

Zusammenfassung IAUT

Eric Lindegger – HS25

Danke fürs Herunterladen und Verwenden meiner IAUT Zusammenfassung! Hoffe ihr findet sie nützlich. Falls ihr Fehler findet oder Inputs habt, schreibt mir doch auf eric.lindegger@stud.hslu.ch
Quelldateien: GitHub

Inhaltsverzeichnis

1 Automatisierung	2
1.1 Steuerung - Überblick	2
1.1.1 Klemmen	2
1.2 Allgemeine Aufgaben	2
1.3 Allgemeine Anforderungen	2
2 SPS	3
2.1 Feldbus	3
2.1.1 Spezifische Feldbusse	3
2.2 Diskrete Regelungsaufgabe	3
3 Programmierung	3
3.1 EN 61499	3
3.2 Grundaufbau	4
3.2.1 Programm	4
3.2.2 Funktionsblock	4
3.2.3 Funktion	4
3.3 VariablenTypen	4
3.3.1 System Datentypen	4
3.3.2 Array	4
3.3.3 DUT - Data Unit Type	4
3.3.4 Globale Variablen Liste (GVL)	5
3.3.5 Type Casting	5
3.4 Conditional Statements	5
3.4.1 If/Else	5
3.4.2 Switch-Case	5
3.5 Schleifen	5
3.5.1 WHILE	5
3.5.2 REPEAT	5
3.5.3 FOR	5
3.5.4 Schleifen Modifikatoren	5
4 TwinCat	6
4.1 Bitshifting	6
4.2 Limit	6
4.3 Timer	6
4.3.1 Timer Off-Delay (TOF)	6
4.3.2 Timer On-Delay (TON)	6
4.3.3 Timer Pulse Generator (TP)	6
5 Digitale Regelung	6
5.1 Laplace Diskretisierung	6
5.1.1 Rückwärts-Rechteckregel	6
5.1.2 Vorwärtsrechteck-Regel	6
5.1.3 Trapezoidal-Regel (Tustin Approximation)	6
5.2 PT1-Prozess	7
5.3 PT2-Prozess	7
5.4 Diskreter PID-Regler	7
5.4.1 P-Anteil	7
5.4.2 I-Anteil	7
5.4.3 D-Anteil	7
5.5 Stellgrössensättigung	7
5.5.1 Antireset-Windup (ARW)	8
6 FSM - Finite State Machine	8
7 Beispiel Code	8
7.1 Flankendetektor	8
7.1.1 Struct	8
7.1.2 Funktionsblock	9
7.2 PID-Regler	9
7.2.1 Parameter Struct	9
7.2.2 System Modus Enum	9
7.2.3 Funktionsblock	9
7.3 PT1 Prozess	10
7.4 PT2 Prozess	10
7.4.1 Parameter Struct	10

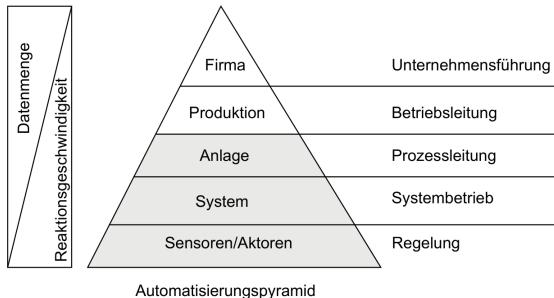
7.4.2 Funktionsblock	10
8 Matlab-Befehle	10
8.1 Sprungantwort modellieren	10
8.2 Simulink	11
9 Hilfsblätter	11
9.1 Laplace Transformation	11
9.2 z-Transformation	12

1 Automatisierung

Definition Automatisierung

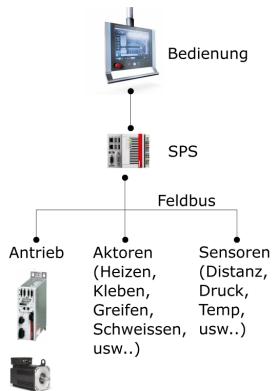
Unter Automatisierung (DIN19222) versteht man das gezielte Ausrüsten einer Einrichtung, so dass sie ganz oder teilweise ohne Mitwirkung des Menschen geschieht und arbeiten kann.

Die Automatisierungspyramide ist der Versuch die gesamte Automatisierungskette auf ein Bild abzubilden:



Die Darstellung ist wie folgt zu interpretieren: Je weiter unten, desto schneller müssen die Komponenten reagieren und je weiter oben, ist die Datenmenge größer. Die Absicht der Automatisierung ist, dass die automatisierte Anlage bis ins ERP der Firma eingepflegt wird. So muss z.B. keine spezielle Benutzeroberfläche geladen werden um ein neuer Prozess auf der Anlage zu starten.

1.1 Steuerung - Überblick



Die Bedienung ist wo das GUI (Graphical User Interface) dargestellt wird. Hier werden vom Benutzer Eingaben getätigt, wie z.B. Job oder Rezeptwahl.

Der Controller innerhalb der Steuerung/Anlage ist die CPU/SPS. Eine Automatisierung ist häufig eine (PID-)Regelungsaufgabe. Die SPS ist mit Aktoren und Sensoren verbunden:

- Sensoren liefern Informationen
- Aktoren lösen Aktion aus

Die Schnittstellen der Sensoren/Aktoren ist entweder eine Spannung oder einen Strom. Die Spannung ist jedoch eher weniger gängig im Moment, da die Spannung anfälliger auf Störeinflüsse ist als ein Strom. Die gängigsten Schnittstellen sind. Ein weiterer Vorteil der Stromschnittstelle ist, dass auch ein Kabelbruch detektiert werden kann, da der Strom nie 0 wird:

- $0V - 10V$
- $4mA - 20mA$

1.1.1 Klemmen

Die Klemmen stellen die Ein- und Ausgänge einer SPS dar und können auf der Feldebene der SPS über ein Bussystem erweitert werden:



Klemmen können über RS485 abgesetzt werden und an einem anderen Ort als direkt bei der SPS eingesetzt werden.

1.2 Allgemeine Aufgaben

Echtzeitanforderung
Regler
Prozessabläufen
usw.

keine Echtzeitanforderung
Mensch-Maschine-Interface (HMI)
Produktionsplanung
Archivierung
usw.

