Artem Moskalew
Jonas Christ
Maximilian Nolte

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Inhaltsverzeichnis

1 Einführung

Mit den Barrier Bucket (BB) RF Systemen können am neu entstehenden Synchrotron SIS100 oder im Experimentier Speicherring (ESR) am GSI Helmholtzzentrum viele longitudinale Manipulationen am Teilchenstrahl vorgenommen werden. Der dazu notwendige Spannungspuls hat die Form wie in Abb. (1.1) dargestellt. Wenn die Wiederholfrequenz des Spannungspulses gleich der Umlauffrequenz ist, so wird im Phasenraum eine stationäre Potential Barriere erstellt. Wenn die Wiederholfrequenz nicht gleich der Umlauffrequenz ist verschiebt sich die Potential Barriere im Phasenraum und es entstehen Bunches mit unterschiedlicher Länge.
Der Anspruch an diese Systeme liegt darin eine hohe Qualität des Impulses am Gap der Kavität zu erzeugen, damit sogenannte Microbunches unerwünscht entstehen, deshalb müssen die Nachschwinger nach dem Einzelsinus kleiner als 2,5% von $\hat{U}_{BB}$ sein.

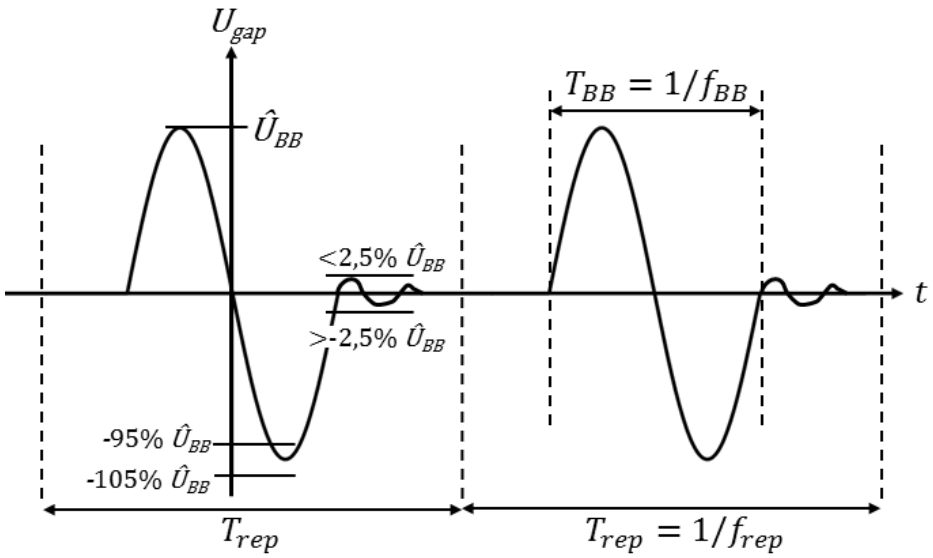

Abbildung 1.1: Ausgangssignal

Dabei benutzen wir im Folgenden f_{rep} für die Wiederholfrequenz des Spannungssignals und die Barrier Bucket Frequenz f_{BB} bestimmt die Breite der Potential Barriere. Für unsere Messungen haben wir $f_{rep} = 900\text{kHz}$ und $f_{BB} = 5\text{MHz}$ verwendet.

1.1 Versuchsaufbau und Modell

Der Prototyp des ESR BB besteht aus einem Funktionsgenerator (Keysight 3600A series 2-channel AWG), einem Verstärker (AR1000A225) und der Breitband Ringkern Kavität. Der Versuchsaufbau ist in Abb. (1.2) gezeigt.
Dabei kann angenommen werden, dass sich das System bis $\hat{U}_{BB} = 550V$ annähernd linear verhält und durch die Übertragungsfunktion $\underline{H}$ beschrieben werden kann. Eine mathematische Modellierung ist ebenfalls in Abb. (1.2) gegeben, bei der zur linearen Übertragungsfunktion $\underline{H}$ noch eine nichtlineare Kennlinie enthalten ist. Die Größen werden im nächsten Abschnitt erklärt.

