Zusammenfassung IAS

[Automatisierung 3](#_Toc155367896)

[Datentypen 3](#_Toc155367897)

[Zahlenkonstanten 3](#_Toc155367898)

[Array 3](#_Toc155367899)

[Operatoren 4](#_Toc155367900)

[Übersicht IEC Operatoren 5](#_Toc155367901)

[Schleifen 7](#_Toc155367902)

[IF / ELSE 8](#_Toc155367903)

[Typenkonvertierung 8](#_Toc155367904)

[CASE 9](#_Toc155367905)

[STRUCT 9](#_Toc155367906)

[ENUM 9](#_Toc155367907)

[GVL 10](#_Toc155367908)

[Funktionsbaustein 10](#_Toc155367909)

[Funktion 11](#_Toc155367910)

[Programm 11](#_Toc155367911)

[call by Value 12](#_Toc155367912)

[call by Reference 12](#_Toc155367913)

[Timer 13](#_Toc155367914)

[State Maschine Beispiel 14](#_Toc155367915)

[Inputs and Outputs (GVL) 14](#_Toc155367916)

[States (Enum) 14](#_Toc155367917)

[Main 14](#_Toc155367918)

[State Maschine Ampel Beispiel 15](#_Toc155367919)

[PID-Regler 17](#_Toc155367920)

[Funktionen sin/square/pwm 18](#_Toc155367921)

[EdgeDetector 18](#_Toc155367922)

[Methoden 19](#_Toc155367923)

[Property 20](#_Toc155367924)

[PT1 (Rückwärtsdifferenz) 20](#_Toc155367925)

[PT2 (Rückwärtsdifferenz) 21](#_Toc155367926)

[Theorie 21](#_Toc155367927)

[Task 22](#_Toc155367928)

[Export Funktionen 22](#_Toc155367929)

[Remanenz 23](#_Toc155367930)

[Feldbusse 23](#_Toc155367931)

[EN 61131-3 23](#_Toc155367932)

[Verbinden von SPS Klemmen zu Programm 23](#_Toc155367933)

[Programmiersprachen 24](#_Toc155367934)

[Digitale Reglung 25](#_Toc155367935)

[Bestimmung der Regelparameter: Experimentell PT1 – Glied 25](#_Toc155367936)

[Einstellregeln 25](#_Toc155367937)

[Robustheit 25](#_Toc155367938)

[DIGITALER REGLER 26](#_Toc155367939)

[AD-Wandler 26](#_Toc155367940)

[SHANNON ABTASTTHEOREM 26](#_Toc155367941)

[ANTI ALIASING FILTER 27](#_Toc155367942)

[DA – Wandler 27](#_Toc155367943)

[Regler Diskretisieren 27](#_Toc155367944)

[P-Anteil 29](#_Toc155367945)

[I-Anteil 29](#_Toc155367946)

[D-Anteil 29](#_Toc155367947)

[D-Anteil mit Filter 30](#_Toc155367948)

[Saturation 31](#_Toc155367949)

[Pseudocode 31](#_Toc155367950)

[Differenzengleichung 31](#_Toc155367951)

[Z-Transformation 32](#_Toc155367952)

[Eigenschaften der Z-Transformation 32](#_Toc155367953)

[Impulsantwort beschreibt das System 32](#_Toc155367954)

[Die Ordnung eines Systems 32](#_Toc155367955)

[Regelstrecke Diskretisieren 33](#_Toc155367956)

[Störungseinfluss 34](#_Toc155367957)

[Grafischer vergleich Laplace und Z-Transformation 34](#_Toc155367958)

[Fourier Transformation 35](#_Toc155367959)

[Diskreter Frequenzgang 36](#_Toc155367960)

[Beispiel 36](#_Toc155367961)

[Nsipre Befehle: 40](#_Toc155367962)

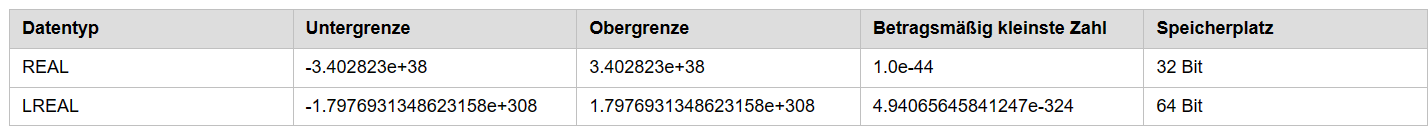
# Automatisierung

## Datentypen

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung  
  
Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

Automatisch generierte Beschreibung

Gleitkommazahlen

## Zahlenkonstanten

Ein Bild, das Text, Reihe, Schrift, Screenshot enthält.

Automatisch generierte Beschreibung

## Array

1-dimensionales Array:

aCounter : ARRAY[0..9] OF INT;

aCounter : ARRAY[0..9] OF INT := [0, 10, 20, 30, 40, 50, 60, 70, 80, 90];

nRes := aCounter[2]; (\*nRes wird 20 zugewiesen\*)

aCounter[5] := 30; (\*30 wird an Position 5 zugewiesen\*)

2-dimensionales Array:

aCardGame : ARRAY[1..2, 3..4] OF INT;

aCardGame : ARRAY[1..2, 3..4] OF INT := [2(10),2(20)];   
// Short notation for [10, 10 | 20, 20]

nLocal1 := aCardGame[1, 3]; // Assignment of 10  
nLocal2 := aCardGame[2, 4]; // Assignment of 20

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

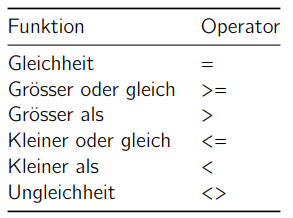
Automatisch generierte Beschreibung

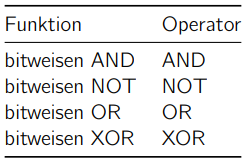
Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

1..3 Spalten, 1..2 Zeilen

## Operatoren





Modulo:  
nVar := 9 MOD 2;

Sizeof: (Grösse in Byte)  
aArr1 : ARRAY[0..4] OF INT;   
nVar  : INT;

nVar := SIZEOF(aArr1); (\*nVar := USINT 10\* weil 5\*2Byte = 10Byte)

Shift Left:  
nRes := SHL( 2#0000000001000101 , 2); (\*Result is 2#0000000100010100\*)

Shift Right:   
nRes := SHR( 2#0000000001000101 , 2); (\*Result is 2#0000000000010001\*)

AND:  
nVar := 2#1001\_0011 AND 2#1000\_1010; (\*Result is 2#1000\_0010\*)

OR:  
nVar := 2#1001\_0011 OR 2#1000\_1010; (\*Result is 2#1001\_1011\*)

NOT:  
nVar := NOT 2#1001\_0011; (\*Result is 2#0110\_1100\*)

XOR:  
nVar := 2#1001\_0011 XOR 2#1000\_1010; (\*Result is 2#0001\_1001\*)

Konvertierung:  
nVar1 := REAL\_TO\_INT(1.5); (\*Result is 2\*)

Equal:  
bVar := 40 = 40; (\*True\*)

Grösser oder gleich:  
bVar := 60 >= 40; (\*True\*)

Grösser als:  
bVar := 20 > 30; (\*False\*)

Kleiner oder gleich:  
bVar := 20 <= 30; (\*True\*)

Kleiner als:  
bVar := 20 < 30; (\*True\*)

Nicht gleich:  
bVar := 40 <> 40; (\*False\*)

### Übersicht IEC Operatoren

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Reihe, Zahl, Screenshot enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Zahl, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

## Schleifen

FOR <INT\_Var> := <INIT\_VALUE> TO <END\_VALUE> {BY  
<stepsize>} DO

For Schleife mit Continue: Bei der For Schleife machts automatisch nCounter + 1 oder der eingestellte Wert

FOR nCounter :=1 TO 5 BY 1 DO   
    nInt1:=nInt1/2;   
    IF nInt1=0 THEN   
        CONTINUE; (\* to avoid a division by zero \*)   
    END\_IF   
    nVar1:=nVar1/nInt1; (\* executed, if nInt1 is not 0 \*)   
END\_FOR;   
nRes:=nVar1;

While Schleife:

WHILE nCounter > 0 DO  
    nVar1 := nVar1\*2  
    nCounter := nCounter-1;  
END\_WHILE;

Repeat Schleife: (Wird mind. 1x durchgeführt)

REPEAT  
    nVar1 := nVar1\*2;  
    nCounter := nCounter-1;  
UNTIL  
    nCounter = 0  
END\_REPEAT;

**EXIT**: Schleifendurchführungen können mit **EXIT;** sofort abgebrochen werden  
Achtung: Keine grossen Loops einbauen => Taktzeit des Tasks muss eingehalten werden können!