1.3 Allgemeine Anforderungen

Echtzeitfähigkeit
Betriebssysteme
Speicherarchitektur

Ein- Ausgabe von Prozesssignalen
Aktoren
Sensoren
Kommunikationssysteme

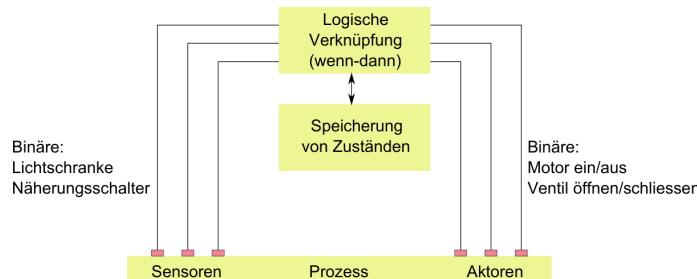
Sicherheit
Zuverlässigkeit
Hochwertige Komponente
Redundanz

Resistenz gegen Umwelteinflüsse
Spezielle Gehäuse
Schaltschrankmontage

2 SPS



Eine SPS ist eine 'Speicherprogrammierbare Steuerung'. Auf Englisch wird sie PLC (Programmable Logic Controller) genannt. Sehr wichtig: die SPS arbeitet konventionell möglichst auf if/else Statements und verarbeitet sämtlicher Code in jedem Zyklus einmal. Das Arbeitsprinzip ist:



2.1 Feldbus

Der Feldbus ist die Kommunikationsschnittstelle der SPS zu den Klemmen und entsprechend Ein- und Ausgängen. Es hängen alle E/A seriell an einem Bus. Wo der Feldbus aktiv ist, wird Feldebene genannt. Der Aufbau des Feldbusses kann wie folgt vorgestellt werden:



Definition Feldbus

Ein Feldbus verbindet in einer Anlage Feldgeräte wie Sensoren und Aktoren zwecks Kommunikation mit einem Steuerungsgerät (SPS).

Durch den Feldbus entstehen einige wesentliche Vor- und Nachteile:

- + geringer Verkabelungsaufwand
- + Erweiterungen oder Änderungen sind einfach
- Komplexität
- Preis
- Aufwendige Messgeräte (Analyzer)
- Längere Reaktionszeit

2.1.1 Spezifische Feldbusse

Im Gebäude sind die gängigen Feldbusse:

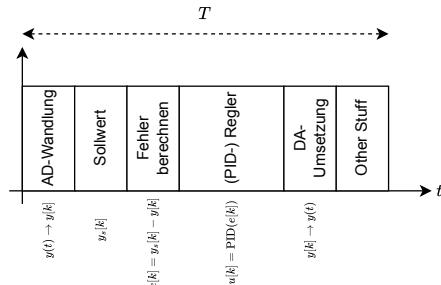
- LON
- EIB
- KNX

Innerhalb der Maschine werden andere Feldbusse verwendet:

- CAN/CANOpen
- Profibus, Profinet
- EtherCAT (nicht TCP basierend)
- Ethernet/IP

2.2 Diskrete Regelungsaufgabe

Eine diskrete Regelungsaufgabe benötigt Zeit. Über Messung der Sensorwerte, Vergleich mit Sollwert etc. verzögern die Reaktionszeit des Reglers drastisch.



Es gilt dabei die folgende Bedingung:

$$T_{AD} + T_C + T_{DA} \ll T$$

3 Programmierung

Definition SPS-Programmierung

Die EN 61131-3 ist die einzige weltweit gültige Norm für Programmiersprachen von SPS-Steuerungen. Sie definiert die folgenden fünf Sprachen:

- AWL - Anweisungsliste (Assembler)
- KOP - Kontaktplan
- FBS - Funktionsbausteinsprache
- ST - Struktureierter Text
- AS - Ablaufsprache

Insgesamt wären es 6, da noch IL (Instruction List) definiert wäre.

Die einzige (relevante) Programmiersprache hinsichtlich des IAUT Unterrichts ist ST. Die anderen werden für einfache logische Verknüpfungen gebraucht, oder für z.B. I/O-Zuweisungen. ST wird in Europa oft für die Programmierung von SPS gewählt.

3.1 EN 61499

Diese Norm stellt eine objektorientierte Erweiterung der EN61131 dar. Dieses Kapitel fasst den Aufbau und Syntax der SPS-Programmiersprache ST zusammen.

3.2 Grundaufbau

Der Aufbau eines Programmes/Funktionsblocks oder Funktion ist immer in zwei Bereiche aufgeteilt: Deklaration und Implementation. Im Deklarationsteil werden alle Inputs, Outputs deklariert und ggf. initialisiert. In der Implementation wird der eigentliche Code geschrieben. Wichtig: die Implementation wird in jedem Clock Zyklus wiederholt.

```

1 // Deklarationsteil
2 VAR_INPUT
3     // input-variablen
4     iVarIn AT %I* : INT;
5 END_VAR
6 VAR_OUTPUT
7     // output-variablen
8     iVarOut AT %Q* : INT;
9 END_VAR
10 VAR_IN_OUT
11     // in/out-variablen
12 END_VAR
13 VAR
14     // lokale variablen
15     tmp : INT;
16 END_VAR
17
18 // Implementationsteil
19 IF iVarIn > 10 THEN
20     tmp := 0;
21 ELSE
22     tmp := iVarIn;
23 END_IF

```

3.2.1 Programm

Programme können als Funktionen aufgerufen werden und besitzen keinen Rückgabewert.

```

1 PROGRAM ProgramName
2 VAR
3     bTrig : BOOL;
4     iVal : INT;
5 END_VAR
6
7 IF(bTrig = TRUE) THEN
8     iVal := 15;
9 END_IF

```

3.2.2 Funktionsblock

Ein Funktionsblock ist Zustandsbehaftet: Interne Variablen behalten ihren Wert über die SPS-Zyklen weg. Jede Instanz besitzt einen eigenen Speicher. Jedoch müssen die Speicher instanziert werden.

```

1 FUNCTION_BLOCK Count
2 VAR_INPUT
3     bTrig : BOOL;
4 END_VAR
5 VAR
6     iCount : INT;
7 END_VAR
8
9 IF bTrig = TRUE THEN
10     iCount := iCount + 1;
11 END_IF

```

3.2.3 Funktion

Eine Funktion kann ohne Instanzierung verwendet werden. Eine Funktion besitzt keine statische Werte, dafür einen Rückgabewert, welcher auf den Funktionsnamen zugewiesen wird.

```

1 FUNCTION fFunc : INT // Rückgabewert
2 VAR_INPUT
3     iTime : INT;
4     iT : INT;
5 END_VAR
6
7 iTmp := iTime;
8 fFunc := iTime / iT;

```

3.3 Variablenarten

3.3.1 System Datentypen

Die gängigen Datentypen sind auch in ST vorhanden. Die Tabelle fasst die Datentypen zusammen:

Typ	Min	Max	Bits	Bemerkung
bool	0	1	8	Nur 1 / 0 / TRUE / FALSE sind gültig!
BYTE	0	255	8	
WORD	0	65535	16	
DWORD	0	4294967295	32	
SINT	-128	127	8	
USINT	0	255	8	
INT	-32768	32767	16	
UINT	0	65535	16	
DINT	-2147483648	2147483647	32	
UDINT	0	4294967295	32	
REAL	$-3.4 \cdot 10^{38}$	$-3.4 \cdot 10^{38}$	64	
LREAL	$-1.8E+308$	$1.8E+308$	64	
STRING	—	—	n+1	fügt Nullterminierung an.
TIME	T#0ms	T#71582m 47s295ms	32	
TOD	TOD#00:00	TOD#1193:02: 47.295	32	
DATE	D#1970-01-01	D#2106-02-06	32	
DT	DT#1970-0-0	DT#2106-02-06- 00:00:00	32	