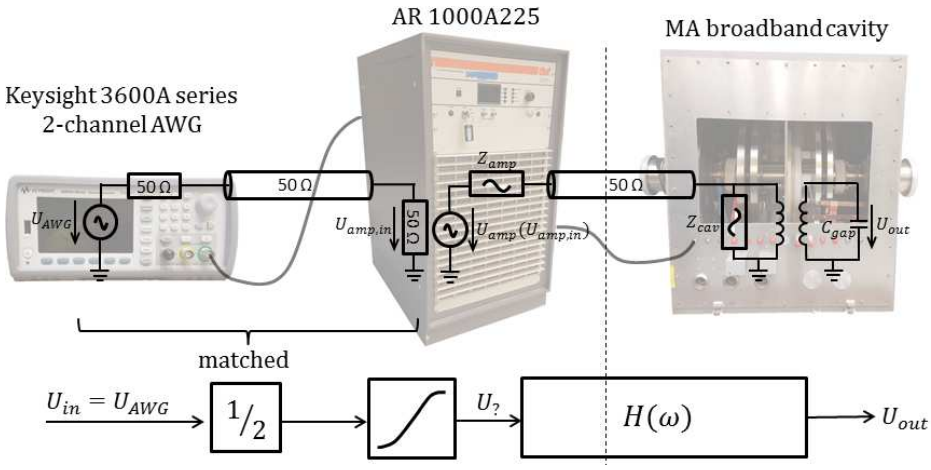

Abbildung 1.2: Versuchsaufbau und Modell

1.1.1 Hammerstein Modell

In Abb. (1.3) sind alle von uns verwendeten Größen dargestellt. U_{in} ist die Eingangsspannung vom Funktionsgenerator. Der erste Block in Abb. (1.3) stellt die nichtlineare Modellierung dar. Die Spannung $U_?$ ist eine rein virtuelle Größe und kann wie in Glg. (1.1) über die Koeffizienten a_n berechnet werden. Bei der Potenzreihe hatte $N = 3$ in der MATLAB Implementierung schon gute Ergebnisse geliefert und wurde von uns auch weiterhin so verwendet. Die nichtlinearen Kennlinie im Folgenden nur noch als K bezeichnet wird als Look-Up Tabelle in unserem Programm hinterlegt. Die Spannung U_{out} ist die Gap Spannung in der Kavität.

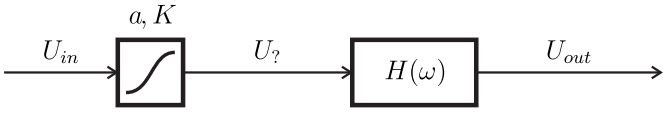

Abbildung 1.3: Hammerstein Modell

$$U_?(t) = \sum_{n=1}^{N} a_n \left[U_{in}(t)\right]^n \qquad U_{out}(t) = \mathscr{F}^{-1}\left\{\underline{H}(\omega) \cdot \underline{U}_{in}(\omega)\right\} \tag{1.1}$$

1.2 Motivation

Dieses Modell wurde bis auf die Berechnung von K schon erfolgreich in Python implementiert, dabei sei auf das Projektseminar [2] verwiesen. Deshalb lag unsere Motivation darin dieses Modell in Python zu vervollständigen, um dann einen möglichen Optimierungsansatz aufzustellen.

1.3 Aufgabenstellung

Im Rahmen unseres Projektseminar sollte eine iterative Optimierung der Vorverzerrung auf Basis einer Hammersteinmodellierung in Python implementiert werden. Dabei sollte das Tool die nichtlineare Kennlinie K und die Übertragungsfunktion $\underline{H}$ wechselseitig optimieren, wobei eine optimale Gewichtung der Parameter zur Optimierung noch nicht erreicht werden musste.

2 Vorgehen

In diesem Kapitel werden die wichtigsten Punkte in der Weiterentwicklung des vorhandenen Tools erklärt. Dabei wird unsere Methodik vorgestellt an den vorhandenen Weg anzuschließen und weiter fortzusetzen im Hinblick darauf, dass an das Projekt von anderen Personen angesetzt werden kann.

2.1 Vorhandene Implementierung

Unser Projekt setzt an 2 Punkten an. Zum einen das Projektseminar [2], in dem der Grundstein in Python gelegt wurde und zum anderen eine funktionierende Implementierung de nichtlinearen Bestandteils des Modells aus Abb. (1.3) nach der Vorstellung aus [3].

2.1.1 Python

Das vorhandene Python Tool konnte mit einem Pseudorauschen die Übertragungsfunktion des Systems im linearen Bereich bestimmt werden. Diese Funktionalität haben wir vollständig übernommen. Darüber hinaus waren folgende Funktionalitäten implementiert:

- $\underline{H}$ auf das Spektrum von $U_{out,ideal}$ anzuwenden.
- Signale an das AWG zu senden und vom Oszilloskop zu messen.
- aus gegebenen Koeffizienten a_n eine Kennlinie als Look-Up Tabelle zu berechnen.

2.1.2 MATLAB

Als zweiten Ansatzpunkt hatten wir funktionierende Methoden aus der MATLAB Implementierung des nichtlinearen Teils mit folgenden Funktionalitäten:

- $\underline{H}$ auf das Spektrum von $U_{out,ideal}$ anzuwenden.
- a_n für die nichtlineare Kennlinie zu berechnen.
- K als Look-Up Tabelle aufzustellen.
- ein vorverzerrtes Eingangssignal über K zu berechnen.

