Generierung des Eingangssingals für Barrier Bucket RF Systeme and der GSI

TECHNISCHE UNIVERSITÄT DARMSTADT

Jonas Christ, Artem Moskalew, Maximilian Nolte
Jens Harzheim, M.Sc.

Projektseminar Beschleunigertechnik

Outline

Problemstellung

- Barrier-Bucket System

Problemstellung

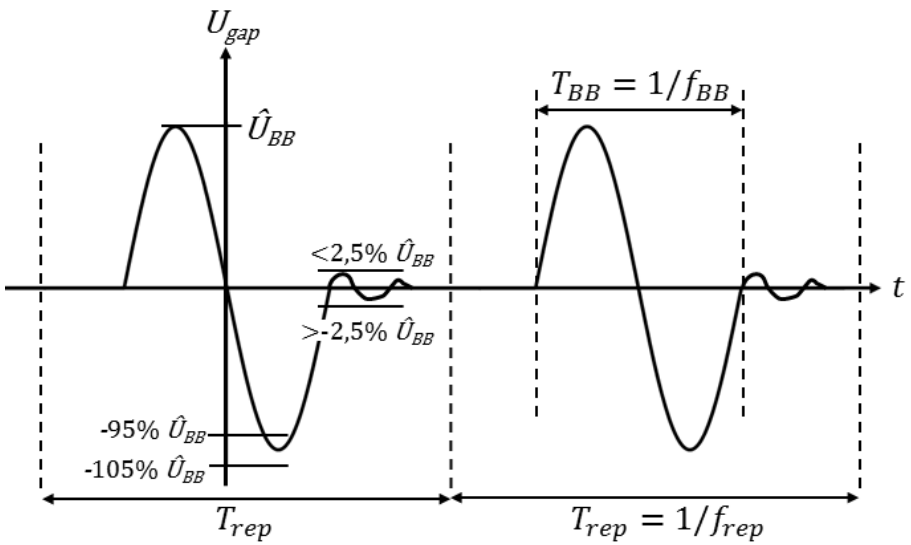
- Barrier-Bucket System :
  - Longitudinale Manipulation der Bunches

Problemstellung

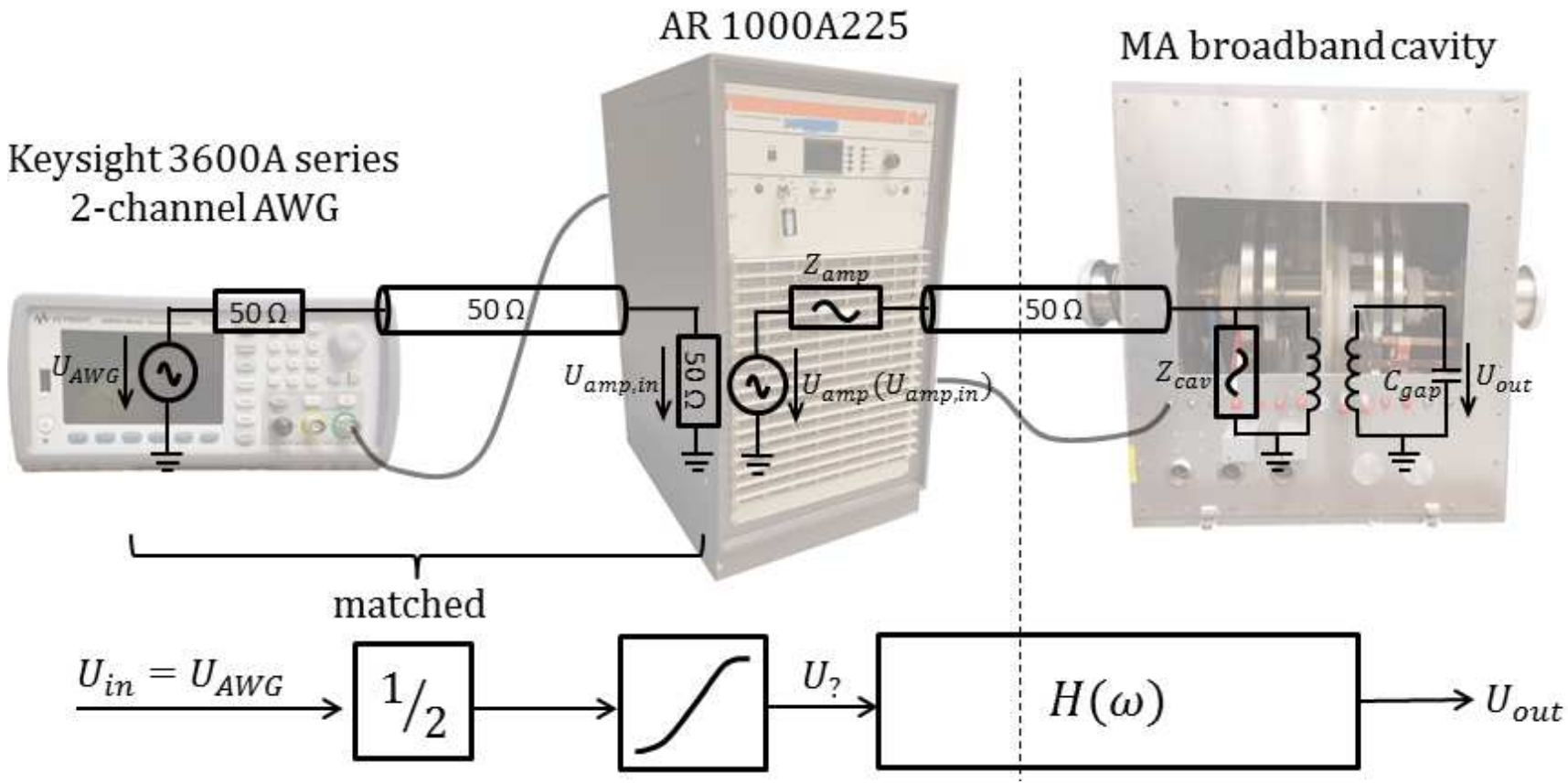
- Barrier-Bucket System :
  - Longitudinale Manipulation der Bunches
- Ziel

Problemstellung

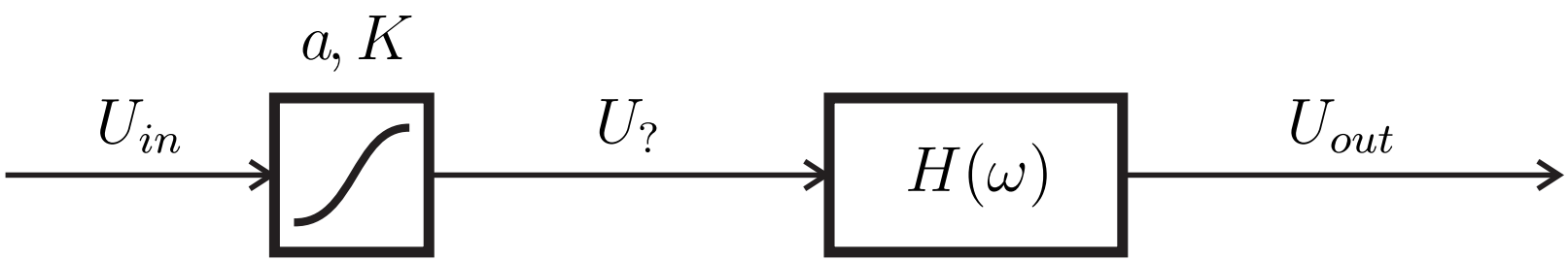
- Barrier-Bucket System :
  - Longitudinale Manipulation der Bunches
- Ziel :
  - Gap Spannung in Form einer Ein-Sinus Periode
  - Qualität das Signals