3.3.2 Array

```

1 VAR
2     aValues : ARRAY[0..99] OF INT;
3 END_VAR

```

3.3.3 DUT - Data Unit Type

Über DUT können benutzerdefinierte Datentypen angelegt werden. Die relevanten sind **STRUCT** und **ENUM**. Durch diese Typen kann ein Programm bedeutend lesbacher gestaltet werden. Für die Erstellung eines DUTs wird ein neues File in der DUT Ordnerstruktur erstellt. Die Definition geht wie folgt:

Struct

```

1 // Struct
2 TYPE stDataType:
3     STRUCT
4         iVar1 : INT;
5         iPointX : INT;
6         iPointY : INT;
7     END_STRUCT
8 END_TYPE

```

Enumeration

```

1 // Enumeration
2 TYPE eSystemState:
3 (
4     None, // = 0 - default
5     init, // = 1
6     run
7 ) := init; // default state
8 END_TYPE

```

3.3.4 Globale Variablen Liste (GVL)

Die GVL deklariert Variablen, welche in jedem Programm, Funktionsblock und Funktion aufrufbar. Vielfach macht Sinn die In-/Outputs, welche mit der Hardware kommunizieren in einer GVL zu deklarieren.

3.3.5 Type Casting

Variabtentypen können mit Hilfe von Casting Funktionen umgewandelt werden. Fast alle Typen können gewandelt werden:

```
1 rRealVal := INT_TO_REAL(iVal);
```

Casting ist die zielführende Methode, um keine Fehler im Implementieren des Programms zu generieren. TwinCat hat implizite Casts nicht so gerne.

3.4 Conditional Statements

Conditional Operatoren

Wie in C gibt es in ST mehrere Conditional Operatoren. Diese sind wie folgt:

=	ist gleich	AND	und
<>	ist ungleich	OR	oder
<	kleiner als	NOT	'nicht'
>	grösser als	XOR	exklusiv oder
<=	kleiner gleich		
>=	grösser gleich		

Diese Operatoren können kombiniert angewendet werden:

```

1 IF motor AND NOT fault THEN
2 IF (a = b) AND (x <> y OR NOT z) THEN

```

3.4.1 If/Else

```

1 IF <cond1> THEN
2 ...
3 ELSIF <cond2> THEN
4 ...
5 ELSE
6 ...
7 END_IF

```

3.4.2 Switch-Case

```

1 CASE iVar OF
2     1:
3         // do something for 1
4     2..5:
5         // do something for range 2 - 5
6 ELSE
7     // do this as default-assignment
8 END_CASE

```

Die Switch-Bedingung kann auch für Enum-Typen gebraucht werden:

```

1 CASE eState OF
2     Init:
3         // handle init state
4     Running:
5         // handle running state
6     Error:
7         // handle error state
8 ELSE
9     // it is beneficial to always have a default
10    statement
11 END_CASE

```

3.5 Schleifen

Wie in C gibt es in TwinCat Schleifen. Dabei ist es von zentraler Bedeutung, dass diese Schleife nicht Endlos ist und eine Abbruchbedingung enthält. Ansonsten kann es vorkommen, dass auf die SPS nicht mehr zugriffen werden kann. Die verschiedenen Schleifen werden hier aufgeführt.

3.5.1 WHILE

WHILE-Schleife

Sofora die Bedingung beim ersten Erreichen nicht zutrifft, wird der Inhalt der Schleife nicht ausgeführt.

```

1 WHILE iCounter <> 0 DO
2     iVar1 := iVar2;
3     iCounter := iCounter - 1;
4 END WHILE

```

3.5.2 REPEAT

REPEAT-Schleife

Diese Schleife wird mindestens einmal durchgeführt.

```

1 REPEAT
2     iVar1 := iVar2;
3     iCounter := iCounter - 1;
4 UNTIL
5     iCounter = 0;
6 END_REPEAT

```

3.5.3 FOR

```

1 FOR iCounter = 1 TO 5 BY 1 DO
2     iVar1 := iVar2;
3 END_FOR

```

3.5.4 Schleifen Modifikatoren

Mit diesen Modifikatoren kann wie in C beeinflusst werden, wie das Verhalten in der Schleife ist.

```

1 EXIT      // Ausbruch aus aktuellem Loop
2 CONTINUE   // Direkter Sprung zu Loop Anfang

```

4 TwinCat

TwinCat ist die Entwicklungsumgebung von Beckhoff für deren SPS Systeme. TwinCat liefert diverse Funktionen vorgefertigt für den Benutzer. In diesem Kapitel werden die relevantesten aufgelistet.

Semicolons

TwinCat ist sehr tolerant was Semicolons nach `END_*` Anweisungen betrifft. Gemäss Standard wird jede Anweisung mit ; abgeschlossen. TwinCat toleriert das nicht Setzen dieser Semicolons.

4.1 Bitshifting

Mit Bitshifting Operationen können sämtliche Bits innerhalb einer Variable nach links oder rechts geshiftet werden. Die Bits am Rand werden verworfen und neu entstandene Lücken mit 0 aufgefüllt.

```
1 res := SHL(bByte, 2); // Shift Left
2 res := SHR(bByte, 2); // Shift Right
```

Mit Bitrotate Befehlen können herausgeschobene Bits rotiert werden (herausgeschobene Bits werden am anderen Ende wieder hineingeschoben).

```
1 res := ROL(bByte,2); // Circular Shift Left
2 res := ROL(bByte,3); // Circular Shift Right
```

Die Shift-Operationen können mit allen gängigen Datentypen verwendet werden.

4.2 Limit

Die Limit-Funktion limitiert die Eingangsvariable innerhalb von einer Minimal- und Maximalwert. Die Funktion kann wie folgt verwendet werden:

```
1 rUsat := LIMIT(umin, rU, umax);
2 // rU wird von umin und umax limitiert
```

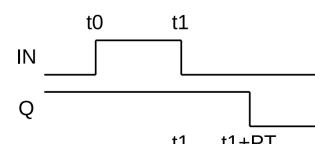
4.3 Timer

Verschiedene Timer Module können direkt aus TwinCat instanziert werden. Diese haben jeweils verschiedene Verhaltensmuster und sind für andere Anwendungen nützlich.