## IF / ELSE

IF nTemp < 17 THEN  
    bHeatingOn  := TRUE;  
ELSIF nTemp > 25 THEN  
    bOpenWindow := TRUE;  
ELSE  
    bHeatingOn  := FALSE;  
    bOpenWindow := FALSE;  
END\_IF;

## Typenkonvertierung

<type>\_TO\_BOOL

bVar := BYTE\_TO\_BOOL(2#11010101); // True

BOOL\_TO\_<type>

nVar := BOOL\_TO\_INT(TRUE); // 1

<INT type>\_TO\_<INT type>

nVar := INT\_TO\_SINT(4223); // 127

REAL/LREAL\_TO\_<type> (Es wird normal gerundet)

nVar1 := REAL\_TO\_INT(1.5); // 2

nVar2 := REAL\_TO\_INT(1.4); // 1

nVar1 := REAL\_TO\_INT(-1.5); // -2

TIME/TOD\_TO\_<type>

nVar := TIME\_TO\_DWORD(T#5m); // 300000

## CASE

CASE nVar OF   
     1,5 : bVar1 := TRUE;   
         bVar3 := FALSE;   
  
     2 : bVar2 := FALSE;   
         bVar3 := TRUE;   
  
10..20 : bVar1 := TRUE;   
         bVar3 = TRUE;   
ELSE   
         bVar1 := NOT bVar1;   
         bVar2 := bVar1 OR bVar2;   
END\_CASE;

Else ist optional

## STRUCT

Eine Struktur ist ein benutzerdefinierter Datentyp und fasst mehrere Variablen mit beliebigen Datentypen zu einer logischen Einheit zusammen. Die innerhalb einer Struktur deklarierten Variablen werden als Komponenten bezeichnet.

**Aufbau:**  
TYPE Param\_Struct:  
STRUCT  
 K : REAL;  
 T\_1 : REAL;  
 T\_2 : REAL;  
END\_STRUCT  
END\_TYPE

**Initialisieren:**VAR  
 PT1Param : Param\_Struct;  
END\_VAR

**Wert zuweisen:**  
PT1T2Parameter.K := 10;  
PT1T2Parameter.T\_1 := 0.1;

## ENUM

Eine Enumeration oder Aufzählung ist ein benutzerdefinierter Datentyp, der sich aus einer kommaseparierten Reihe von Komponenten, auch Enumerationswerte genannt, zusammensetzt, um benutzerdefinierte Variablen zu deklarieren. Die Deklaration einer Enumeration nehmen Sie in einem DUT-Objekt vor, das Sie über den Befehl Hinzufügen > DUT im Kontextmenü des SPS-Projektbaums im Projekt anlegen.

**Aufbau:**  
TYPE System\_Modus\_Enum :  
(  
 System\_Init := 0,  
 System\_Run  
);  
END\_TYPE

**Initialisieren:**VAR  
 Modus : System\_Modus\_Enum := System\_Modus\_Enum.System\_Init;  
END\_VAR

**Wert zuweisen:**  
Modus := System\_Modus\_Enum.System\_Run;

## GVL

**Wert zuweisen:**  
iTime := GVL.T;  
iInt := GVL.Sensor\_input\_V;  
GVL.Enable\_Motor := TRUE;

**Aufbau:**  
VAR\_GLOBAL  
 T : REAL := 0.01;  
 Sensor\_input\_V AT%I\*: INT;  
 Enable\_Motor AT%Q\* : BOOL;  
END\_VAR

## Funktionsbaustein

Ein Funktionsbaustein ist eine POU, die bei der Ausführung einen oder mehrere Werte liefert. Die Werte der Ausgabevariablen und der internen Variablen bleiben nach einer Ausführung bis zur nächsten erhalten. Dies bedeutet, dass der Funktionsbaustein bei mehrmaligem Aufruf mit denselben Eingabevariablen nicht unbedingt dieselben Ausgabewerte liefert. Ausgang kann auch direkt beim Aufruf mit **=>** einer Variabel zugewiesen werden.

**Initialisieren:**VAR  
 FB\_PWM : fb\_PWM;

y\_t\_pwm : REAL;  
END\_VAR

**Aufrufen und Wert abfragen:**  
FB\_PWM(Tastgrad := 0.7, y\_off := 0, w:=w, y\_min := 0, y\_max := 200); //auch möglich y\_t => y\_t\_pwm  
y\_t\_pwm := FB\_PWM.y\_t;

**Aufbau:**  
FUNCTION\_BLOCK fb\_PWM  
VAR\_INPUT  
 Tastgrad: REAL;  
 y\_min : REAL;  
 y\_max : REAL;  
 y\_off : REAL;  
 w : REAL;  
END\_VAR  
VAR\_OUTPUT  
 y\_t : REAL;  
END\_VAR  
VAR  
 timer : TON;  
 toggleTime : TIME;  
END\_VAR

Ein Bild, das Text, Elektronik, Screenshot, Display enthält.

Automatisch generierte BeschreibungVar Output andere Methode

## Funktion

Eine Funktion ist eine POU, die bei der Ausführung genau ein Datenelement liefert und dessen Aufruf in textuellen Sprachen als Operator in Ausdrücken vorkommen kann. Das Datenelement kann auch ein Array oder eine Struktur sein.

**Aufbau:**  
FUNCTION f\_Aint\_to\_A\_HTRS10 : REAL  
VAR\_INPUT  
 iInt : INT;  
 Offset: REAL;  
END\_VAR  
VAR  
 Aufloesung : INT := 32767;  
 Strom\_ma : REAL;  
 Aufloesung\_Sensor : REAL := 2.5;  
END\_VAR

**Program:**Strom\_ma := INT\_TO\_REAL(iInt)\*20/Aufloesung;  
f\_Aint\_to\_A\_HTRS10 := Aufloesung\_Sensor\*(Strom\_ma-(12+Offset));

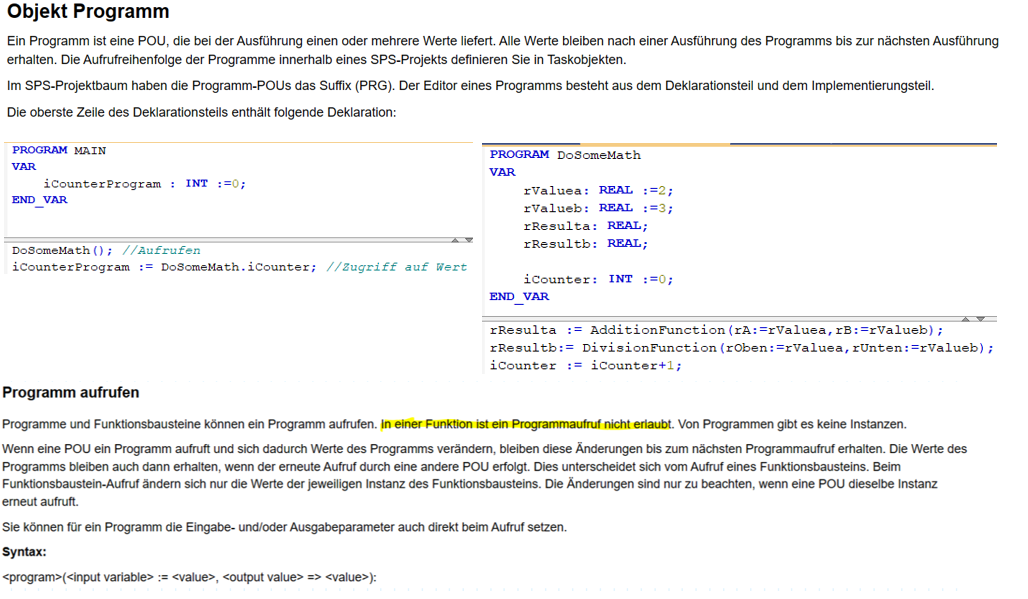
**Aufrufen und Wert abfragen:**  
x\_A\_10 := f\_Aint\_to\_A\_HTRS10(iInt:= GVL.Sensor\_input\_A\_10, Offset := -0.1);

Funktionen haben keine interne Statusinformation, das bedeutet, dass Funktionen die Werte ihrer Variablen nicht bis zum nächsten Aufruf speichern. Aufrufe einer Funktion mit denselben EingabevariablenWerten liefern immer denselben Ausgabewert. Deshalb dürfen Funktionen keine globalen Variablen und Adressen verwenden!

## Programm

Ein Programm ist eine POU, die bei der Ausführung einen oder mehrere Werte liefert. Alle Werte bleiben nach einer Ausführung des Programms bis zur nächsten Ausführung erhalten. Die Aufrufreihenfolge der Programme innerhalb eines SPS-Projekts definieren Sie in Taskobjekten.

Wenn eine POU ein Programm aufruft und sich dadurch Werte des Programms verändern, bleiben diese Änderungen bis zum nächsten Programmaufruf erhalten. Die Werte des Programms bleiben auch dann erhalten, wenn der erneute Aufruf durch eine andere POU erfolgt. Dies unterscheidet sich vom Aufruf eines Funktionsbausteins. Beim Funktionsbaustein-Aufruf ändern sich nur die Werte der jeweiligen Instanz des Funktionsbausteins. Die Änderungen sind nur zu beachten, wenn eine POU dieselbe Instanz erneut aufruft.



## call by Value

Es muss ein Rückgabewert übergeben werden und muss zusätzlichen Speicher allozieren werden.

