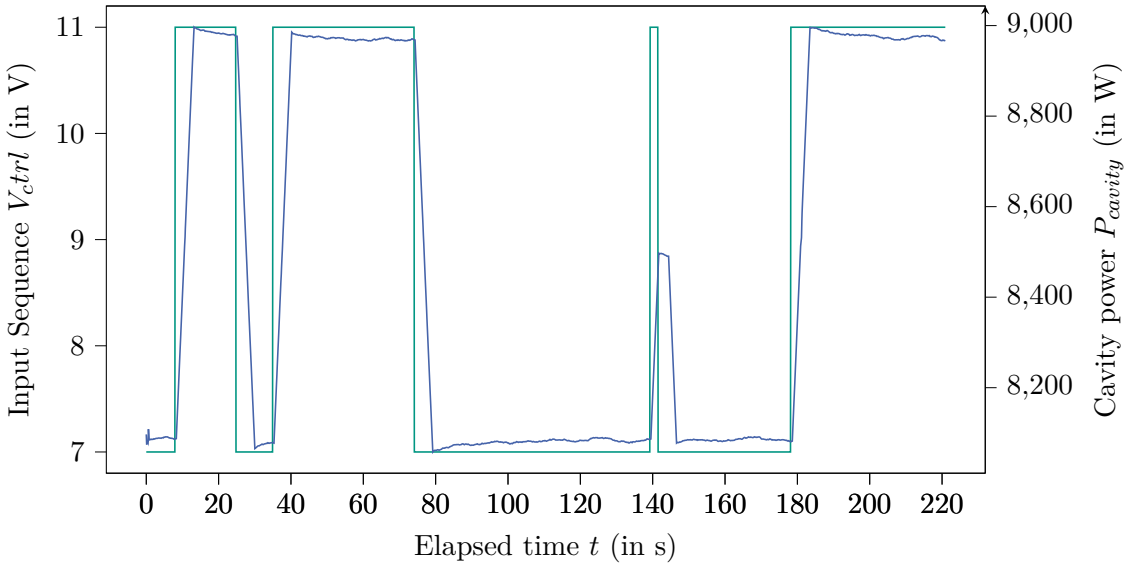

Figure 1.2.: Small section of the input sequence (green) and the system response (blue)

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 1.2. With the parameter `toggleP`, the average length of one constant voltage level can be controlled.

Listing 1.2: 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 page 3.

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

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

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

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

emerge.