4.3.1 Timer Off-Delay (TOF)



In: Eingangssignal (1 zu 0 startet Timer)
PT: Verzögerungszeit
Q: Ausgang des Timers (1 wenn PT erreicht)
ET: Abgelaufene Zeit

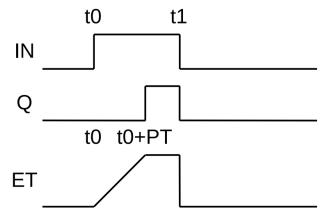


```
1 VAR
2   fbTOF : TOF;
3 END_VAR
4
5 fbTOF(IN := bTrig, PT := T#500ms);
6 bOut := fbTOF.Q;
```

4.3.2 Timer On-Delay (TON)



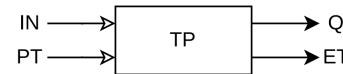
In: Eingangssignal (0 zu 1 startet Timer)
PT: Verzögerungszeit
Q: Ausgang des Timers (1 wenn PT erreicht)
ET: Abgelaufene Zeit



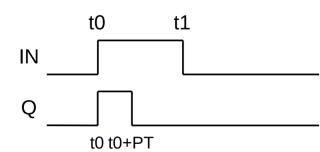
```
1 VAR
2   fbTON : TON;
3 END_VAR
```

```
4
5 fbTON(IN := bTrig, PT := T#500ms);
6 bOut := fbTON.Q;
```

4.3.3 Timer Pulse Generator (TP)



In: Eingangssignal (0 zu 1 startet Timer)
PT: Pulslänge
Q: Ausgang des Timers (1 wenn t < PT)
ET: Abgelaufene Zeit



```
1 VAR
2   fbTP : TP;
3 END_VAR
```

```
4
5 fbTP(IN := bTrig, PT := T#500ms);
6 bOut := fbTP.Q;
```

5 Digitale Regelung

Für Automationsanwendungen werden sehr häufig PID-Regler verwendet. Für eine digitale Regelung muss der Prozess und der Regler diskretisiert werden. Verschiedene Diskretisierungsmethoden existieren.

5.1 Laplace Diskretisierung

Um ein System in Laplace Bereich zu diskretisieren (in den z-Bereich) zu transformieren existieren verschiedene Methoden. Diese sind hier aufgeführt.

5.1.1 Rückwärts-Rechteckregel

$$s = \frac{z - 1}{T_S z} = \frac{1 - z^{-1}}{T_S}$$

Wobei T_S die Zykluszeit der SPS ist. Diese ist standardmäßig 10 ms. Gängigste Art zu diskretisieren

5.1.2 Vorwärtsrechteck-Regel

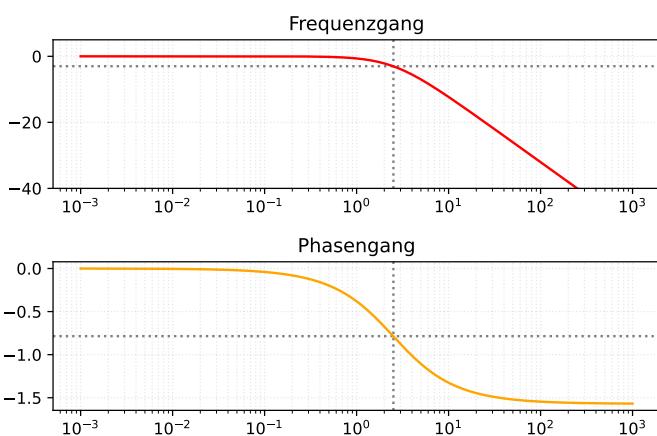
$$s = \frac{z}{T_S z - 1}$$

Für Regelung eine ungeeignete Diskretisierungsmethode.

5.1.3 Trapezoidal-Regel (Tustin Approximation)

$$s = \frac{2}{T_S} \frac{z - 1}{z + 1}$$

5.2 PT1-Prozess

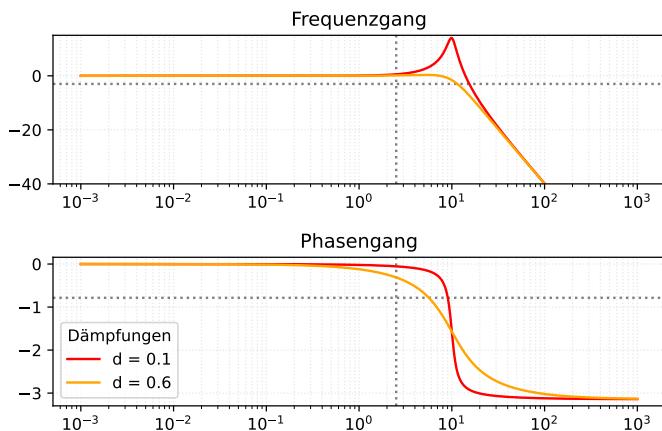


PT1 Glieder besitzen eine Phasenverschiebung von 90° bzw. $\frac{\pi}{2}$. Die Grenzfrequenz liegt bei $\omega_0 = 2\pi\tau$.

$$G(s) = \frac{1}{1 + \tau s} \rightarrow y[k] = a_1 x[k] - a_2 y[k - 1]$$

$$\text{Mit: } a_1 = \frac{T_S}{T_S + \tau} \quad a_2 = \frac{\tau}{T_S + \tau}$$

5.3 PT2-Prozess



Mit zunehmender Dämpfung d sinkt die Resonanz bei der Grenzfrequenz ω_0 . Die Phasenverschiebung ist 180° respektive π .

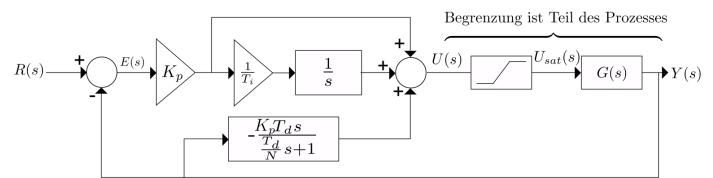
$$G(s) = \frac{\omega_0^2}{s^2 + 2d\omega_0 s + \omega_0^2}$$

Diskretisiert:

$$y[k] = \frac{a_3}{a_1} x[k] - \frac{a_2}{a_1} y[k - 1] - \frac{1}{a_1} y[k - 2]$$

$$\text{Mit: } a_1 = 1 + 2d\omega_0 T_s + \omega_0^2 \quad a_2 = -2 - 2d\omega_0 T_s \quad a_3 = T_s^2 \omega_0^2$$

5.4 Diskreter PID-Regler



Der PID-Regler muss auch diskretisiert werden. Der zeitkontinuierliche PID-Regler hat die Form:

$$C(s) = \frac{U(s)}{E(s)} = k_P \left(1 + \frac{1}{T_i s} + \frac{T_d s}{T_d N s + 1} \right)$$

Legende:

k_P :	Reglerverstärkung
T_i :	Nachstellzeit [s]
T_d :	Vorhaltzeit [s]
N :	$\frac{T_d}{N}$ ist die Zeitkonstante des Filters

5.4.1 P-Anteil

Der diskretisierte P-Anteil ist somit:

$$u_P[k] = k_P \cdot e[k]$$

5.4.2 I-Anteil

Der diskretisierte I-Anteil des Reglers ist:

$$u_I = u_I[k - 1] + \frac{k_P T_s}{T_i} e[k]$$

5.4.3 D-Anteil

Der diskretisierte D-Anteil des Reglers ist:

$$u_D = \frac{k_P \cdot T_d}{T_s} (e[k] - e[k - 1])$$

Kompletter diskreter PID-Regler

Der diskretisierte PID-Regler ist damit zusammengesetzt:

$$u = u_P + u_I + u_D$$

Die vorhergehenden Fehler und Stellgrößen müssen immer mitgeführt werden.