## call by Reference

VAR  
    pSample : POINTER TO INT; //pSample := ADR(nVar1);   
END\_VAR

//Oder   
VAR  
    refInt  : REFERENCE TO INT; //refInt REF= nVar1;   
END\_VAR

FUNCTION\_BLOCK FB\_Sample  
VAR\_INPUT  
    refInput1  : REFERENCE TO INT;   
    refInput2  : REFERENCE TO INT;  
END\_VAR

PROGRAM MAIN  
VAR  
    fbSample   : FB\_Sample;  
    n1         : INT;  
    n2         : INT;  
END\_VAR

fbSample.refInput1 REF= n1; // refInput1 points now to n1  
fbSample(refInput2   := n2); // refInput2 has got the value n2

Es muss kein Rückgabewert übergeben werden und **kein** zusätzlichen Speicher allozieren werden  [Zuweisungsoperators REF=](https://infosys.beckhoff.com/content/1031/tc3_plc_intro/2529458827.html#4415939851) setzen

## Timer

**TON => EINSCHALTVERZÖGERUNG**

|  |  |
| --- | --- |
| TON 2: | VAR\_INPUT  IN : BOOL; (\* starts timer with rising edge, resets timer with falling edge \*)  PT : TIME; (\* time to pass, before Q is set \*)  END\_VAR VAR\_OUTPUT  Q : BOOL; (\* is TRUE, PT seconds after IN had a rising edge \*)  ET : TIME; (\* elapsed time \*)  END\_VAR |

**TOF => AUSSCHALTVERZÖGERUNG**

|  |  |
| --- | --- |
| TOF 2: | VAR\_INPUT  IN : BOOL; (\* starts timer with falling edge, resets timer with rising edge \*)  PT : TIME; (\* time to pass, before Q is set \*)  END\_VAR VAR\_OUTPUT  Q : BOOL; (\* is FALSE, PT seconds after IN had a falling edge \*)  ET : TIME; (\* elapsed time \*)  END\_VAR |

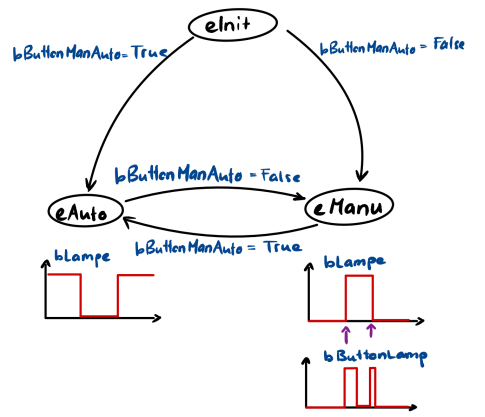
**Beispielprogramm:**

PROGRAM MAIN  
VAR  
 timer: TON;  
 bBlinker : BOOL;  
END\_VAR

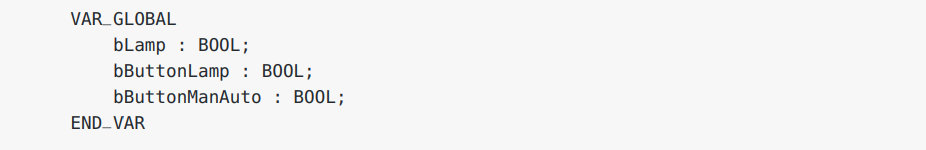
------------------------------------------------------------------------------

IF NOT (timer.Q) THEN  
 timer.IN := TRUE;  
ELSE  
 timer.IN := FALSE;  
 bBlinker := NOT (bBlinker);  
END\_IF  
timer.PT := T#2S;  
timer();

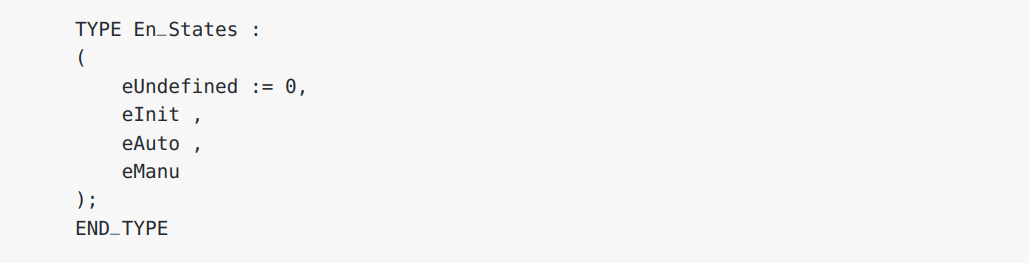
## State Maschine Beispiel



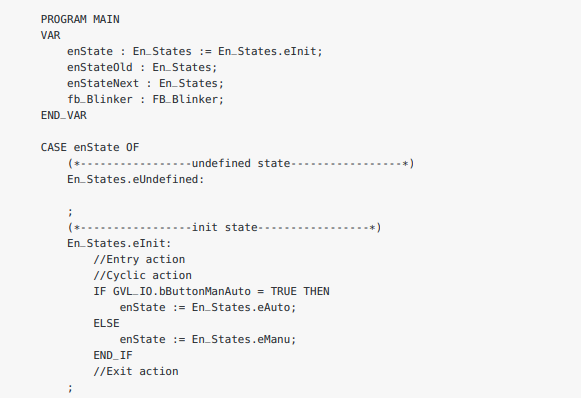
### Inputs and Outputs (GVL)



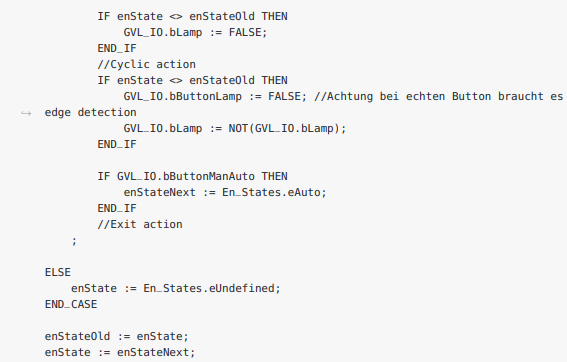
### States (Enum)



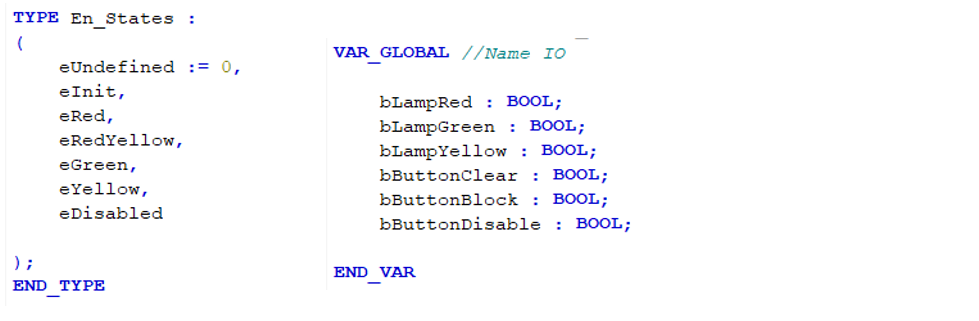
### Main







## State Maschine Ampel Beispiel



Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

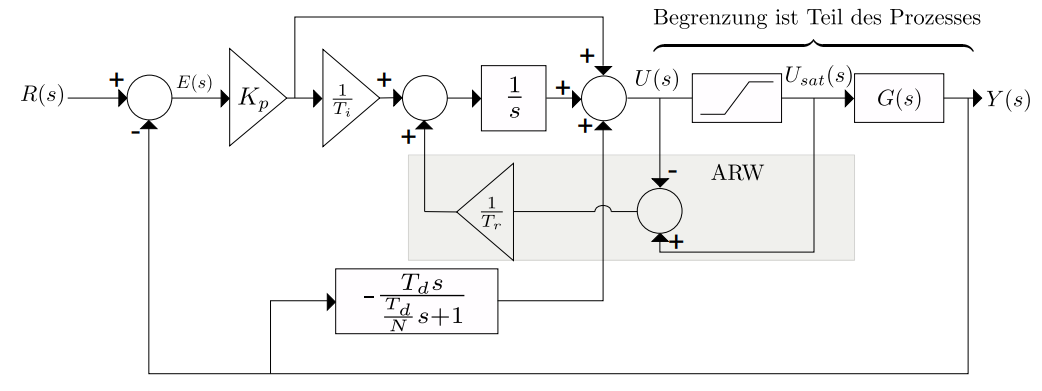
Automatisch generierte BeschreibungEin Bild, das Diagramm, Kreis, Reihe, Schrift enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

## PID-Regler



FUNCTION\_BLOCK PID\_Regler

VAR\_INPUT

ref : REAL;

y :REAL;

Modus:System\_Modus\_Enum;

Param : PID\_Parameter\_Strukt;

T : REAL:=0.01;

END\_VAR

VAR\_OUTPUT

u : REAL;

END\_VAR

VAR

up : REAL;

ui : REAL;

ud : REAL;

u\_oS : REAL; //u ohne Saturation

ui\_old : REAL;

ud\_old : REAL;

d1 : REAL;

d2 : REAL;

i: REAL;

error:REAL;

error\_old:REAL;

END\_VAR

CASE Modus OF

(\*-----------------Init state-----------------\*)

System\_Modus\_Enum.System\_Init:

//Entry action

//Cyclic action

i:=Param.Kp/Param.Ti/2\*T;

d1 := (Param.Kp\*Param.Td\*Param.N)/(Param.Td+Param.N\*T);

d2 := (Param.Td)/(Param.Td+Param.N\*T);

//Exit action

;

(\*-----------------Run state-----------------\*)

System\_Modus\_Enum.System\_Run:

//Entry action

error:=ref-y;

//Cyclic action

//P-Anteil

up:=Param.Kp\*error;

//D-Anteil

ud:=d1\*(error-error\_old)+d2 \* ud\_old;

//PID Regler

u\_oS:=up+ui+ud;

//Saturation

IF u\_oS > Param.umax THEN

u := Param.umax;

ELSIF u\_oS < Param.umin THEN

u := Param.umin;

ELSE

u := u\_oS;

END\_IF

//Anti Windup

IF Param.ARW THEN

//I-Anteil mit anti Windup

ui:=ui\_old +i\*(error+error\_old) + (T/Param.Tr) \* (u - u\_oS);