Aufbau und Modell

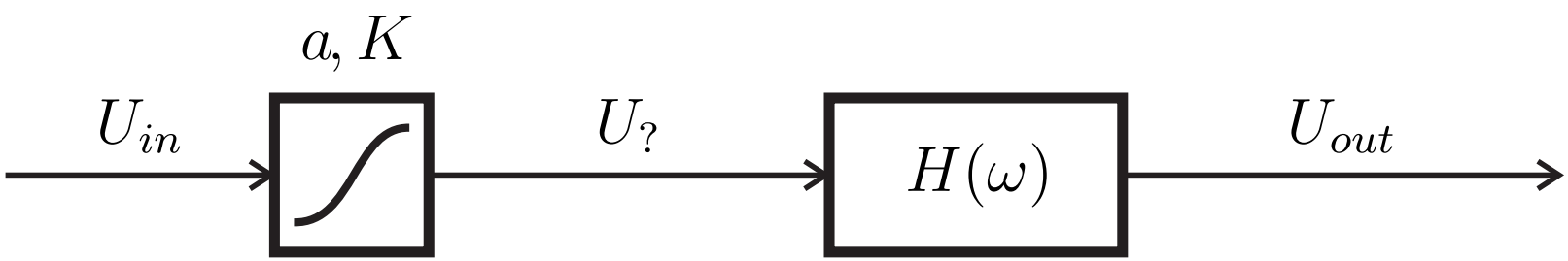


Aufbau und Modell



Aufbau und Modell

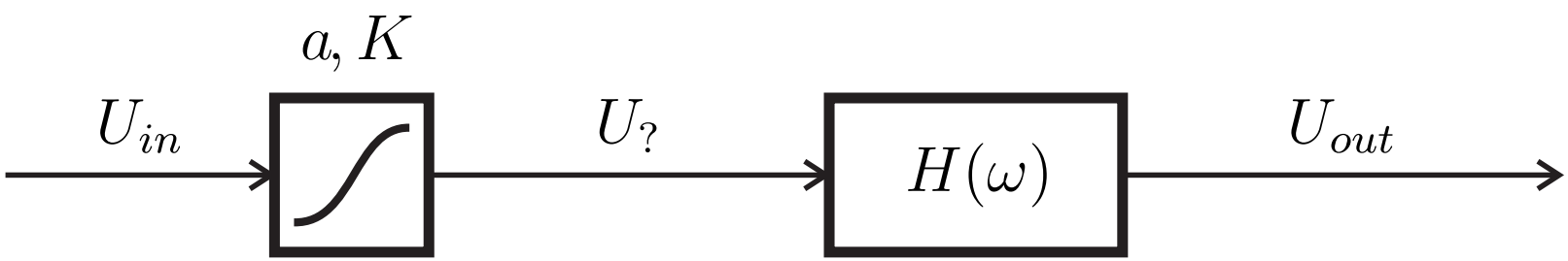
- Gegeben:
  - Lineare Übertragungsfunktion H bestimmt durch Pseudorauschen
  - System linear bis $\hat{U}_{BB} \approx 550\,V$ genähert



Aufbau und Modell

- Gegeben:
  - Lineare Übertragungsfunktion H bestimmt durch Pseudorauschen
  - System linear bis $\hat{U}_{BB} \approx 550\,V$ genähert
- Hammerstein Modell :
  - Ergänzung um eine nichtlineare Vorverzerrung mit einem Potenzreihenansatz

$$U_?(t) = \sum_{n=1}^{N} a_n \, [U_{in}(t)]^n \qquad \underline{U}_{out}(\omega) = H(\omega) \cdot \underline{U}_?(\omega)$$

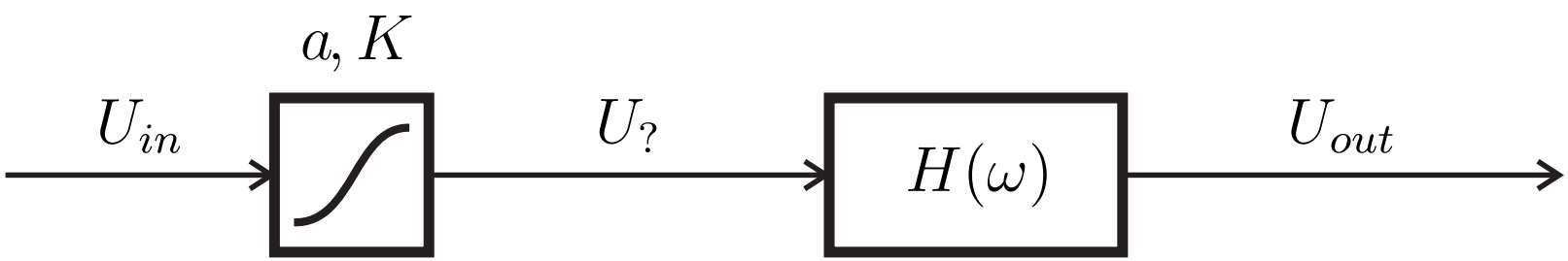


Aufbau und Modell

- Gegeben:
  - Lineare Übertragungsfunktion H bestimmt durch Pseudorauschen
  - System linear bis $\hat{U}_{BB} \approx 550\,V$ genähert
- Hammerstein Modell :
  - Ergänzung um eine nichtlineare Vorverzerrung mit einem Potenzreihenansatz

$$U_?(t) = \sum_{n=1}^{N} a_n \, [U_{in}(t)]^n \qquad \underline{U}_{out}(\omega) = H(\omega) \cdot \underline{U}_?(\omega)$$

- Zielsetzung :
  - Parameter a_n der Kennlinie K zubestimmen



Dokumentation und Gerätekommunikation

- Dokumentation

- Gerätekommunikation

Dokumentation und Gerätekommunikation

- Dokumentation :
  - Handhabung der Geräte, Vorgehensweise bei Tests
  - Bedienung des Programms
  - Ausführliches Kommentieren der Code-Funktionalität
- Gerätekommunikation

Dokumentation und Gerätekommunikation

- Dokumentation :
  - Handhabung der Geräte, Vorgehensweise bei Tests
  - Bedienung des Programms
  - Ausführliches Kommentieren der Code-Funktionalität
- Gerätekommunikation :
  - Treiber und Programmer-Manuals zur Nutzung des Programms von anderen Geräten aus
  - Laufzeitoptimierung durch Abfrage von Gerätezuständen mittels VISA
  - Verbesserung der Auflösung des Signals durch Anpassung der Darstellung des Oszilloskops mittels VISA