5.5 Stellgrössensättigung

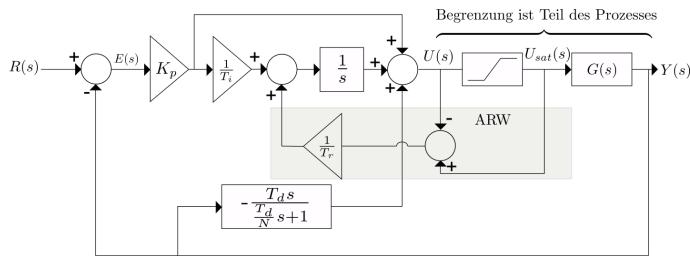
Fast alle Stellgrößen sind in ihrem Output limitiert. Damit die Hardware keinen Schaden nehmen kann, muss die Stellgröße in der Software gesättigt werden. Die Sättigungsfunktion funktioniert anhand eines einfachen mathematischen Prinzips:

$$u_{sat}[k] = \begin{cases} u_{max}, & \text{wenn } u[k] > u_{max} \\ u_{min}, & \text{wenn } u[k] < u_{min} \\ u[k], & \text{sonst} \end{cases}$$

Die Stellgrössensättigung generiert ein neues Problem: der I-Anteil kann sich in die Unendlichkeit aufintegrieren. Um das Problem einzuziehen, wird ein Antireset-Windup (ARW) implementiert. Das Sättigungsverfahren wird in ST wie folgt gelöst:

```
1 usat := LIMIT(umin, u, umax);
```

5.5.1 Antireset-Windup (ARW)



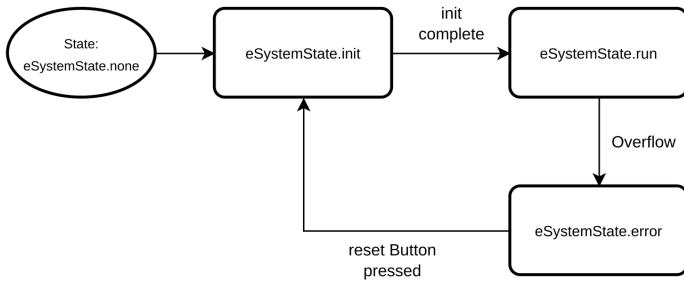
Wird ein ARW implementiert, integriert sich der I-Anteil nicht mehr über das Limit heraus. Der Regler ist damit viel reaktiver. Der I-Anteil mit ARW ist somit:

$$u_I[k] = u_I[k-1] + \frac{k_P T_s}{T_i} e[k] + \frac{T_s}{T_r} (u_{sat}[k] - u[k])$$

Wobei $u[k]$ die gesamte PID-Stellgröße ohne Sättigung ist.

6 FSM - Finite State Machine

Finite State Machines oder FSM sind Zustandsmaschinen, welche je nach Zustand Ein- und Ausgänge anders behandeln. Wichtig dabei sind die Übergangsbedingungen, zu welchen der Zustand wechselt. Dargestellt werden sie in Zustandsdiagrammen. Ein einfaches Diagramm:



In ST ist es üblich, dass jeweils ein Enumerationstyp generiert wird, welcher die Zustände im code lesbarer macht:

```

1 TYPE eSystemState:
2 (
3     none,
4     init,
5     run,
6     error
7 )
8 END_TYPE
  
```

Im Programm geht das Ganze dann wie folgt von statthen:

```

1 PROGRAM
2 VAR
3     state : eSystemState;
4     iTemp : REAL;
5 END_VAR
6
7 CASE state OF
8     init:
9         // handle init
10        ....
11        // next state
12        IF iTemp >= 20 THEN
13            state := run;
14        END_IF
15     run:
16         // handle run
17         ....
  
```

```

18 // next state
19 IF iTemp > 100 THEN
20     // overtemperature protection
21     state := error;
22 END_IF
23
24 error:
25     // handle error
26     ....
27     IF bReset = TRUE THEN
28         state := init;
29     END_IF
30
31 ELSE
32     state := init;
33 END_CASE
  
```

7 Beispiel Code

Ein paar Beispiel Codes können hier entnommen werden.

7.1 Flankendetektor

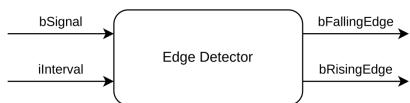
Damit ein Button nur einmal als gedrückt wahrgenommen wird, muss eine Flankendetektion implementiert werden. So kann auch darauf reagiert werden, wenn die Taste losgelassen wird. Für die Lesbarkeit, wird ein Enumerationstyp definiert.

7.1.1 Struct

```

1 // fuer die Button Flanken
2 TYPE sButtonState :
3 STRUCT
4     bState : BOOL;
5     bRisingEdge : BOOL;
6     bFallingEdge : BOOL;
7 END_STRUCT
8 END_TYPE
  
```

7.1.2 Funktionsblock



```

1 FUNCTION_BLOCK EdgeDetector
2 VAR_INPUT
3   bSignal : BOOL;
4   iInterval : INT;
5 END_VAR
6 VAR_OUTPUT
7   bFallingEdge : BOOL;
8   bRisingEdge : BOOL;
9 END_VAR
10 VAR
11   bPrevSignal : BOOL;
12   state : eStateEdgeDetect;
13   interval : INT;
14 END_VAR
15
16 IF bPrevSignal AND NOT bSignal THEN
17   // falling edge detected
18   bFallingEdge := 1;
19   bRisingEdge := 0;
20   state := eStateEdgeDetect.one_press;
21 ELSIF NOT bPrevSignal AND bSignal THEN
22   // rising edge detected
23   bFallingEdge := 0;
24   bRisingEdge := 1;
25   state := eStateEdgeDetect.one_press;
26 ELSE
27   // edge is nor falling or rising
28   bFallingEdge := 0;
29   bRisingEdge := 0;
30 END_IF
31
32 // store signal for next cycle
33 bPrevSignal := bSignal;
  