ELSE

//I-Anteil ohne anti Windup

ui:=ui\_old +i\*(error+error\_old);

END\_IF

// Werte verzögerung [k-1]

ui\_old:=ui;

ud\_old:=ud;

error\_old := error;

//Exit action

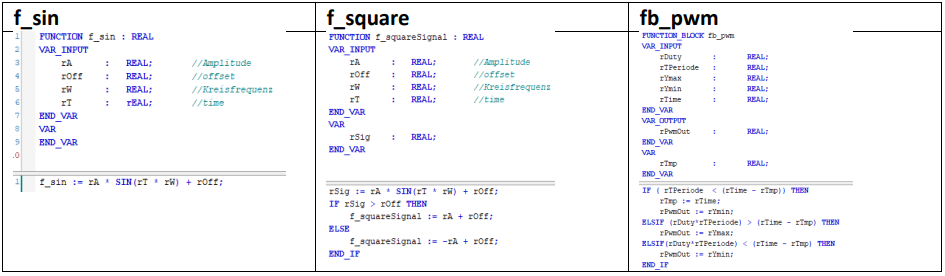
;

ELSE

Modus :=System\_Modus\_Enum.System\_Init;

END\_CASE;

## Funktionen sin/square/pwm



## EdgeDetector

FUNCTION\_BLOCK EdgeDetector

VAR\_INPUT

inputValue : BOOL;

timerDelayTime : TIME := T#10MS;

END\_VAR

VAR\_OUTPUT

risingEdge : BOOL;

fallingEdge : BOOL;

risingEdgeDouble : BOOL;

END\_VAR

VAR

oldInputValue : BOOL;

risingDetector : R\_Trig;

fallingDetector : F\_TRig;

timer\_Delay : TOF;

END\_VAR

IF (oldInputValue=TRUE)AND(inputValue=FALSE) THEN

fallingEdge := TRUE;

ELSIF (oldInputValue=FALSE)AND(inputValue=TRUE) THEN

risingEdge := TRUE;

ELSE

fallingEdge := FALSE;

risingEdge := FALSE;

END\_IF

timer\_Delay();

IF risingEdge AND (timer\_Delay.Q = TRUE) THEN

risingEdgeDouble := TRUE;

ELSE

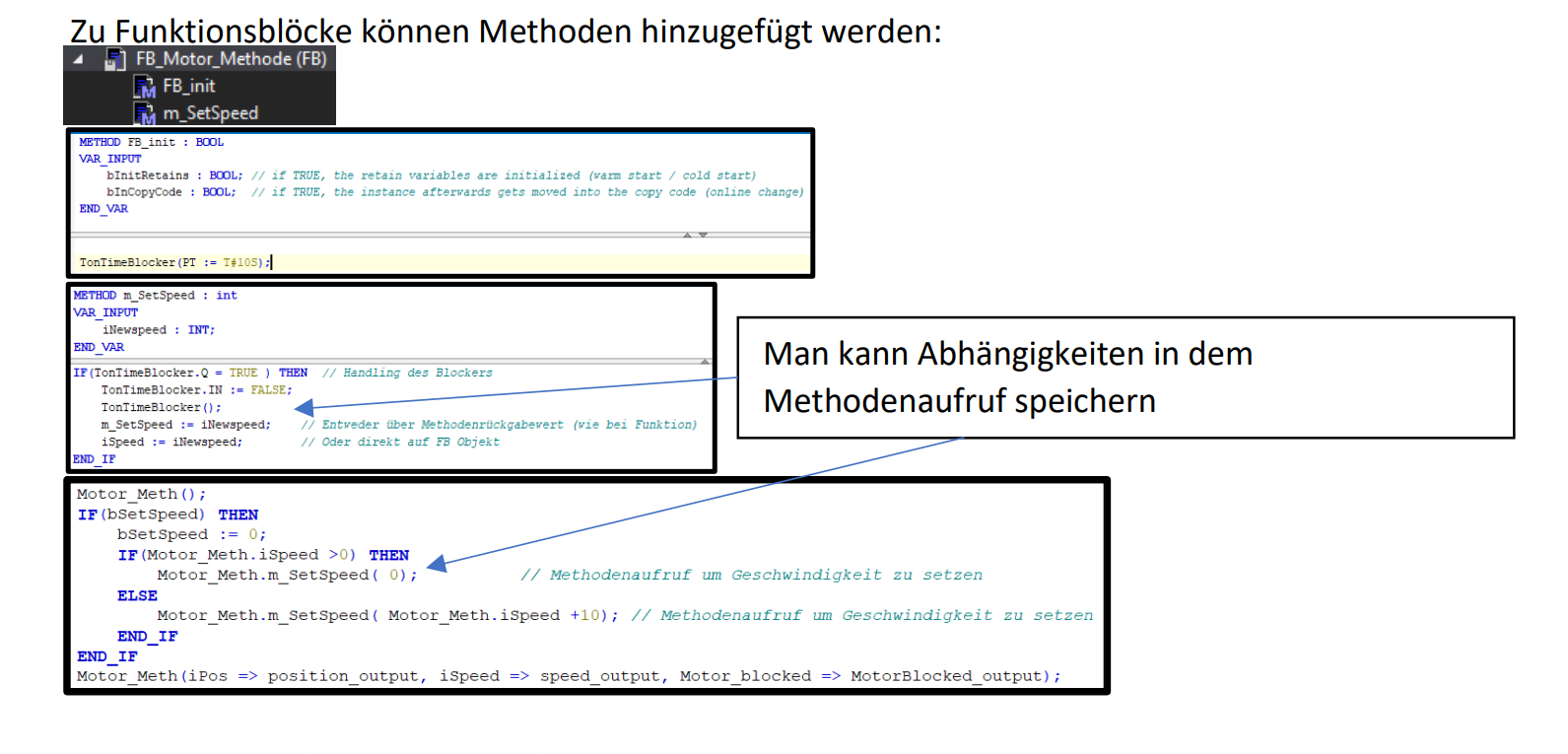
risingEdgeDouble := FALSE;

END\_IF

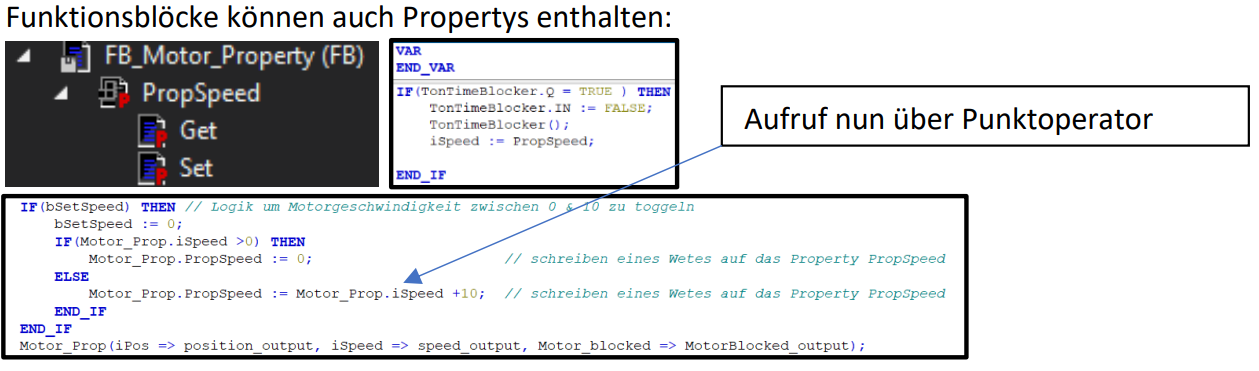
timer\_Delay(IN := inputValue, PT := timerDelayTime);

oldInputValue := inputValue;