Dokumentation und Gerätekommunikation - Evaluierung

Dokumentation und Gerätekommunikation - Evaluierung

- Unvollständige Dokumentation:
  - Übergabeparameter der Funktionen dokumentieren

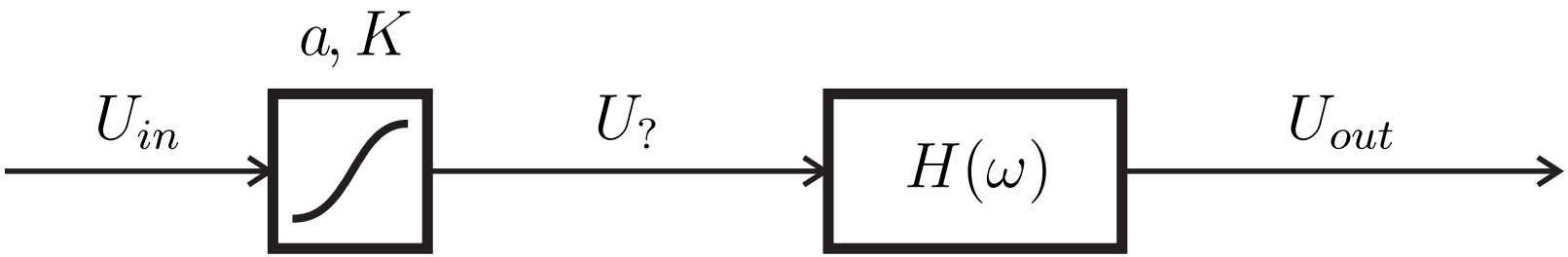
Dokumentation und Gerätekommunikation - Evaluierung

- Unvollständige Dokumentation:
  - Übergabeparameter der Funktionen dokumentieren
- Getestete Teile der Gerätekommunikation:
  - VISA-Protokoll und PyVisa Package Installation
  - Kommunikation mit AWG von anderem Laptop aus über USB

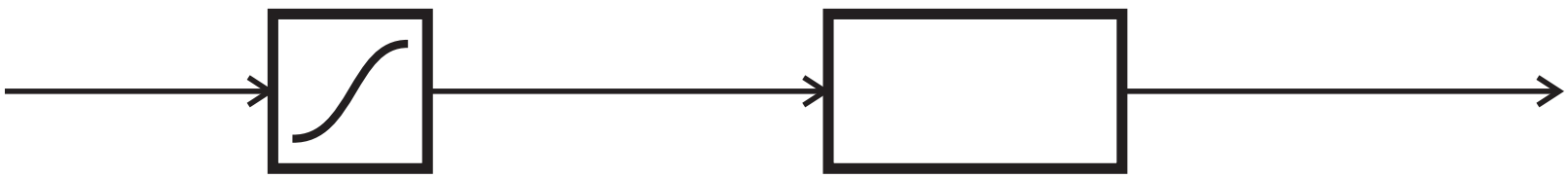
Dokumentation und Gerätekommunikation - Evaluierung

- Unvollständige Dokumentation:
  - Übergabeparameter der Funktionen dokumentieren
- Getestete Teile der Gerätekommunikation:
  - VISA-Protokoll und PyVisa Package Installation
  - Kommunikation mit AWG von anderem Laptop aus über USB
- Ausstehende Teile der Gerätekommunikation:
  - Kommunikation mit Oszilloskop von anderem Laptop
  - Laufzeit: Status-Abfrage der Geräte mit `BUSY?` oder `*WAI`
  - Anpassung der Auflösung des DSO