```

7.2.2 System Modus Enum

```

1 TYPE eSystem_Modus :
2 (
3   System_Init := 0,
4   System_Run := 1
5 );
6 END_TYPE
  
```

7.2.3 Funktionsblock

```

1 FUNCTION_BLOCK fbPID
2 VAR_INPUT
3   ref      : REAL;
4   y        : REAL;
5   Mode     : eSystem_Modus;
6   T        : REAL;
7   sPID_Par : sPID_Params;
8 END_VAR
9 VAR_IN_OUT
10
11 END_VAR
12 VAR_OUTPUT
13   u      : REAL;
14   usat  : REAL;
15 END_VAR
16 VAR
17   rParamI  : REAL;
18   rParamD1 : REAL;
19   rParamD2 : REAL;
20   rE       : REAL;
21   rEm     : REAL;
22   rUp     : REAL;
23   rUi    : REAL;
24   rUim   : REAL;
25   rUd    : REAL;
26   rUdm   : REAL;
27   u_arw  : REAL;
28 END_VAR
29
30 rE := ref - y;      // momentary error
31 CASE Mode OF
32   eSystem_Modus.System_Init:
33     // Parameter for I-Part
34     rParamI := sPID_Par.Kp*T / sPID_Par.Ti;
35     // Parameter for D-Part
36     rParamD1 := (
37       sPID_Par.N*sPID_Par.Kp*sPID_Par.Td ) /
38       (sPID_Par.Td + sPID_Par.N*T);
39     rParamD2 := ( sPID_Par.Td ) / (sPID_Par.Td +
40       sPID_Par.N*T);
41     u := 0;
42
43   eSystem_Modus.System_Run:
44     // P - Part
45     rUp := sPID_Par.Kp * rE;
46     // I - Part
47     rUi := rUim + rParamI * rE;
48     // D - Part
49     rUd := rParamD1 * (rE - rEm) + rParamD2 * rUdm;
50     u := rUp + rUi + rUd;
51     // saturating u if out of bounds
52     usat := LIMIT(u, sPID_Par.umin, sPID_Par.umax);
53     IF sPID_Par.ARW THEN
54       rUi := rUi + (1.0 / sPID_Par.Tr) * (usat -
55       u);
  
```

7.2 PID-Regler

7.2.1 Parameter Struct

```

1 TYPE sPID_Params :
2 STRUCT
3   Kp      : REAL; // Proportional Faktor
4   Ti      : REAL; // Nachstellzeit
5   Td      : REAL; // Vorhaltzeit
6   Tr      : REAL; // ARW Zeitkonstante
7   N       : UINT; // Filterkonstante
8   ARW    : BOOL; // ARW Enable/Disable
9   umin   : REAL; // usat minimal
10  umax   : REAL; // usat maximal
11 END_STRUCT
12 END_TYPE
  
```

```

52     END_IF
53     rUim := rUi;
54     rUdm := rUd;
55     rEm := rE;
56 END_CASE

```

7.3 PT1 Prozess

```

1 FUNCTION_BLOCK PT1
2 VAR_INPUT
3   signal : REAL;
4   fg : REAL;
5 END_VAR
6 VAR_OUTPUT
7   out : REAL;
8 END_VAR
9 VAR
10  T : REAL;
11  tau : REAL;
12  prev_out : REAL;
13 END_VAR
14
15 tau := 1/(2*pi*fg);
16 T:= 0.01;
17
18 out := (T/(T+tau))*signal + (tau/(T+tau))*prev_out;
19
20 prev_out := out;

```

7.4 PT2 Prozess

7.4.1 Parameter Struct

```

1 TYPE dPT1T2_Params :
2 STRUCT
3   K : REAL;
4   T1 : REAL;
5   T2 : REAL;
6 END_STRUCT
7 END_TYPE

```

7.4.2 Funktionsblock

```

1 FUNCTION_BLOCK PT1T2_Prozess
2 VAR_INPUT
3   u : REAL;
4   Modus : eSystem_Modus;
5   Param : dPT1T2_Params;
6   T : REAL;
7 END_VAR
8 VAR_OUTPUT
9   out : REAL;
10 END_VAR
11 VAR
12   b0 : REAL;
13   alph1 : REAL;
14   alph2 : REAL;
15   a1 : REAL;
16   a2 : REAL;
17   outm : REAL;
18   outmm : REAL;
19 END_VAR
20
21 CASE Modus OF
22   eSystem_Modus.System_Init:

```

```

23   // alphas
24   alph1 := T + Param.T1;
25   alph2 := T + Param.T2;
26   // funct. parameters
27   b0 := (Param.K * T * T) / (alph1*alph2);
28   a1 := -( (Param.T1/alph1) + (Param.T2/alph2) );
29   a2 := (Param.T1*Param.T2)/(alph1*alph2);
30
31   eSystem_Modus.System_Run:
32   out := -a1*outm - a2*outmm + b0 * u;
33   outmm := outm;
34   outm := out;
35 END_CASE

```

8 Matlab-Befehle

Für die Auslegung und modellierung wurde in IAUT Matlab verwendet. Dabei wurden Tools wie der 'PID-Tuner' oder 'Siso-Tool' für die Auslegung der Regler eingesetzt. Grundsätzlich folgt der Reglerentwurf diesem Schema:

1. Prozess messen
2. Prozessmodellierung (Sprungantwort parametrisieren, etc.)
3. Regler mit Tool auslegen

Kennengelernte Matlab Befehle sind:

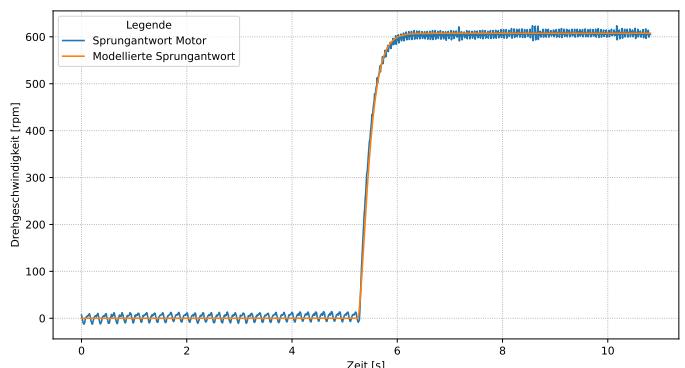
```

1 margin(C*H/(1+H*C)) // Phasen und Gain Reserve
2           closed loop
3
4 num = [p0 0 0];
5 den = [p1 p2 p3];
6 H = tf(num,den,Ts); // Zeitdiskrete
7           Uebertragungsfunktion
8
9 y = lsim(H, U, t/1000); // Sprungantwort simulieren
10
11 stepz(num,den,Ts) // Sprungantwort
12 freqz(num,den) // Frequenzgang

```

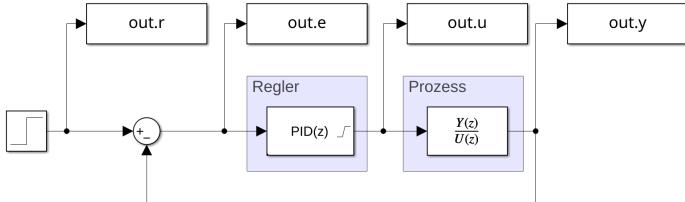
8.1 Sprungantwort modellieren

Um eine Sprungantwort zu modellieren, muss jeweils entschieden werden, wie der Sprung am besten simuliert werden kann. Der Sprung muss immer genutzt werden (bei 0 starten) aber darf beim Aufnehmen nicht bei 0 starten. Entsprechend: Motor Bsp 500 rpm zu 1500 rpm. Dann -500 rechnen, um Sprung zu modellieren. Ein Modell eines Sprunges kann so aussehen:



8.2 Simulink

Das Modell kann genau so nachgebaut werden, wie modelliert. Dann kann mittels diskreten PID-Regler den Regler ausgelegt werden.