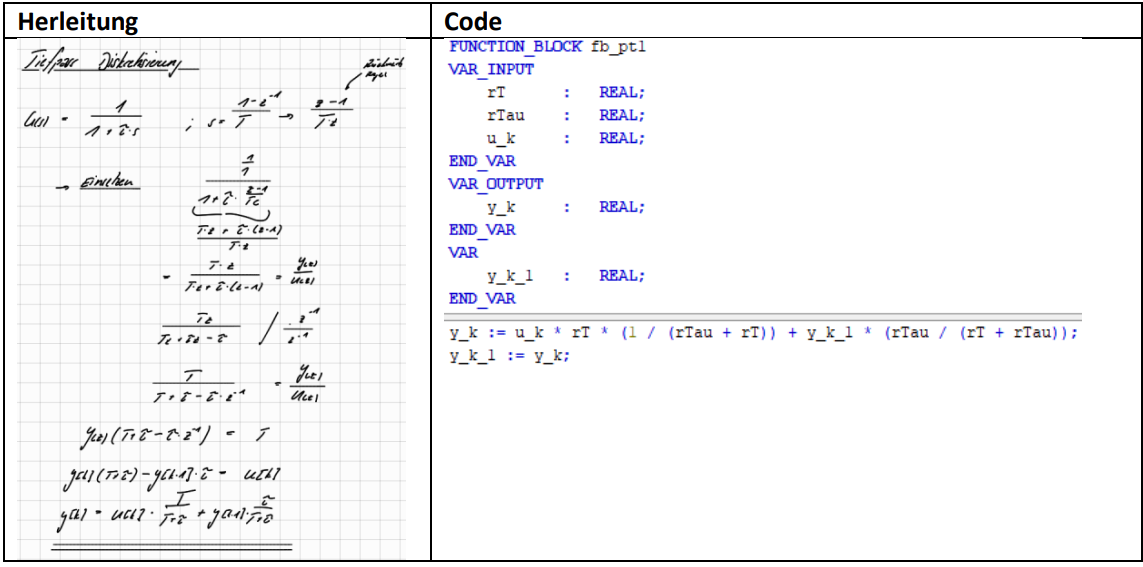
## Methoden



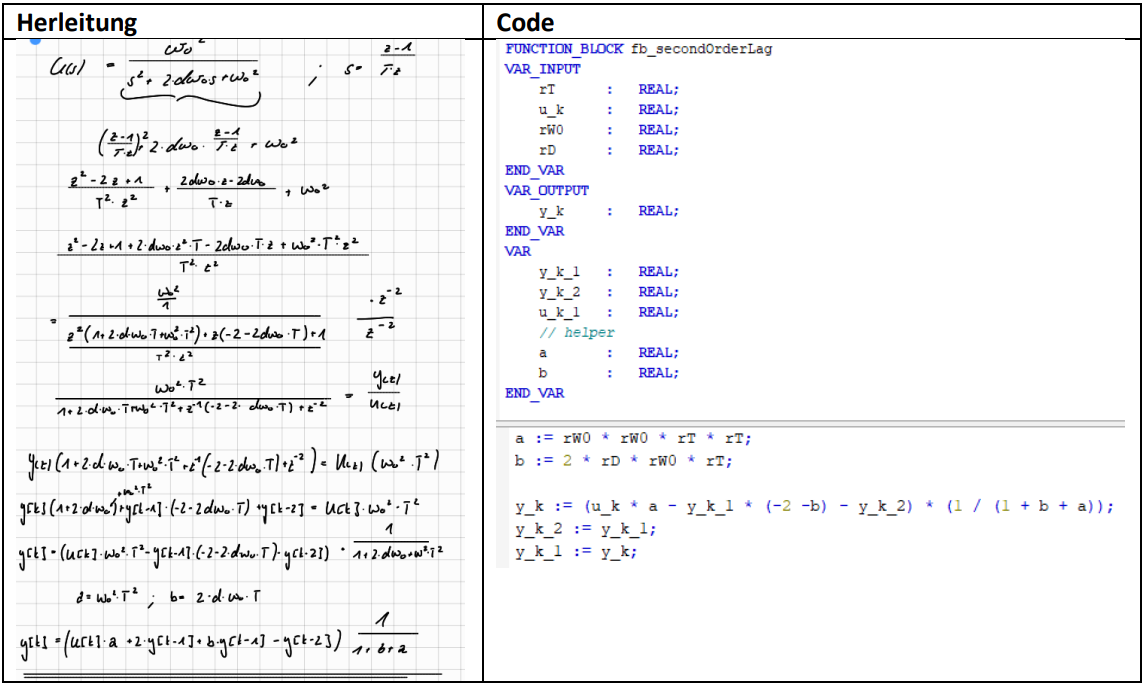
## Property



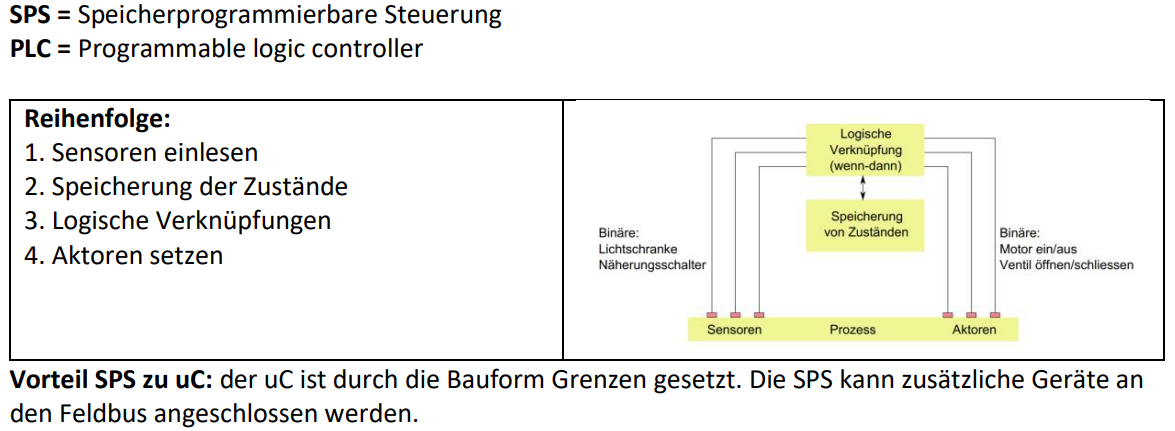
## PT1 (Rückwärtsdifferenz)



## PT2 (Rückwärtsdifferenz)



## Theorie

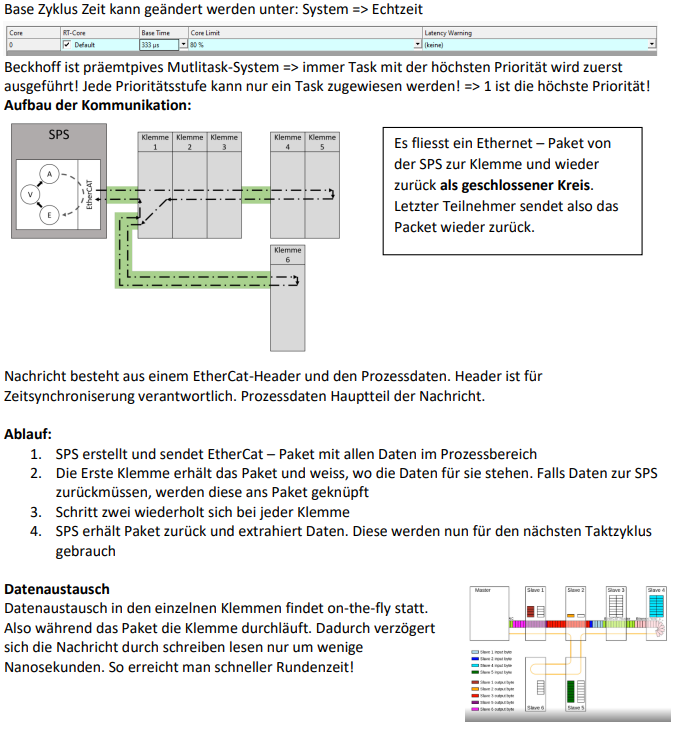


Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Screenshot enthält.