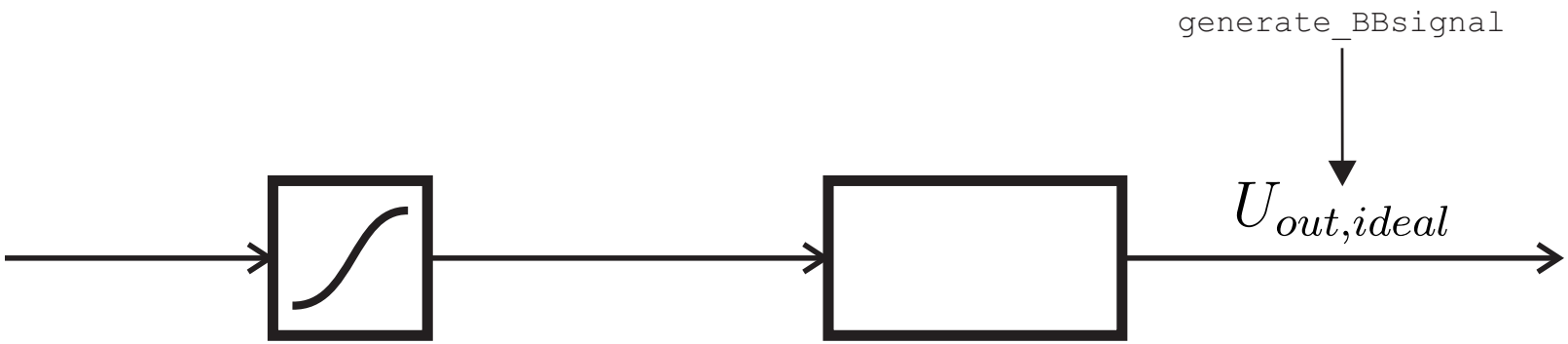
Code: Die Bausteine



Code: Die Bausteine

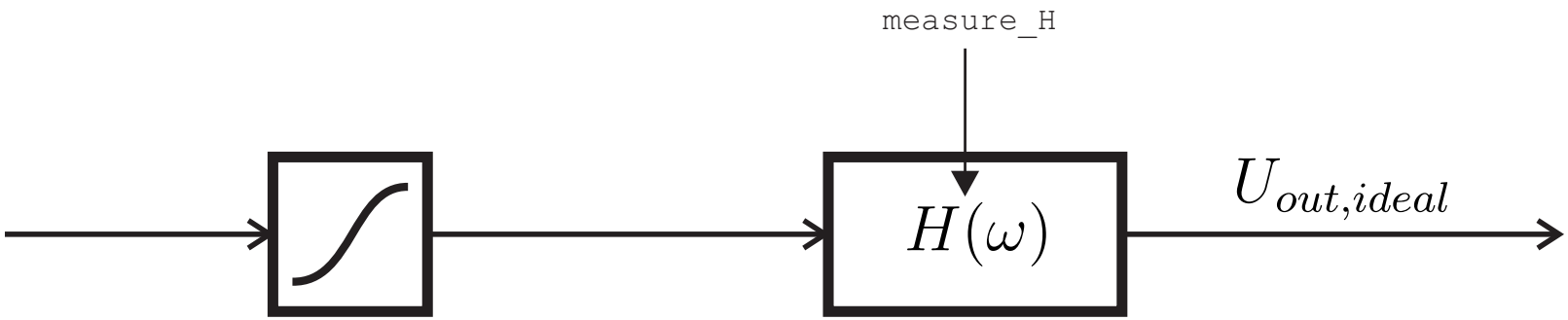


Code: Die Bausteine



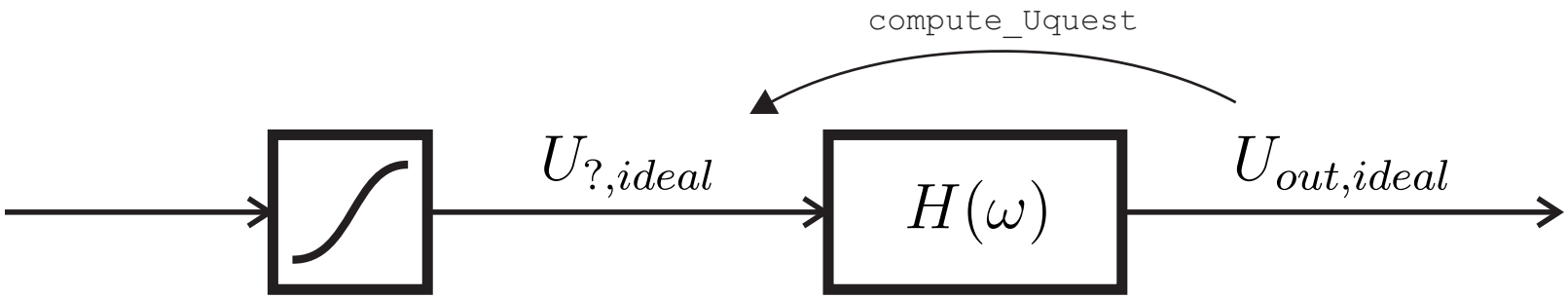

```
1 Uout_ideal = generate_BBsignal ( fq_rep , fq_bb , vpp )
```

Code: Die Bausteine



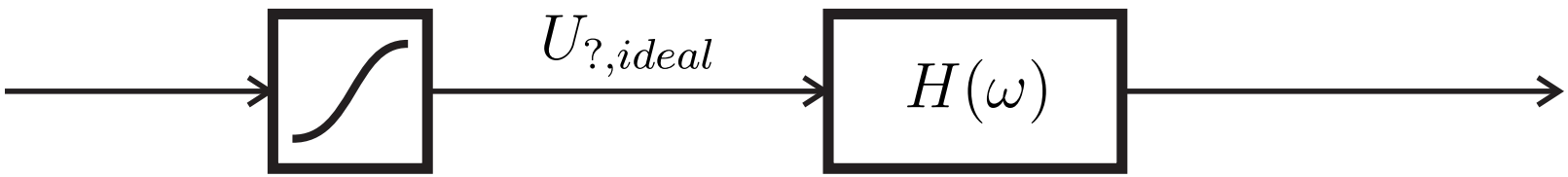

```
1 Uout_ideal = generate_BBsignal ( fq_rep, fq_bb, vpp )
2 H = measure_H ( )
```

Code: Die Bausteine



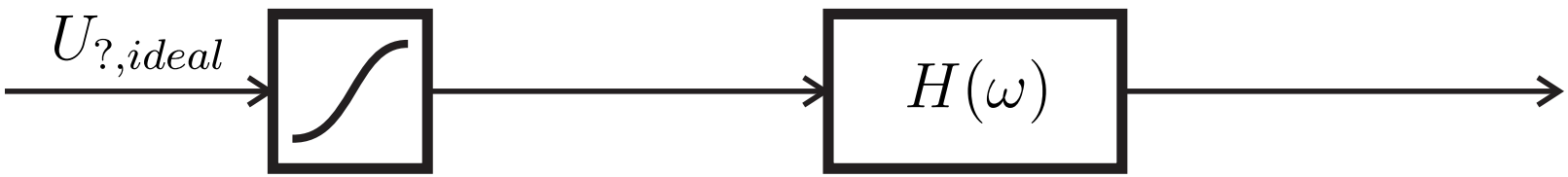

```
1 Uout_ideal = generate_BBsignal ( fq_rep, fq_bb, vpp )
2 H = measure_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal, H )
```

Code: Die Bausteine



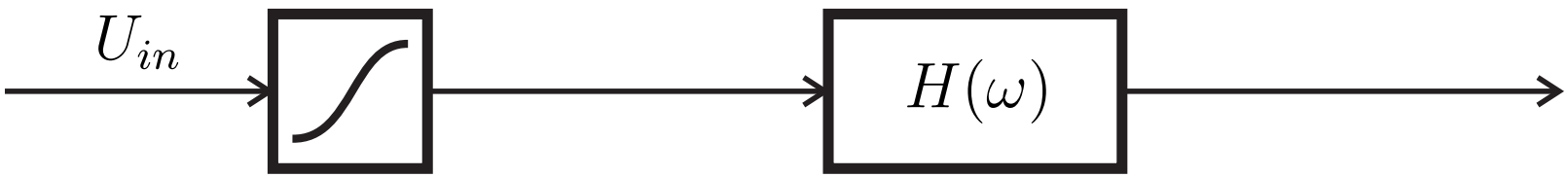

```
1 Uout_ideal = generate_BBsignal ( fq_rep, fq_bb, vpp )
2 H = measure_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal, H )
```

Code: Die Bausteine



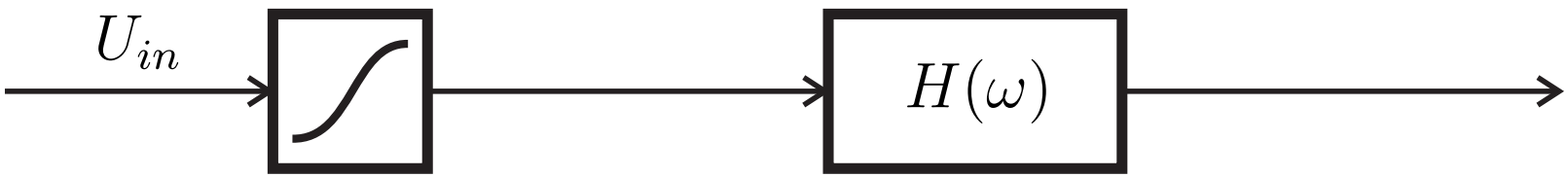

```
1 Uout_ideal = generate_BBsignal ( fq_rep, fq_bb, vpp )
2 H = measure_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal, H )
```

Code: Die Bausteine



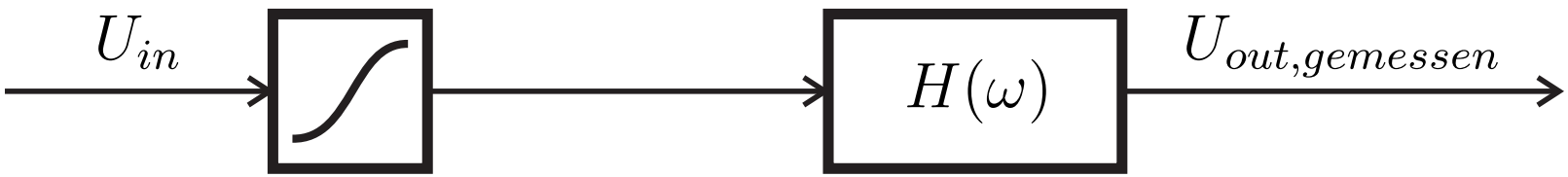

```
1 Uout_ideal = generate_BBsignal ( fq_rep, fq_bb, vpp )
2 H = measure_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal, H )
4 Uin = Uquest_ideal
```