2.2 Funktionalität reproduzieren

Als ersten wurde versucht die vorhandene Implementierung zu überblicken und letztendlich beide in Python zusammenzuführen. Dabei wurde eine starke Kopplung festgestellt, was die Fehlersuche und Fortführung erschwerte. Ein paar Funktionen hatten teilweise auch mehrere logische Aufgaben, die für eine bessere Verständlichkeit getrennt werden sollten. Mit dem Code wurden schon gute Ergebnisse erzielt und das erste Ziel war, den Code in eine neue Form zu überführen, ohne die Funktionalität zu beeinträchtigen.

2.2.1 Neues Design

Das in Abb. (1.3) dargestellte Modell wird in zwei Blöcke gegliedert und so wurde ein Code-Design gewählt, das diese Blöcke repräsentiert. Dafür bietet sich ein modulares Design an, das in Kap. 3 genauer erklärt ist und dabei möglichst selbsterklärend dieses Modell abbildet.

2.2.2 Tests

2.2.3 Fehlersuche

2.3 Funktionalität erweitern

2.3.1 Erste Optimierungs Idee

2.3.2 Laufzeit Optimierung

2.4 Gerätekommunikation

Um Messergebnisse sinnvoll für die Verarbeitung in Python aufzunehmen, ist eine effiziente Kommunikation zwischen den Messgeräten und dem angeschlossenen Computer essentiell. Besonders relevant ist hier die zeitliche Abstimmung zwischen Computer und Gerät sowie die bestmögliche Nutzung der Genauigkeiten der verwendeten Geräte. Hierbei wurden die zu Beginn vorliegenden Implementierungen für die Gerätekommunikation erweitert, wobei vor allem

1. die Optimierung der Laufzeit beim Schreiben und Lesen von Gerätewerten sowie
2. die Anpassung der Darstellung des Oszilloskops an die gemessenen Daten für eine höhere Genauigkeit

Ziele der Anpassungen waren.

Die Fernsteuerung von Messgeräten, der `Remote`-Modus, wird standardmäßig durch das Visa-Protokoll und im vorliegenden Fall durch die Implementierung von National Instruments, NIVisa, geregelt. Die Einbindung in Python wird durch die Erweiterung PyVisa ermöglicht. Übertragen werden Befehle in Form standardisierter Befehlsstrukturen, den SCPI (Standard Commands for Programmable Instruments). Dem inneren Aufbau von Messgeräten zugrunde liegt der IEE-488 Standard, sodass unabhängig vom konkreten Gerät oder der Art der Verbindung eine Reihe von Befehlen existiert, die gemäß IEE-488.2 gebräuchliche Standards bieten [5, S. 224 ff.]. Weiterhin bieten die Geräte spezifische Befehle zur Manipulation der Einstellungen, die in großer Analogie zur direkten Benutzeroberfläche formuliert sind. Diese werden nach Kontext gegliedert in sogenannte Subsysteme oder Command Groups.

Laufzeitoptimierung

Im vorliegenden Messaufbau ist es notwendig, beim Konfigurieren des AWG alle Einstellungen zurückzusetzen und anschließend neu zu setzen. Dabei werden Befehle nach dem First-Come-First-Serve Prinzip aus dem internen Speicher abgearbeitet. Dies führt aufgrund begrenzter Speicherkapazität und Verarbeitungsgeschwindigkeit bei einer Reihe von Befehlen zur Notwendigkeit, das Senden von Befehlen im Programm mit dem Arbeitsstatus des Gerätes zu synchronisieren. Einen Eindruck der zeitlichen Größenordnung bietet Tab. (2.1).

In der zu Beginn vorliegenden Implementierung lagen pauschale Wartezeiten nach einigen Befehlen sowohl in der Ansteuerung des AWG als auch des Oszilloskops vor.

Die Wartezeiten für das AWG wurde letztendlich mit ———————————— ———— ———————————— angepasst. Die Laufzeit der Routine zum Schreiben eines Arbiträrsignals in

(a) Knotenliste

$P_\#$	x	y
1	$\cos(72°)$	$\sin(72°)$
2	1	0
3	$\cos(72°)$	$-\sin(72°)$
4	$-\cos(36°)$	$-\sin(36°)$
5	$-\cos(36°)$	$\sin(36°)$
6	0	0

Tabelle 2.1: Laufzeit unterschiedlicher Befehle am Keysight 33622A

der speziellen hier geforderten Form wurde damit von etwa 25 s auf ungefähr — reduziert.
Erfolglos blieben Versuche mit einigen anderen Standardbefehlen. `*WAI` und `*BUSY?` sind keine für das AWG definierten Befehle. `*OPC?` zwingt Python zum Warten auf Beenden aller Befehle im AWG und führt zu einem `TimeoutError`, da die Ausführung zu lange für die internen Routinen von Python braucht, vergleiche hierfür [7, S. 9]. Weiterhin wurde die pauschale Wartezeit an der im Datenblatt [6, S. 21] für die Verbindung via USB angegebenen, gemessenen Ladedauer eines Arbiträrsignals orientiert. Dieser Richtwert von 1.25 s stellte sich jedoch als zu gering für den hier vorliegenden Fall heraus.