Automatisch generierte Beschreibung

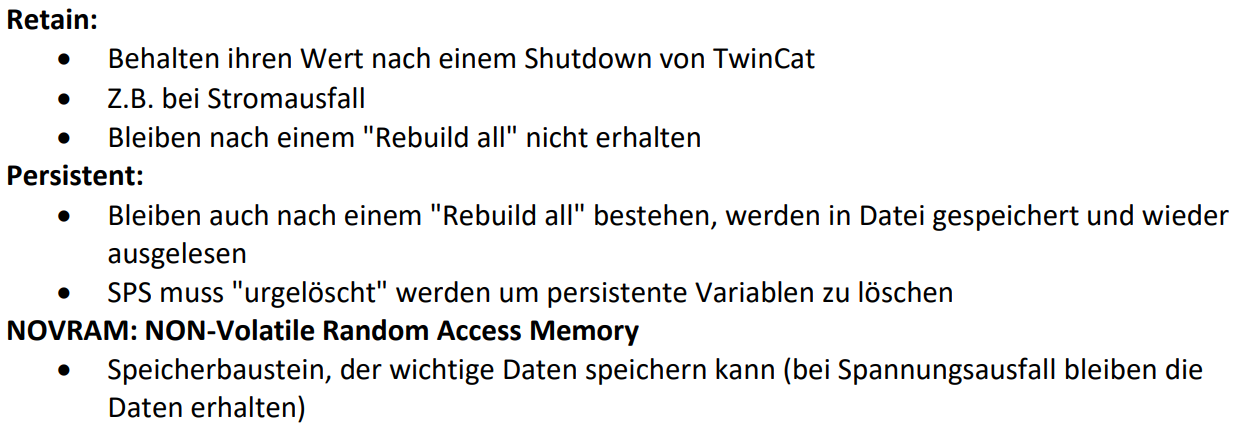
### Task



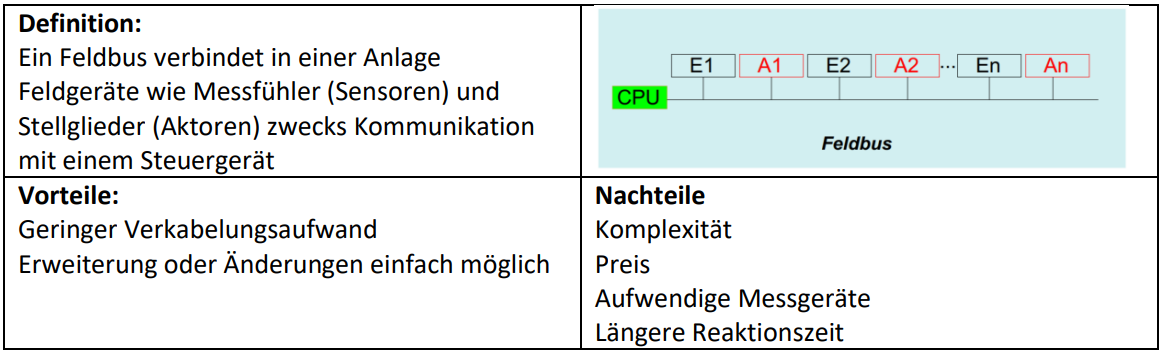
### Export Funktionen



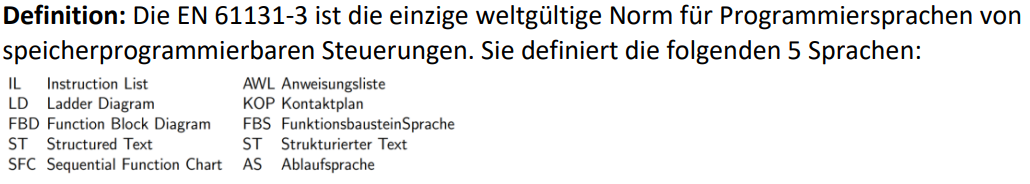
### Remanenz



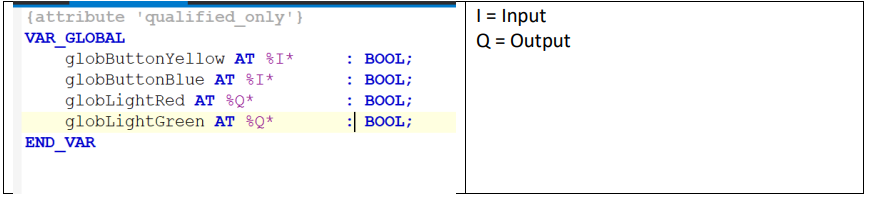
### Feldbusse



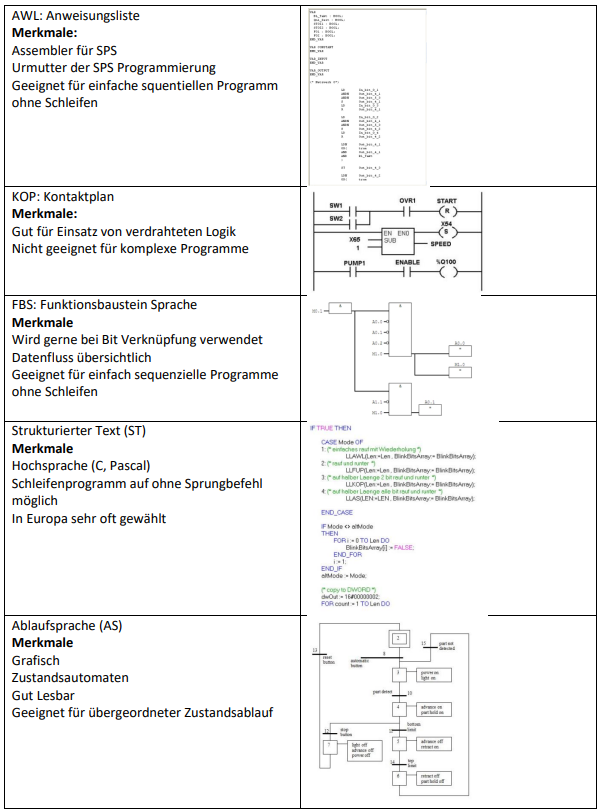
### EN 61131-3



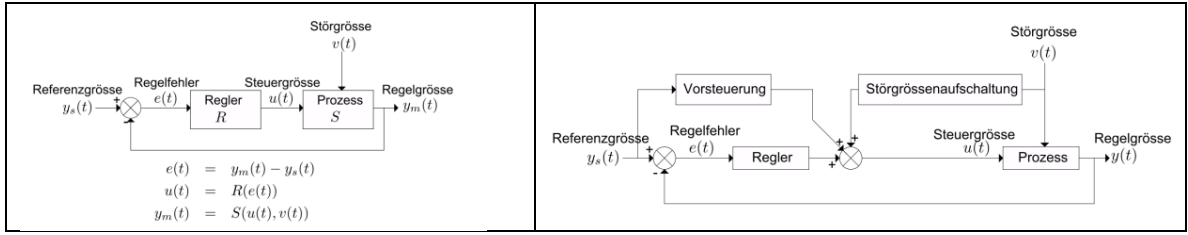
### Verbinden von SPS Klemmen zu Programm



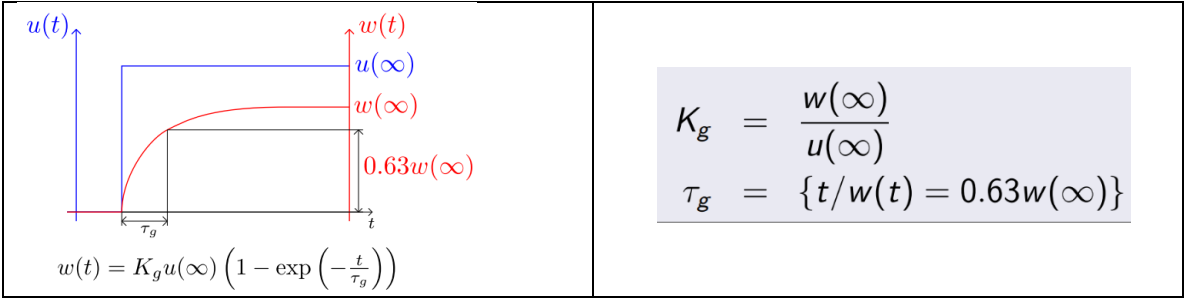
### Programmiersprachen



# Digitale Reglung



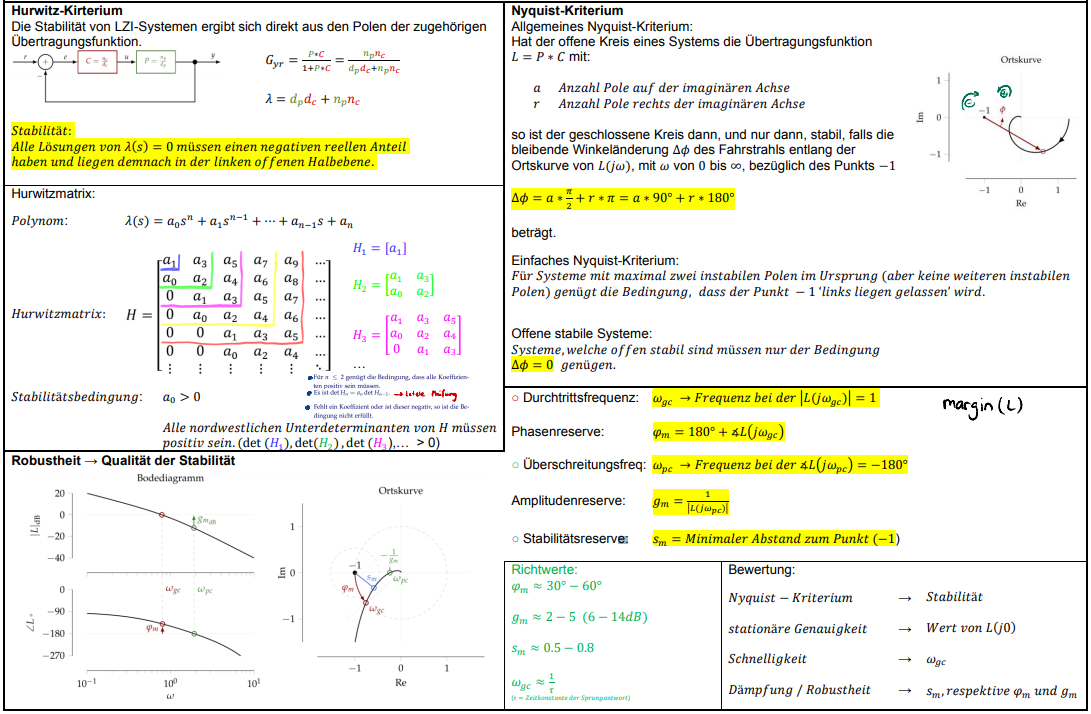
## Bestimmung der Regelparameter: Experimentell PT1 – Glied