Code: Die Bausteine



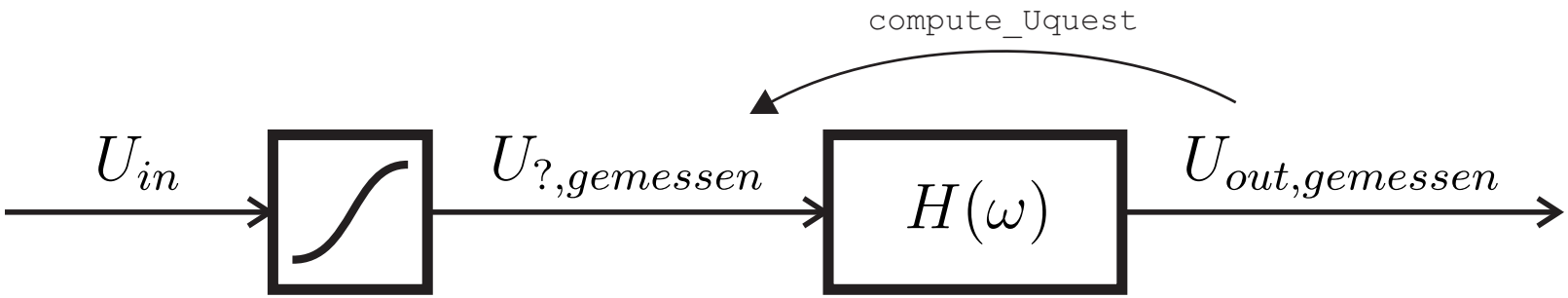

```
Uout_ideal = generate_BBsignal ( fq_rep, fq_bb, vpp )
H = measure_H ( )
Uquest_ideal = compute_Uquest ( Uout_ideal, H )
Uin = Uquest_ideal
Uout_measured = measure_Uout ( Uin )
```

Code: Die Bausteine



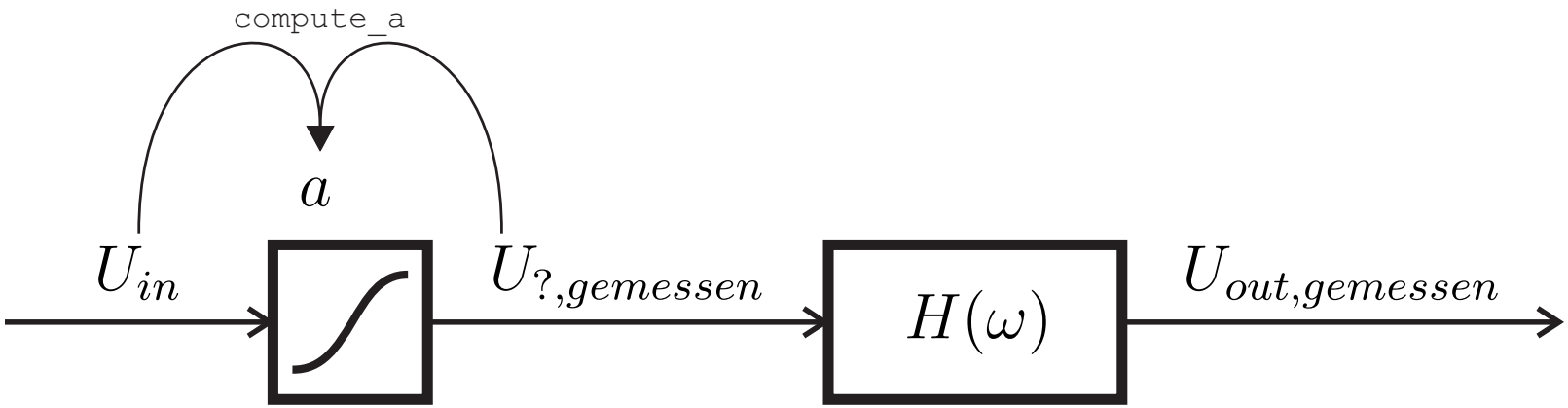

```
Uout_ideal = generate_BBsignal ( fq_rep, fq_bb, vpp )
H = measure_H ( )
Uquest_ideal = compute_Uquest ( Uout_ideal, H )
Uin = Uquest_ideal
Uout_measured = measure_Uout ( Uin )
```

Code: Die Bausteine



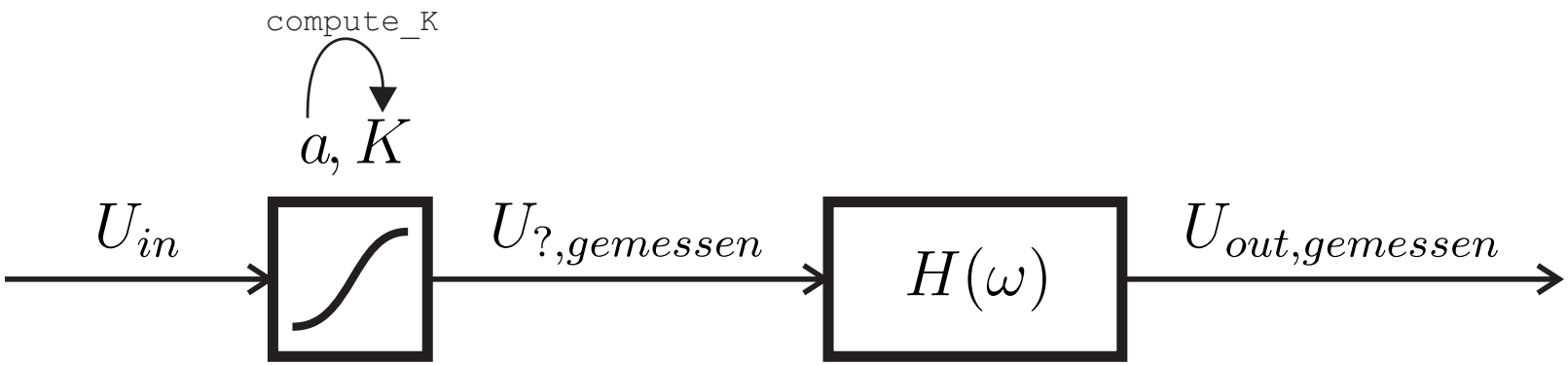

```
1 Uout_ideal = generate_BBsignal ( fq_rep, fq_bb, vpp )
2 H = measure_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal, H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured, H )
```