9 Hilfsblätter

9.1 Laplace Transformation

C.3 Laplace-Transformation

Definition

zweiseitig:	einseitig:
$X(s) = \int_{-\infty}^{\infty} x(t) e^{-st} dt$	$X(s) = \int_0^{\infty} x(t) e^{-st} dt$
Konvergenzgebiet \mathcal{K} : $a < \operatorname{Re}\{s\} < b$	Konvergenzgebiet \mathcal{K} : $\operatorname{Re}\{s\} > a$

Eigenschaften und Rechenregeln

	Zeitbereich	Bildbereich	Konvergenz
Linearität	$c_1 x_1(t) + c_2 x_2(t)$	$c_1 X_1(s) + c_2 X_2(s)$	$\mathcal{K}_{x_1} \cap \mathcal{K}_{x_2}$
Faltung	$x(t) * y(t)$	$X(s) \cdot Y(s)$	$\mathcal{K}_x \cap \mathcal{K}_y$
Verschiebung	$x(t - t_0)$	$e^{-st_0} \cdot X(s)$	\mathcal{K}_x
Dämpfung	$e^{at} \cdot x(t)$	$X(s - a)$	$\mathcal{K}_x + \operatorname{Re}\{a\}$
lineare Gewichtung	$t \cdot x(t)$	$-\frac{d}{ds} X(s)$	\mathcal{K}_x
Differentiation	$\frac{d}{dt} x(t)$	$s \cdot X(s)$	\mathcal{K}_x
Integration	$\int_{-\infty}^t x(\tau) d\tau$	$\frac{1}{s} \cdot X(s)$	$\mathcal{K}_x \cap \{\operatorname{Re}\{s\} > 0\}$
Skalierung	$x(at)$	$\frac{1}{ a } \cdot X\left(\frac{s}{a}\right)$	$a \cdot \mathcal{K}_x$
konj. komplexes Signal	$x^*(t)$	$X^*(s^*)$	\mathcal{K}_x

Spezielle Eigenschaften der einseitigen Laplace-Transformation

Verschiebung links	$x(t + t_0), t_0 > 0$	$e^{st_0} \left[X(s) - \int_0^{t_0} x(t) \cdot e^{-st} dt \right]$
Verschiebung rechts	$x(t - t_0), t_0 > 0$	$e^{-st_0} \left[X(s) + \int_{-t_0}^0 x(t) \cdot e^{-st} dt \right]$
Differentiation	$\frac{d}{dt} x(t)$	$s \cdot X(s) - x(0^-)$
Anfangswertsatz	$x(0^+) = \lim_{s \rightarrow \infty} s \cdot X(s)$, falls $x(0^+)$ existiert	
Endwertsatz	$\lim_{t \rightarrow \infty} x(t) = \lim_{s \rightarrow 0} s \cdot X(s)$, falls $\lim_{t \rightarrow \infty} x(t)$ existiert	

Komplexe Umkehrformel der Laplace-Transformation

$$x(t) = \frac{1}{2\pi j} \int_{\sigma-j\infty}^{\sigma+j\infty} X(s) \cdot e^{st} ds, \quad \sigma \in \mathcal{K}$$

Rücktransformation durch Partialbruchzerlegung

$$X(s) = g_0 + \sum_i \frac{r_i}{s - \alpha_i} + \sum_i \sum_{l=1}^{m_i} \frac{\tilde{r}_{i,l}}{(s - \tilde{\alpha}_i)^l}, \quad g_i, \alpha_i, \tilde{\alpha}_i, r_i, \tilde{r}_{i,l} \in \mathbb{C}$$

kann für \mathcal{K} : $\operatorname{Re}\{s\} > a_0$ mit den Korrespondenzen 1, 4, 5 und 10 gliedweise zurücktransformiert werden.

Korrespondenzen der Laplace-Transformation

Nr.	$x(t)$	$X(s)$	\mathcal{K}
1	$\delta(t)$	1	$s \in \mathbb{C}$
2	$\varepsilon(t)$	$\frac{1}{s}$	$\operatorname{Re}\{s\} > 0$
3	$\rho(t) = t \cdot \varepsilon(t)$	$\frac{1}{s^2}$	$\operatorname{Re}\{s\} > 0$
4	$e^{at} \cdot \varepsilon(t)$	$\frac{1}{s-a}$	$\operatorname{Re}\{s\} > \operatorname{Re}\{a\}$
5	$\frac{1}{m!} e^{at} \cdot \varepsilon(t)$	$\frac{1}{(s-a)^{m+1}}$	$\operatorname{Re}\{s\} > \operatorname{Re}\{a\}$
6	$\sin(\omega_0 t) \cdot \varepsilon(t)$	$\frac{\omega_0}{s^2 + \omega_0^2}$	$\operatorname{Re}\{s\} > 0$
7	$\cos(\omega_0 t) \cdot \varepsilon(t)$	$\frac{s}{s^2 + \omega_0^2}$	$\operatorname{Re}\{s\} > 0$
8	$\sin(\omega_0 t + \varphi_0) \cdot \varepsilon(t)$	$\frac{s \cdot \sin(\varphi_0) + \omega_0 \cdot \cos(\varphi_0)}{s^2 + \omega_0^2}$	$\operatorname{Re}\{s\} > 0$
9	$\delta(t - t_0)$	e^{-st_0}	$s \in \mathbb{C}$

Spezielle Korrespondenzen zur Rücktransformation konj. komplexer Polpaare

10	$2 r e^{\operatorname{Re}\{\alpha\}t} \cos(\operatorname{Im}\{\alpha\}t + \angle r) \varepsilon(t)$	$\frac{r}{s-\alpha} + \frac{r^*}{s-\alpha^*}$	$\operatorname{Re}\{s\} > \operatorname{Re}\{\alpha\}$
11	$e^{at} \cdot \frac{\cos(\omega_0 t + \varphi_0)}{\cos(\varphi_0)} \cdot \varepsilon(t)$	$\frac{s-d}{s^2 - bs + c}, \quad c > \frac{b^2}{4}$ $a = \frac{b}{2}, \quad \omega_0 = \sqrt{c - \frac{b^2}{4}}, \quad \varphi_0 = \arctan\left(\frac{2d-b}{\sqrt{4c-b^2}}\right)$	$\operatorname{Re}\{s\} > \frac{b}{2}$