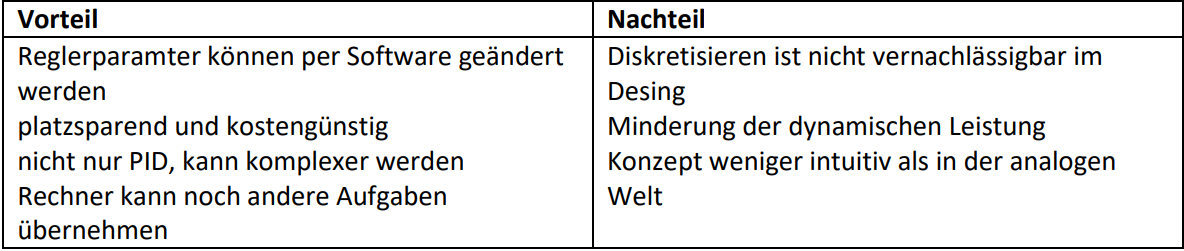
## Einstellregeln

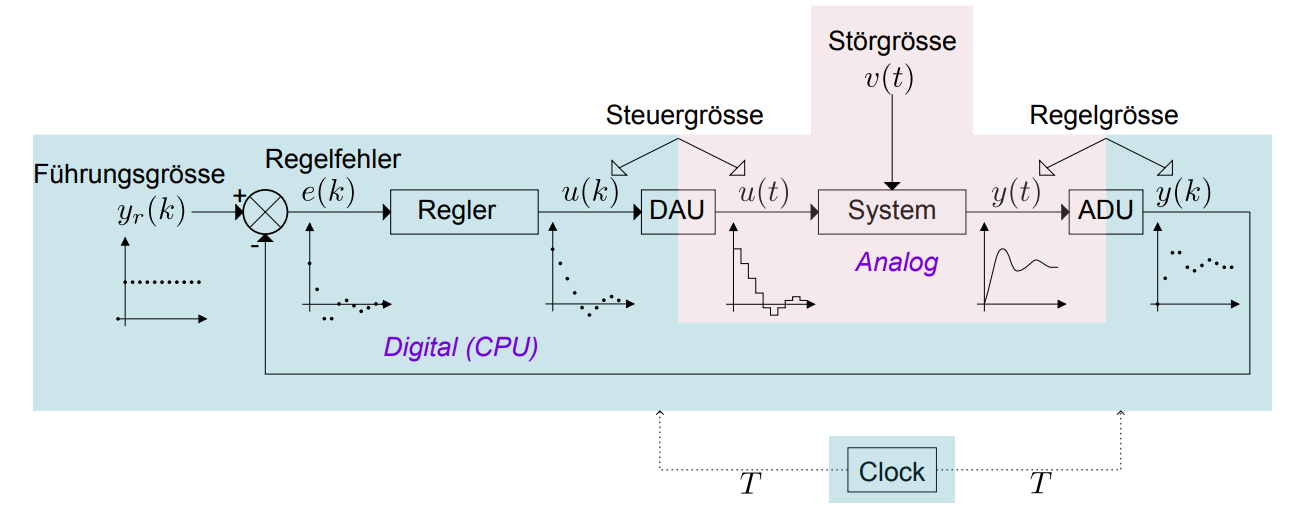
|  |  |
| --- | --- |
|  |  |

## Robustheit



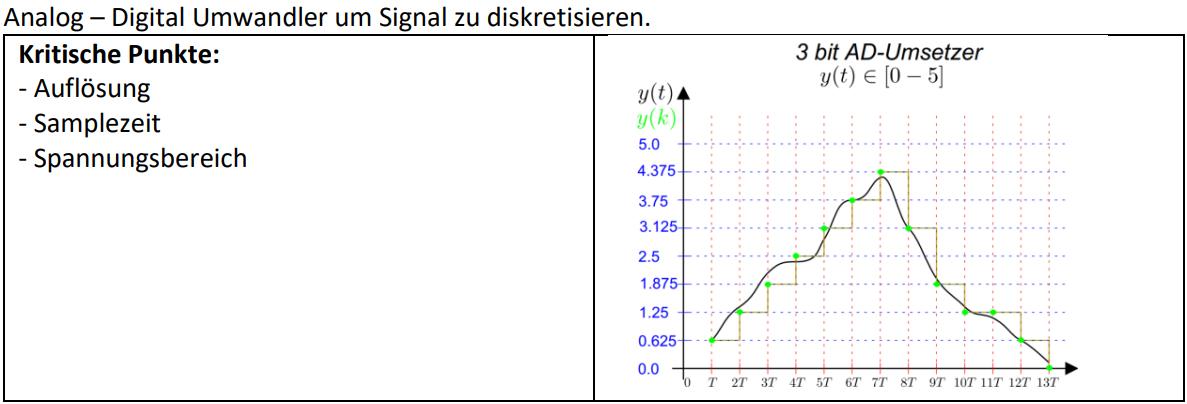
## DIGITALER REGLER





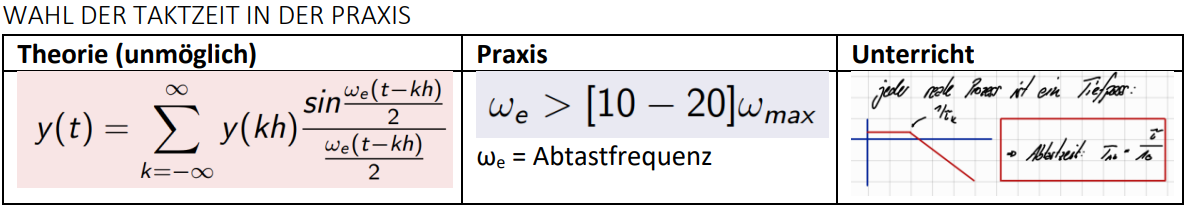
|  |  |
| --- | --- |
| Implementierungsmöglichkeiten | Zentralisiert - Feldbus |
| Industrial PC mit Echtzeitsystem  Mikrocontroller  SPS (Speicherprogrammierbare Steuerung) | Zentralisiert: ADU / DAU im Mikrocontroller, AD/DA für PC, I/O Module für SPS  Feldbus: Profibus, CAN/CANOpen, EtherCat etc |

### AD-Wandler



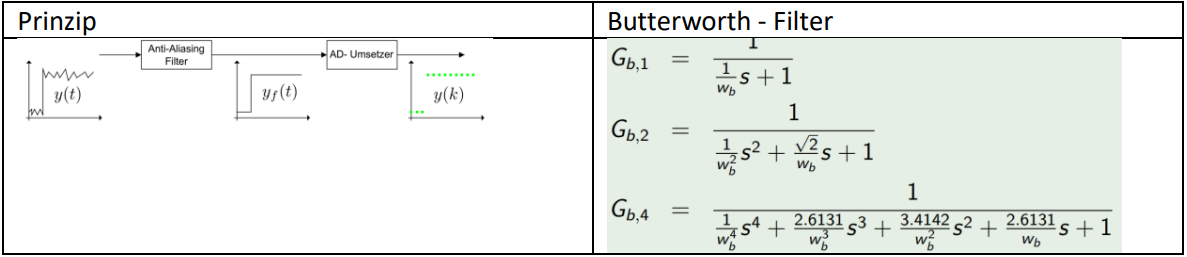
### SHANNON ABTASTTHEOREM

Ein kontinuierliches, bandbegrenztes Signal y(t), mit einer Minimalfrequenz von 0 und einer maximalen Frequenz ωmax muss mit einer Frequenz ωe grösser als 2 \* ωmax abgetastet werden, damit man aus dem so erhaltenen zeitdiskreten Signal y[k] das Ursprungssignal ohne Informationsverlust (aber mit unendlich grossem Aufwand) exakt rekonstruieren kann und (mit endlichem Aufwand) beliebig genau approximieren kann.



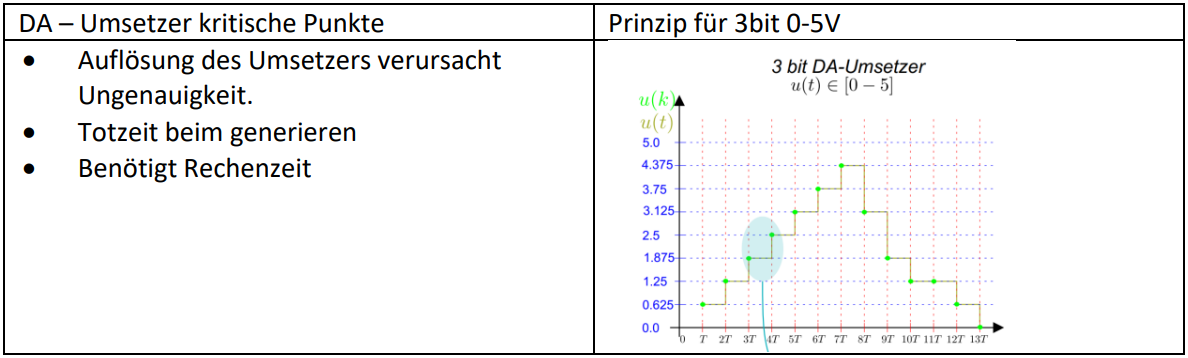
### ANTI ALIASING FILTER

Anforderung ans Shannon Abtasttheorem ist, dass das Signal bandbegrenzt ist. Daher muss ein Tiefpassfilter eingesetzt werden, um dies zu erfüllen.