Code: Die Bausteine



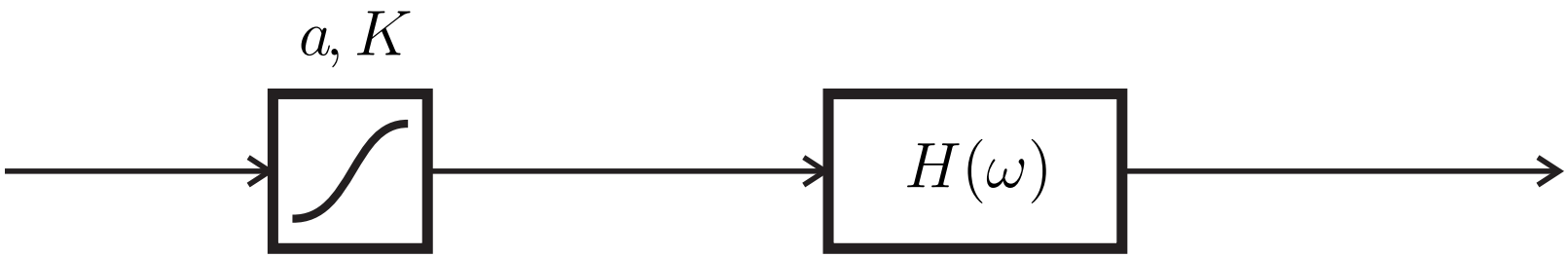

```
1 Uout_ideal = generate_BBsignal ( fq_rep, fq_bb, vpp )
2 H = measure_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal, H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured, H )
7 a = compute_a ( Uin, Uquest_measured, N )
```

Code: Die Bausteine



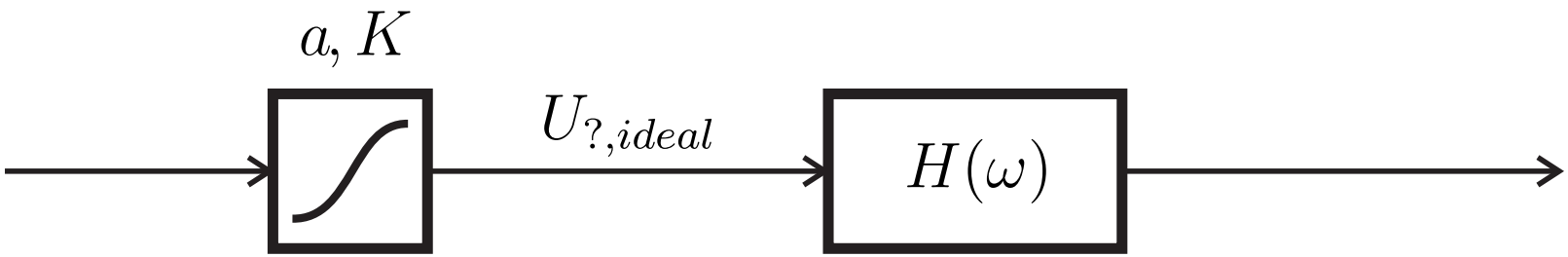

```
1 Uout_ideal = generate_BBsignal ( fq_rep, fq_bb, vpp )
2 H = measure_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal, H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured, H )
7 a = compute_a ( Uin, Uquest_measured, N )
8 K = compute_K ( a )
```

Code: Die Bausteine



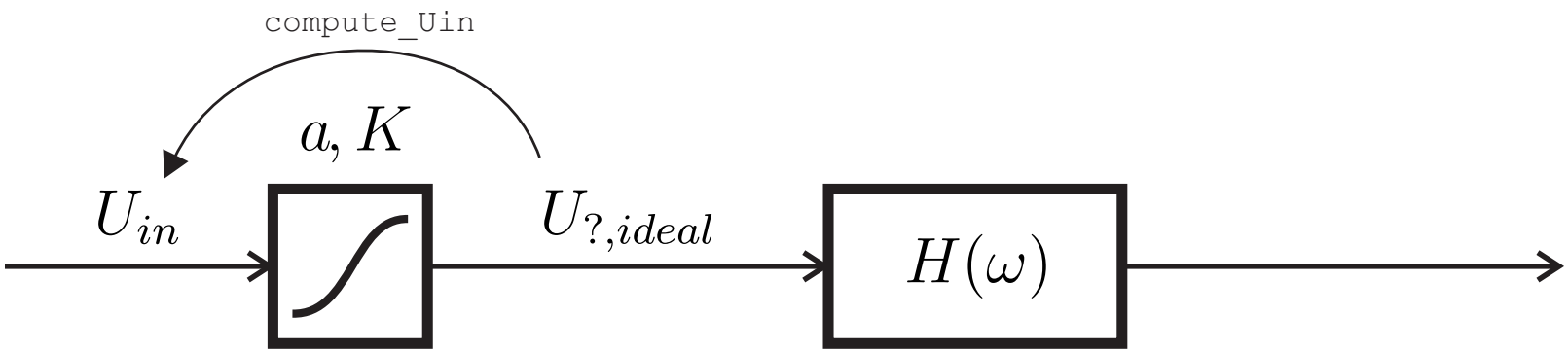

```
Uout_ideal = generate_BBsignal ( fq_rep, fq_bb, vpp )
H = measure_H ( )
Uquest_ideal = compute_Uquest ( Uout_ideal, H )
Uin = Uquest_ideal
Uout_measured = measure_Uout ( Uin )
Uquest_measured = compute_Uquest ( Uout_measured, H )
a = compute_a ( Uin, Uquest_measured, N )
K = compute_K ( a )
```

Code: Die Bausteine



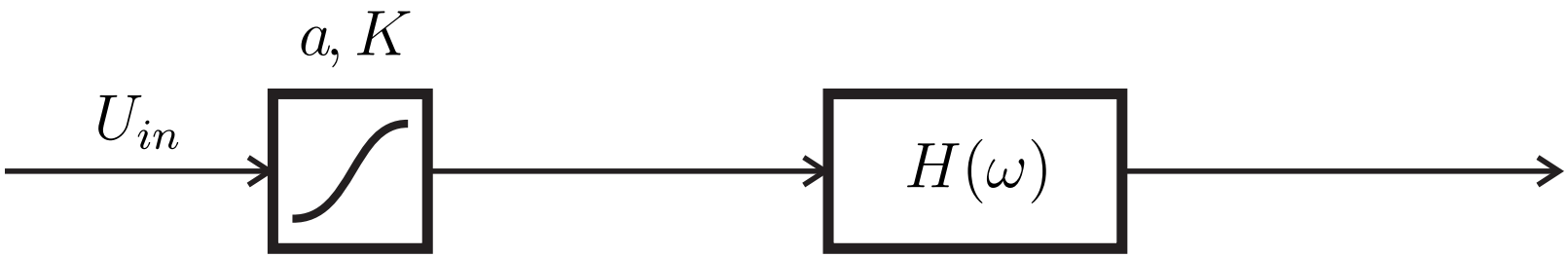

```
1 Uout_ideal = generate_BBsignal ( fq_rep, fq_bb, vpp )
2 H = measure_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal, H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured, H )
7 a = compute_a ( Uin, Uquest_measured, N )
8 K = compute_K ( a )
```