9.2 z -Transformation

Definition

zweiseitig:	einseitig:
$X(z) = \sum_{k=-\infty}^{\infty} x[k] z^{-k}$	$X(z) = \sum_{k=0}^{\infty} x[k] z^{-k}$
Konvergenzgebiet \mathcal{K} : $a < z < b$	Konvergenzgebiet \mathcal{K} : $ z > a$

Inverse z -Transformation

$$x[k] = \frac{1}{2\pi j} \oint_{\mathcal{C}} X(z) z^{k-1} dz = \sum_{\alpha_i \in \mathcal{A}} \operatorname{Res}\{X(z) \cdot z^{k-1}; \alpha_i\}$$

\mathcal{C} : pos. orientierte Kurve in \mathcal{K} . \mathcal{A} : Menge aller von \mathcal{C} umschlossenen Pole.

Eigenschaften und Rechenregeln

	Zeitbereich	Bildbereich	Konvergenz
Linearität	$c_1 x_1[k] + c_2 x_2[k]$	$c_1 X_1(z) + c_2 X_2(z)$	$\mathcal{K}_{x_1} \cap \mathcal{K}_{x_2}$
Faltung	$x[k] * y[k]$	$X(z) \cdot Y(z)$	$\mathcal{K}_x \cap \mathcal{K}_y$
Verschiebung	$x[k - k_0]$	$z^{-k_0} X(z)$	\mathcal{K}_x
Dämpfung	$a^k \cdot x[k]$	$X\left(\frac{z}{a}\right)$	$ a \cdot \mathcal{K}_x$
lineare Gewichtung	$k \cdot x[k]$	$-z \cdot \frac{d}{dz} X(z)$	\mathcal{K}_x
konj. komplexes Signal	$x^*[k]$	$X^*(z^*)$	\mathcal{K}_x
Zeitinversion	$x[-k]$	$X\left(\frac{1}{z}\right)$	$1/\mathcal{K}_x$
diskrete Ableitung	$x[k] - x[k - 1]$	$X(z) \cdot \frac{z-1}{z}$	\mathcal{K}_x
diskrete Integration	$\sum_{i=\infty}^k x[i]$	$X(z) \cdot \frac{z}{z-1}$	$\mathcal{K}_x \cap \{ z > 1\}$
periodische Fortsetzung	$\sum_{i=0}^{\infty} x[k - iN_p]$	$X(z) \cdot \frac{1}{1-z^{-N_p}}$	$\mathcal{K}_x \cap \{ z > 1\}$
Upsampling	$x\left[\frac{k}{N}\right]$	$X(z^N)$	$\sqrt[N]{\mathcal{K}_x}$

Spezielle Eigenschaften der einseitigen z -Transformation

Verschiebung links	$x[k + k_0], k_0 > 0$	$z^{k_0} X(z) - \sum_{i=0}^{k_0-1} x[i] z^{k_0-i}$
Verschiebung rechts	$x[k - k_0], k_0 > 0$	$z^{-k_0} X(z) + \sum_{i=-k_0}^{-1} x[i] z^{-k_0-i}$
Anfangswertsatz	$x[0] = \lim_{z \rightarrow \infty} X(z), \text{ falls Grenzwert existiert}$	
Endwertsatz	$\lim_{k \rightarrow \infty} x[k] = \lim_{z \rightarrow 1} (z-1) X(z), \text{ falls } X(z) \text{ nur Pole mit } z < 1 \text{ oder bei } z = 1$	

Rücktransformation durch Partialbruchzerlegung

Die Partialbruchzerlegung von $\tilde{X}(z) = \frac{X(z)}{z}$ führt auf die Form:

$$X(z) = \sum_{i \geq 1} g_i \cdot z^i + \sum_i \frac{z \cdot r_i}{z - \alpha_i} + \sum_i \sum_{l=1}^{m_i} \frac{z \cdot \tilde{r}_{i,l}}{(z - \tilde{\alpha}_i)^l}, \quad g_i, \alpha_i, \tilde{\alpha}_i, r_i, \tilde{r}_{i,l} \in \mathbb{C},$$

welche für $\mathcal{K} : |z| > r_0$ mit den Korrespondenzen 2, 5, 6 und 12 gliedweise zurücktransformiert werden kann.

Korrespondenzen der z -Transformation

Nr.	$x[k]$	$X(z)$	\mathcal{K}
1	$\delta[k]$	1	$z \in \mathbb{C}$
2	$\delta[k - k_0]$	z^{-k_0}	$0 < z < \infty$
3	$\varepsilon[k]$	$\frac{z}{z-1}$	$ z > 1$
4	$k \cdot \varepsilon[k]$	$\frac{z}{(z-1)^2}$	$ z > 1$
5	$a^k \cdot \varepsilon[k]$	$\frac{z}{z-a}$	$ z > a $
6	$\binom{k}{m} a^{k-m} \cdot \varepsilon[k]$	$\frac{z}{(z-a)^{m+1}}$	$ z > a $
7	$\sin(\Omega_0 k) \cdot \varepsilon[k]$	$\frac{z \cdot \sin(\Omega_0)}{z^2 - 2z \cos(\Omega_0) + 1}$	$ z > 1$
8	$\cos(\Omega_0 k) \cdot \varepsilon[k]$	$\frac{z \cdot [z - \cos(\Omega_0)]}{z^2 - 2z \cos(\Omega_0) + 1}$	$ z > 1$
9	$d^k \cdot \varepsilon[-k-1]$	$-\frac{z}{z-a}$	$ z < a $
10	$a^{ k }, a < 1$	$\frac{z \cdot \left(a - \frac{1}{a}\right)}{(z-a)(z-\frac{1}{a})}$	$ a < z < \frac{1}{a} $
11	$\frac{1}{k!} \cdot \varepsilon[k]$	$e^{\frac{1}{z}}$	$ z > 0$

Spezielle Korrespondenzen zur Rücktransformation konj. komplexer Polpaare

12	$2 r \alpha ^k \cos(\angle \alpha \cdot k + \angle r) \cdot \varepsilon[k]$	$\frac{z \cdot r}{z - \alpha} + \frac{z \cdot r^*}{z - \alpha^*}$	$ z > \alpha$
13	$a^k \cdot \frac{\cos(\Omega_0 k + \varphi_0)}{\cos(\varphi_0)} \cdot \varepsilon[k]$	$\frac{z(z-d)}{z^2 - bz + c}, \quad c > \frac{b^2}{4}$	$ z > \sqrt{c}$