Es wurde festgestellt, das die Verarbeitungsgeschwindigkeit des Oszilloskops bei den hier benötigten Abfragen und Befehlen kein Hindernis darstellt und die Ausführung der Befehle in Python ohne Zeitverzögerung möglich ist. Durch diese Feststellung konnte die Laufzeit je Lese-Aufruf bereits um ungefähr 15 s reduziert werden. Bei den zur Sicherheit implementierten Warte-Befehlen handelt es sich um Status-Abfragen ähnlich den oben für das AWG erläuterten.

Letztlich sei erwähnt, dass die Abfrage der Fehler im AWG mittels `SYSTem:ERRor?` möglich ist und durch das dabei erfolgte Löschen der Fehler aus dem internen Speicher nicht nur die Abfrage sondern auch das Rücksetzen des Fehlerstatus ermöglicht. Dies ist insbesondere für das Debugging vorteilhaft, wenn Befehle ausprobiert werden und die gegebenenfalls auftretenden Fehler gehandhabt werden müssen [5, S. 454]. Dies vermeidet das unter Umständen mehrmalige Trennen und Wiederherstellen der Verbindung zum AWG sowie das Löschen der Fehler an der Benutzeroberfläche.

Anpassung der horizontalen Auflösung am Oszilloskop

Um die Auflösung des Oszilloskops ausnutzen zu können, ist die Anpassung der horizontalen wie auch vertikalen Auflösung an das zu messende Signal notwendig. Während die horizontale Auflösung durch die Wiederholfrequenz des Pulses mit f_{BB} sich nahezu unabhängig des betrachteten Signals setzen lässt, muss die vertikale Skalierung an das Signal angepasst werden. Um dies etwa während einer Optimierung mit wechselnden Spannungsamplituden zu ermöglichen, sind zwei direkt aufeinander folgende Messungen notwendig. Dabei wird aus der ersten Messung die Amplitude des Signals entnommen und die Skalierung für die zweite, dann genauere Messung, dahingehend angepasst, dass das Signal bildfüllender dargestellt wird. Dieser Gewinn an Genauigkeit in der Messung stellte sich insbesondere für die ?? als notwendig heraus.

2.5 Kritische Punkte

Infolge der Auseinandersetzung mit dem anfangs vorliegenden Code sowie im Zuge der Ausarbeitung sind einige Punkte aufgetreten, an deren Bedeutung die implementierte Funktionalität teilweise essen-

tiell hing, die jedoch im Zuge der Implementierung für Probleme sorgten. Diese seien aufgrund ihrer unterschiedlichen Natur hier stichpunktartig erläutert:

- Bei diskreten Zeitsignalen ist zu beachten, dass die Periodendauer T nicht durch die Differenz der Zeitwerte des letzten und des ersten Wertes, sondern noch zuzüglich eines Zeitschritts zu berechnen ist, also `T = time[end] - time[0] + delta_time`, womit gelten sollte `T = N * delta_time`. Dies wirkt sich insbesondere auch auf die FFT aus, wenn die zugehörige Frequenzachse berechnet werden muss.

3 Code-Design

Vorstellung Namenskonvention und

3.1 Motivation

Waraum wurde das Design von uns so gewählt?

3.2 Aufbau

3.2.1 Namenskonvention

3.2.2 Ordnerstruktur

3.2.3 Abstact Data Types

3.3 Methodik

3.3.1 Test Driven Development

3.3.2 Mock-System

Literaturverzeichnis

[1] Leslie Lamport, LaTeX: a document preparation system, Addison Wesley, Massachusetts, 2nd edition, 1994.

[2] Denys Bast, Armin Galetzka, Projektseminar Beschleunigertechnik, 2017.

[3] Jens Harzheim et al., Input Signal Generation For Barrier Bucket RF Systems At GSI, In Proceedings of IPAC2017 in Copenhagen, Dänemark. pp. 3948 - 3950 2017.

[4] Kerstin Gross et al., Test Setup For Automated Barrier Bucket Signal Generation, In Proceedings of IPAC2017 in Copenhagen, Dänemark. pp. 3948 - 3950 2017.

[5] Keysight Technologies, Keysight Trueform Series Operating and Service Guide, 2015.

[6] Keysight Technologies, 33600A Series Trueform Waveform Generators - Data Sheet, 2014.

[7] C. Tröser, Application Note: Top Ten SCPI Programming Tips for Signal Generators, Rohde & Schwarz, 2013.

4 — Anhang —