Code: Die Bausteine



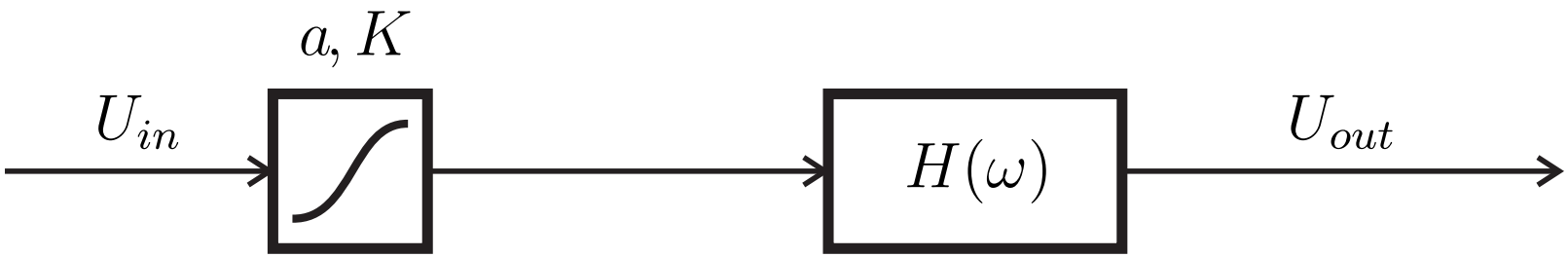

```
1 Uout_ideal = generate_BBsignal ( fq_rep, fq_bb, vpp )
2 H = measure_H ( )
3 Uquest_ideal = compute_Uquest ( Uout_ideal, H )
4 Uin = Uquest_ideal
5 Uout_measured = measure_Uout ( Uin )
6 Uquest_measured = compute_Uquest ( Uout_measured, H )
7 a = compute_a ( Uin, Uquest_measured, N )
8 K = compute_K ( a )
9 Uin = compute_Uin ( Uquest_ideal, K )
```

Code: Die Bausteine




```
Uout_ideal = generate_BBsignal ( fq_rep, fq_bb, vpp )
H = measure_H ( )
Uquest_ideal = compute_Uquest ( Uout_ideal, H )
Uin = Uquest_ideal
Uout_measured = measure_Uout ( Uin )
Uquest_measured = compute_Uquest ( Uout_measured, H )
a = compute_a ( Uin, Uquest_measured, N )
K = compute_K ( a )
Uin = compute_Uin ( Uquest_ideal, K )
```

Code: Die Bausteine




```
Uout_ideal = generate_BBsignal ( fq_rep, fq_bb, vpp )
H = measure_H ( )
Uquest_ideal = compute_Uquest ( Uout_ideal, H )
Uin = Uquest_ideal
Uout_measured = measure_Uout ( Uin )
Uquest_measured = compute_Uquest ( Uout_measured, H )
a = compute_a ( Uin, Uquest_measured, N )
K = compute_K ( a )
Uin = compute_Uin ( Uquest_ideal, K )
Uout = measure_Uout ( Uin )
```

Code: Vorgehensweise

Code: Vorgehensweise

- Refactoring / Anpassung der Matlab-Funktionen an unser Design

Code: Vorgehensweise

- Refactoring / Anpassung der Matlab-Funktionen an unser Design
- Portierung der Matlab-Funktionen nach Python

Code: Vorgehensweise

- Refactoring / Anpassung der Matlab-Funktionen an unser Design
- Portierung der Matlab-Funktionen nach Python
- Überprüfung der portierten Funktionen mithilfe von **TDD**

Code: Vorgehensweise

- Refactoring / Anpassung der Matlab-Funktionen an unser Design
- Portierung der Matlab-Funktionen nach Python
- Überprüfung der portierten Funktionen mithilfe von **TDD**
- Erweiterung der Blöcken mit `adjust_H` und `adjust_a`

Code: Vorgehensweise

- Refactoring / Anpassung der Matlab-Funktionen an unser Design
- Portierung der Matlab-Funktionen nach Python
- Überprüfung der portierten Funktionen mithilfe von **TDD**
- Erweiterung der Blöcken mit `adjust_H` und `adjust_a`
- Neue Klassen `signale_class` und `transfer_function_class`

Test Driven Development

Test Driven Development

- 27 Unit Tests

Test Driven Development

- 27 Unit Tests
- 4 System Tests

Test Driven Development

- 27 Unit Tests
- 4 System Tests

Vorteile:

Test Driven Development

- 27 Unit Tests
- 4 System Tests

Vorteile:

- Ermöglichen:
  - inkrementierende Code-Anpassungen

Test Driven Development

- 27 Unit Tests
- 4 System Tests

Vorteile:

- Ermöglichen:
  - inkrementierende Code-Anpassungen
  - verteiltes Debuggen ohne den Messaufbau

Test Driven Development

- 27 Unit Tests
- 4 System Tests

Vorteile:

- Ermöglichen:
  - inkrementierende Code-Anpassungen
  - verteiltes Debuggen ohne den Messaufbau
- Zwingen zum modularen Code-Design

Test Driven Development

- 27 Unit Tests
- 4 System Tests

Vorteile:

- Ermöglichen:
  - inkrementierende Code-Anpassungen
  - verteiltes Debuggen ohne den Messaufbau
- Zwingen zum modularen Code-Design
- Erleichtern das Migrieren der Funktionen aus anderen Sprachen

Test Driven Development

- 27 Unit Tests
- 4 System Tests

Vorteile:

- Ermöglichen:
  - inkrementierende Code-Anpassungen
  - verteiltes Debuggen ohne den Messaufbau
- Zwingen zum modularen Code-Design
- Erleichtern das Migrieren der Funktionen aus anderen Sprachen
- Dienen auch als Dokumentation der Bausteine

Test Driven Development

- 27 Unit Tests
- 4 System Tests

Vorteile:

- Ermöglichen:
  - inkrementierende Code-Anpassungen
  - verteiltes Debuggen ohne den Messaufbau
- Zwingen zum modularen Code-Design
- Erleichtern das Migrieren der Funktionen aus anderen Sprachen
- Dienen auch als Dokumentation der Bausteine

Nachteile:

Test Driven Development

- 27 Unit Tests
- 4 System Tests

Vorteile:

- Ermöglichen:
  - inkrementierende Code-Anpassungen
  - verteiltes Debuggen ohne den Messaufbau
- Zwingen zum modularen Code-Design
- Erleichtern das Migrieren der Funktionen aus anderen Sprachen
- Dienen auch als Dokumentation der Bausteine

Nachteile:

- Extra Aufwand: mehr Code zu debuggen

Test Driven Development

- 27 Unit Tests
- 4 System Tests

Vorteile:

- Ermöglichen:
  - inkrementierende Code-Anpassungen
  - verteiltes Debuggen ohne den Messaufbau
- Zwingen zum modularen Code-Design
- Erleichtern das Migrieren der Funktionen aus anderen Sprachen
- Dienen auch als Dokumentation der Bausteine

Nachteile:

- Extra Aufwand: mehr Code zu debuggen
- Gewöhnungsbedürftig

Das Mock-System

- Wird genutzt, wenn mit Geräten kommuniziert wird:
  - `mock_system.write_to_AWG`
  - `mock_system.read_from_DSO`

Das Mock-System

- Wird genutzt, wenn mit Geräten kommuniziert wird:
  - `mock_system.write_to_AWG`
  - `mock_system.read_from_DSO`
- Simuliert das Verhalten des Messaufbaus nach dem Hammerstein Model

Das Mock-System

- Wird genutzt, wenn mit Geräten kommuniziert wird:
  - `mock_system.write_to_AWG`
  - `mock_system.read_from_DSO`
- Simuliert das Verhalten des Messaufbaus nach dem Hammerstein Model