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

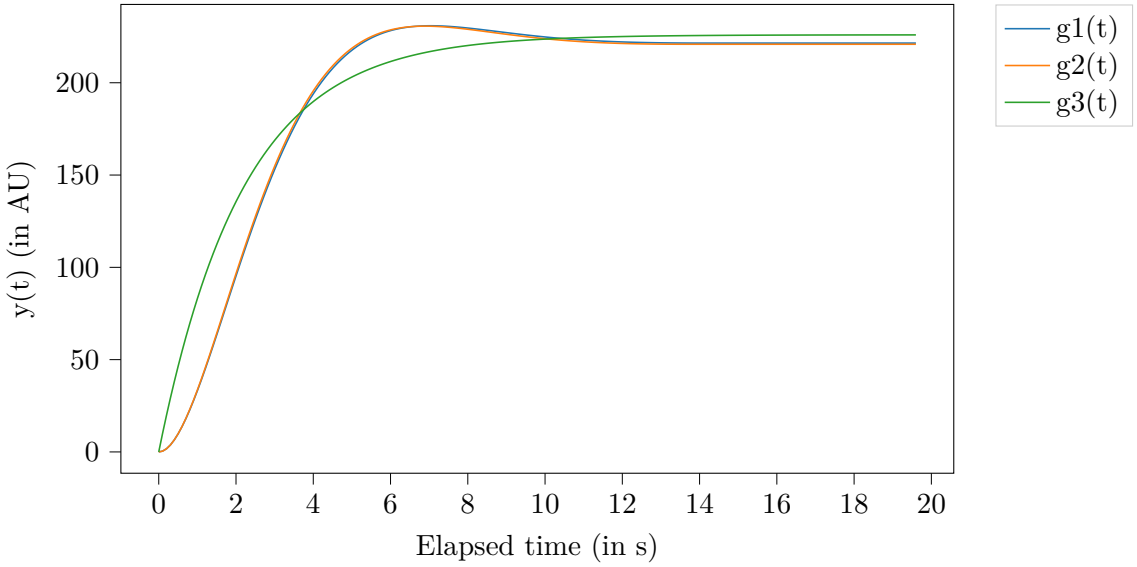

Figure 1.3.: Step responses of the systems $\hat{G}_1(s)$, $\hat{G}_2(s)$, $\hat{G}_3(s)$

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.

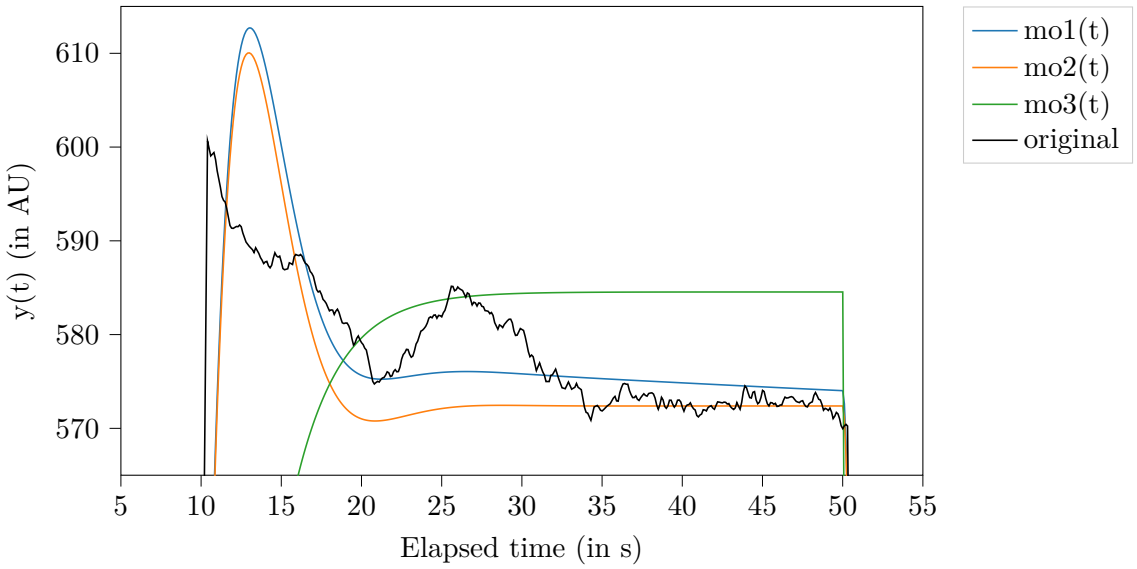

Figure 1.4.: Validating the estimations $\hat{G}_1(s)$, $\hat{G}_2(s)$, $\hat{G}_3(s)$ against real data from the validation data set

1.4. 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 1.5).

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.