Vorteile:

Das Mock-System

- Wird genutzt, wenn mit Geräten kommuniziert wird:
  - `mock_system.write_to_AWG`
  - `mock_system.read_from_DSO`
- Simuliert das Verhalten des Messaufbaus nach dem Hammerstein Model

Vorteile:

- Ermöglicht:
  - Unit Tests von Bausteinen, in den Gerätekommunikation stattfindet

Das Mock-System

- Wird genutzt, wenn mit Geräten kommuniziert wird:
  - `mock_system.write_to_AWG`
  - `mock_system.read_from_DSO`
- Simuliert das Verhalten des Messaufbaus nach dem Hammerstein Model

Vorteile:

- Ermöglicht:
  - Unit Tests von Bausteinen, in den Gerätekommunikation stattfindet
  - System Tests

Das Mock-System

- Wird genutzt, wenn mit Geräten kommuniziert wird:
  - `mock_system.write_to_AWG`
  - `mock_system.read_from_DSO`
- Simuliert das Verhalten des Messaufbaus nach dem Hammerstein Model

Vorteile:

- Ermöglicht:
  - Unit Tests von Bausteinen, in den Gerätekommunikation stattfindet
  - System Tests
  - Testen von Randfällen

Das Mock-System

- Wird genutzt, wenn mit Geräten kommuniziert wird:
  - `mock_system.write_to_AWG`
  - `mock_system.read_from_DSO`
- Simuliert das Verhalten des Messaufbaus nach dem Hammerstein Model

Vorteile:

- Ermöglicht:
  - Unit Tests von Bausteinen, in den Gerätekommunikation stattfindet
  - System Tests
  - Testen von Randfällen
- Hilft, das System besser zu verstehen

Das Mock-System

- Wird genutzt, wenn mit Geräten kommuniziert wird:
  - `mock_system.write_to_AWG`
  - `mock_system.read_from_DSO`
- Simuliert das Verhalten des Messaufbaus nach dem Hammerstein Model

Vorteile:

- Ermöglicht:
  - Unit Tests von Bausteinen, in den Gerätekommunikation stattfindet
  - System Tests
  - Testen von Randfällen
- Hilft, das System besser zu verstehen

Nachteile:

- Extra Aufwand: mehr Code zu debuggen

Ausblick

Ausblick

- Einbindung des RF-Data-Tools zur Beurteilung der Qualität des Ausgangssignals

Ausblick

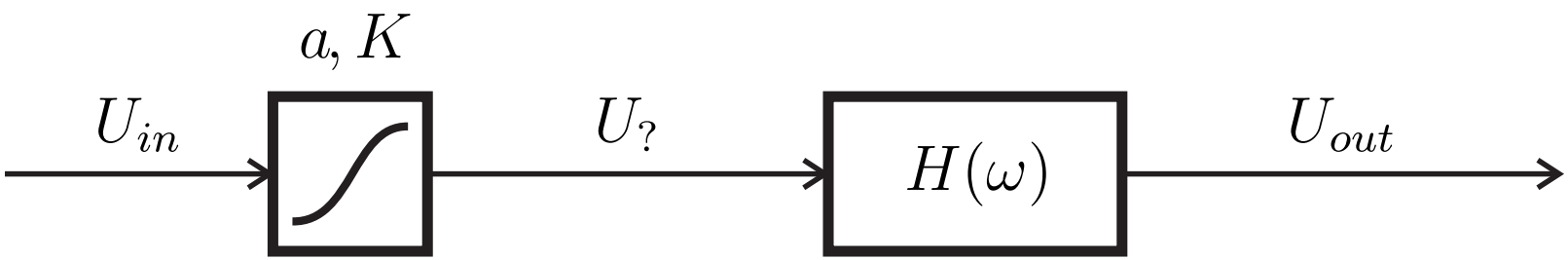
- Einbindung des RF-Data-Tools zur Beurteilung der Qualität des Ausgangssignals
- *Zusatz:* Konvertierung der Funktionalitäten von Python in die TEMF RF-Data-Tools

Ausblick

- Optimierung der linearen Übertragungsfunktion:

$$\underline{H}^{\text{neu}}(\omega) = \underline{H}^{\text{alt}}(\omega)\left(1 + \sigma_H \cdot \left(\frac{\underline{U}_{out,\text{mess}}(\omega)}{\underline{U}_{out,\text{ideal}}(\omega)} - 1\right)\right)$$

mit σ_H als Schrittweite.

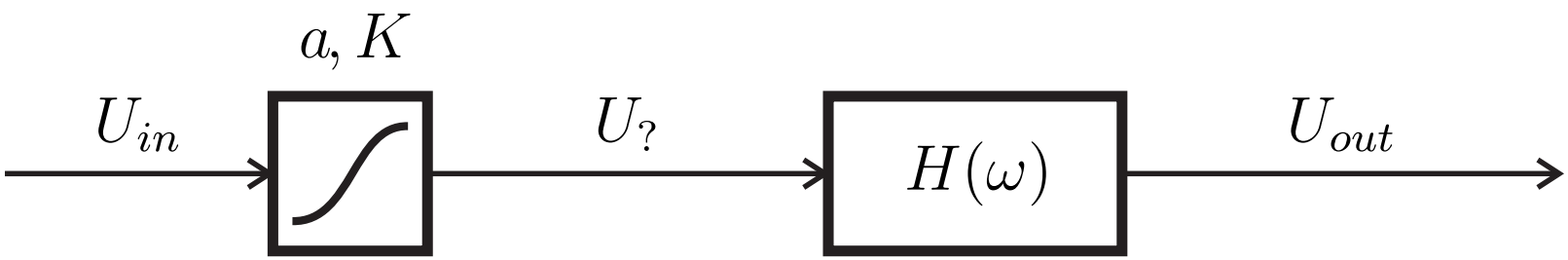


Ausblick

- Optimierung der nichtlinearen Kennlinie:

$$\Delta U_? = U_{?,\text{mess}} - U_{?,\text{berechnet}} = \sum_n \tilde{a}_n U_{in}^n \qquad a_n^{\text{neu}} = a_n^{\text{alt}} + \sigma_a \cdot \tilde{a}_n$$

mit σ_a als Schrittweite.



Empfehlungen zum Code-Design

- Weiter Klasse `K_class` implementieren

Empfehlungen zum Code-Design

- Weiter Klasse `K_class` implementieren
- Refactoring

Überlegungen

Überlegungen

- Reihenfolge der Optimierung: Parallele Iteration $\Leftrightarrow$ alternierende Iteration von H und K

Überlegungen

- Reihenfolge der Optimierung: Parallele Iteration $\Leftrightarrow$ alternierende Iteration von H und K
- Einfluss von K auf das Spektrum von $U_?$ und damit auf Optimierung von H durch Oberschwingungen bei Potenzierung des Eingangssignals

Quellen

- Denys Bast, Armin Galetzka, "Projektseminar Beschleunigertechnik", 2017
- Jens Harzheim *et al.*, "Input Signal Generation For Barrier Bucket RF Systems At GSI",
- Kerstin Gross *et al.*, "Test Setup For Automated Barrier Bucket Signal Generation", 2017