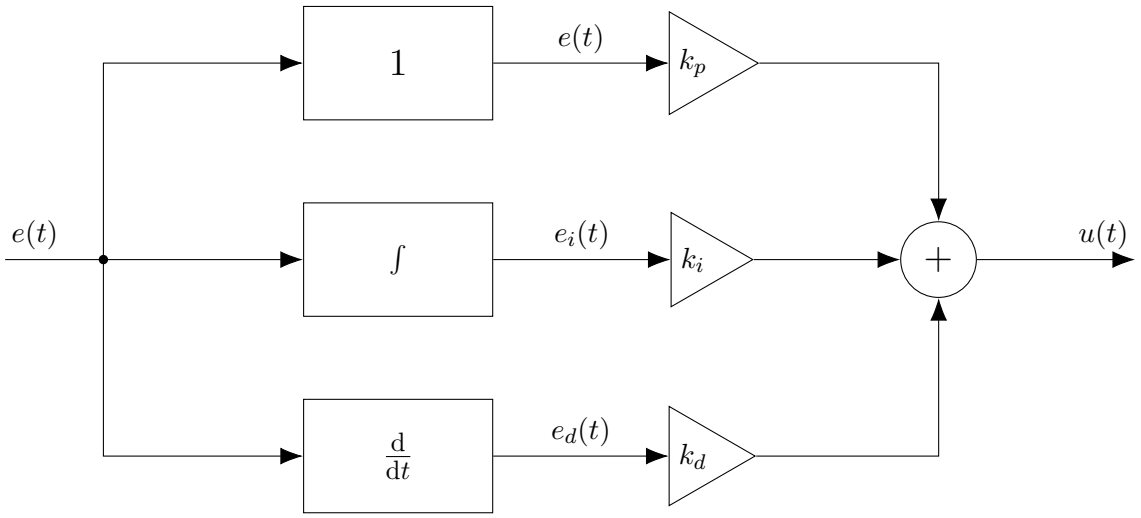

Figure 1.5.: Block diagram of a generic PID controller

1.5. 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

$$e_i[n] = e_i[n-1] + e[n] \cdot dt, \tag{1.5}$$

while the derivative is replaced by a difference

$$e_d[n] = \frac{e_i[n-1] - e[n]}{dt}. \tag{1.6}$$

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.

1.6. 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.

1.7. Results

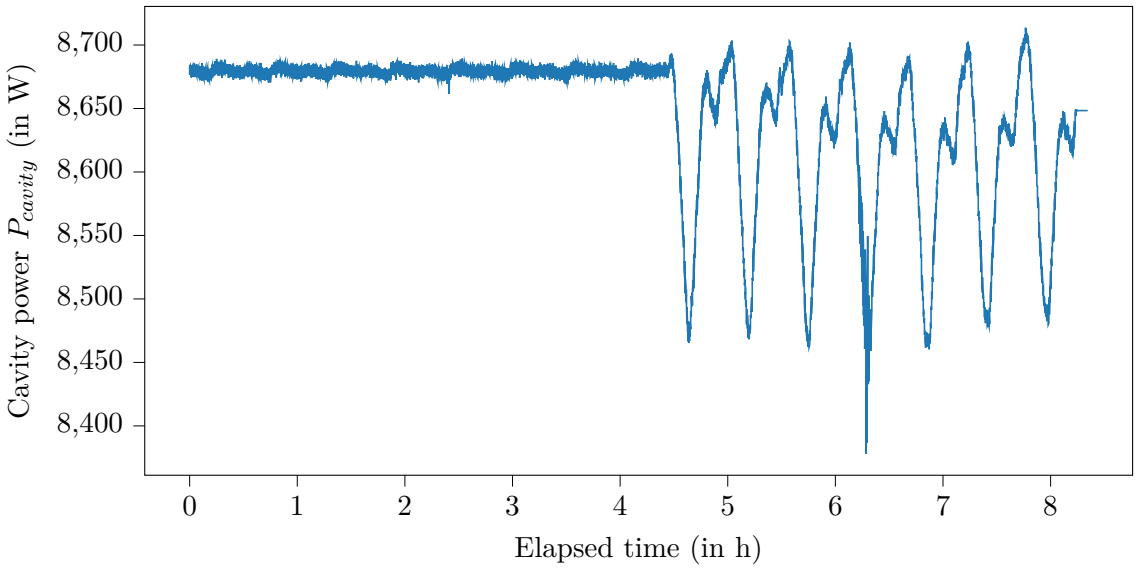

Figure 1.6.: Cavity power with PID controller on (before the 4.5 h mark) and off shows the stabilizing effect of the control system

1.8. Summary and Improvement Suggestions

Appendix

A. Lab Test and Measurement Equipment

A.1. Benchtop multimeters

A.1.1. Agilent 34411A

Table A.1.: 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.2.: 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.3.: 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.4.: 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.5.: 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.6.: 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.7.: 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.8.: 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.9.: 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.10.: 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.11.: 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.12.: 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.13.: 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.14.: Holzworth HA7062C specifications

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