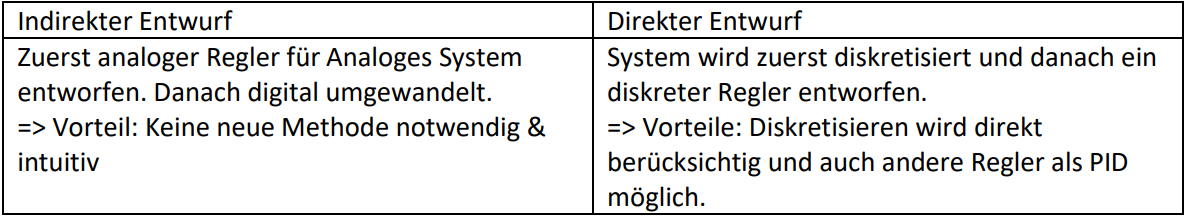


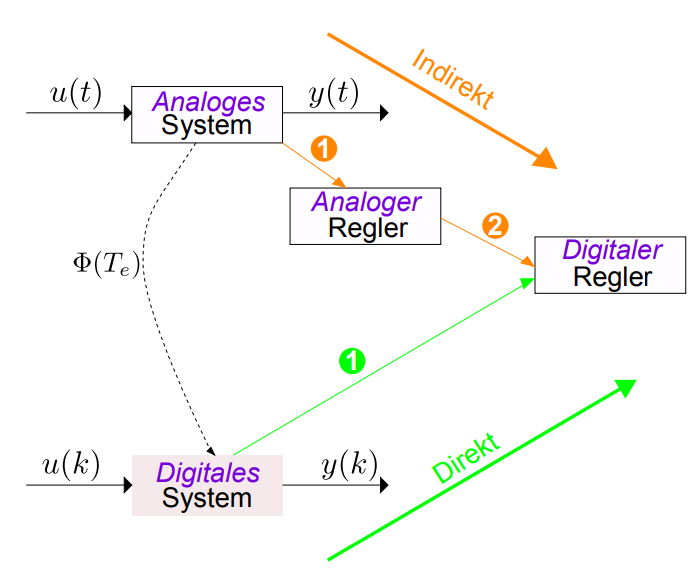
### DA – Wandler

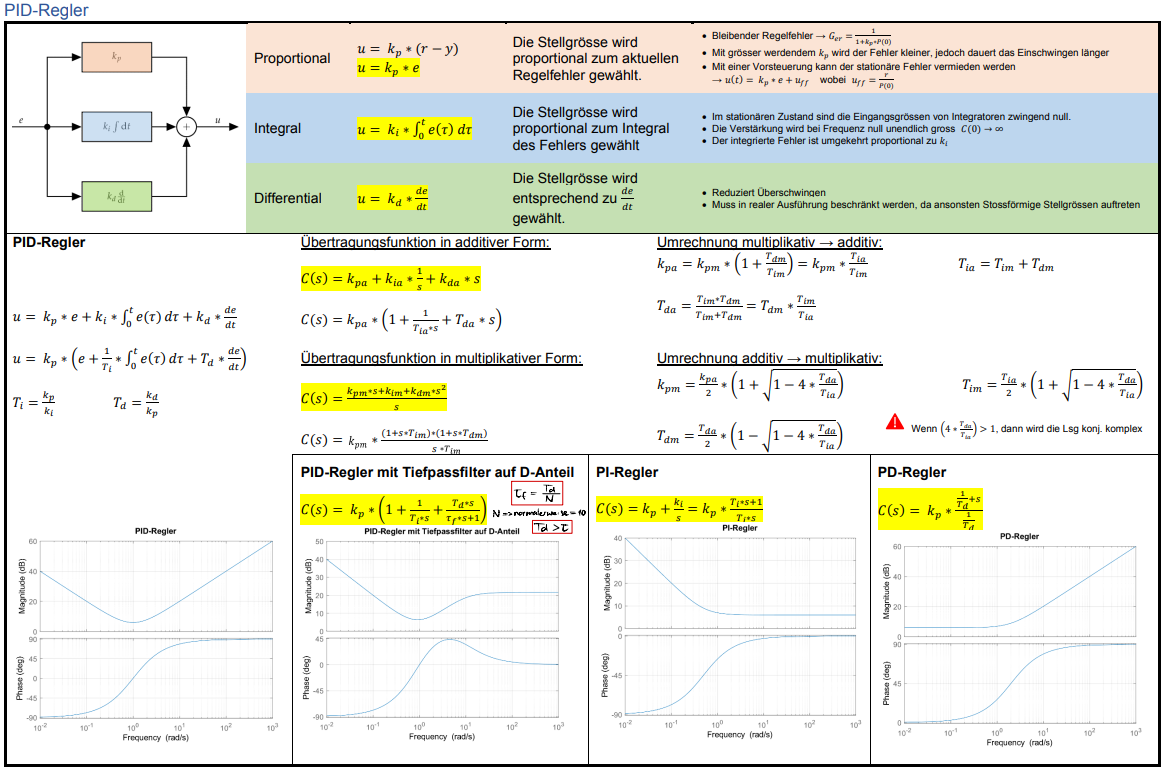
Auch hier ist die perfekte Rekonstruktion des Signals nicht möglich. Die Alternative dazu bietet ein Halte Glied **=> Sample and Hold**



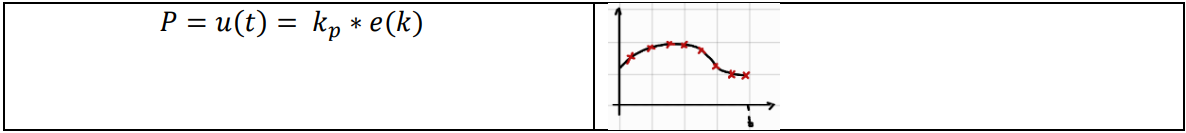
## Regler Diskretisieren







### P-Anteil

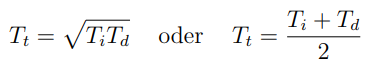


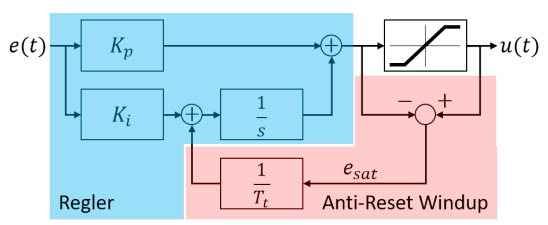
### I-Anteil

Beim I – Anteil gibt es drei Varianten:

|  |  |  |
| --- | --- | --- |
| Rückwärts-Rechteckregel |  |  |
| Vorwärts-Rechteckregel |  |  |
| Trapezregel |  |  |
| Trapezregel mit Anti-Windup (Typ Backcalculation) |  | |



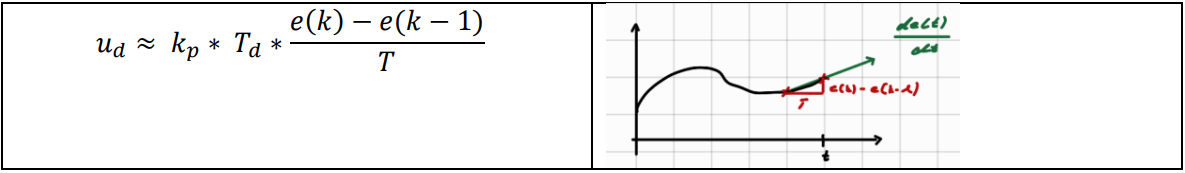


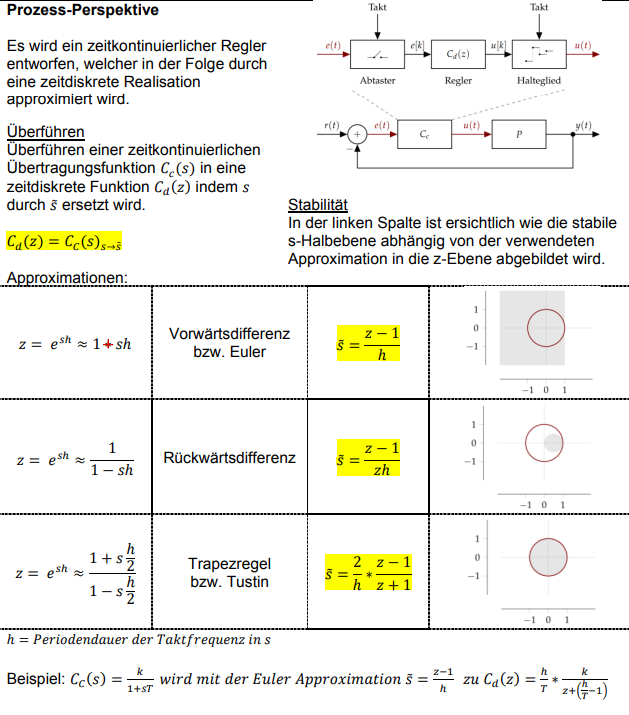


Wird die maximale Stellgrösse erreicht, integriert der I-Anteil weiter. Dies führt zu zwei Problemen: Überlauf und Verzögerung bei Sollwertänderung.

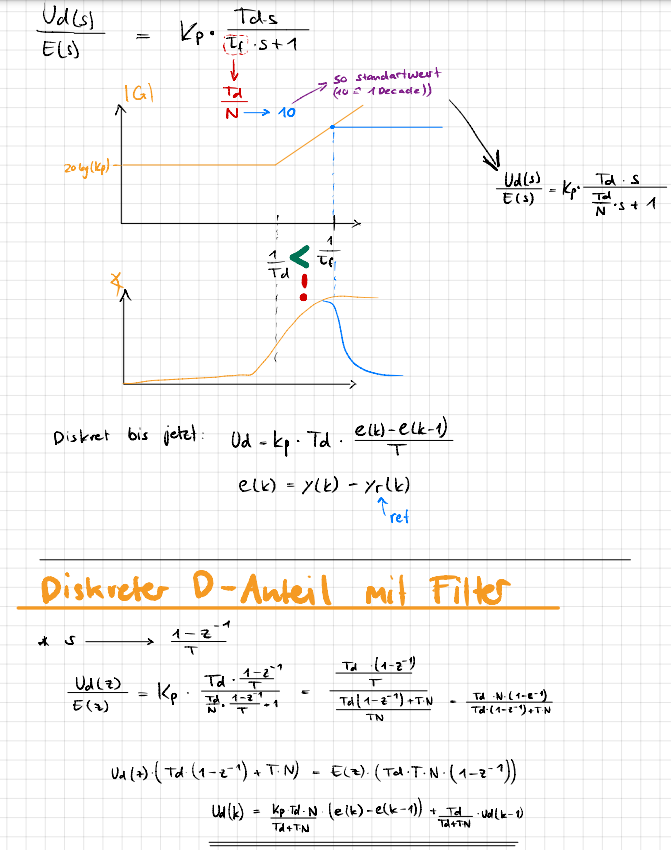
### D-Anteil

|  |  |
| --- | --- |
| Rückwärts-Rechteckregel |  |
| Rückwärts-Rechteckregel mit Filter |  |





### D-Anteil mit Filter



### Saturation

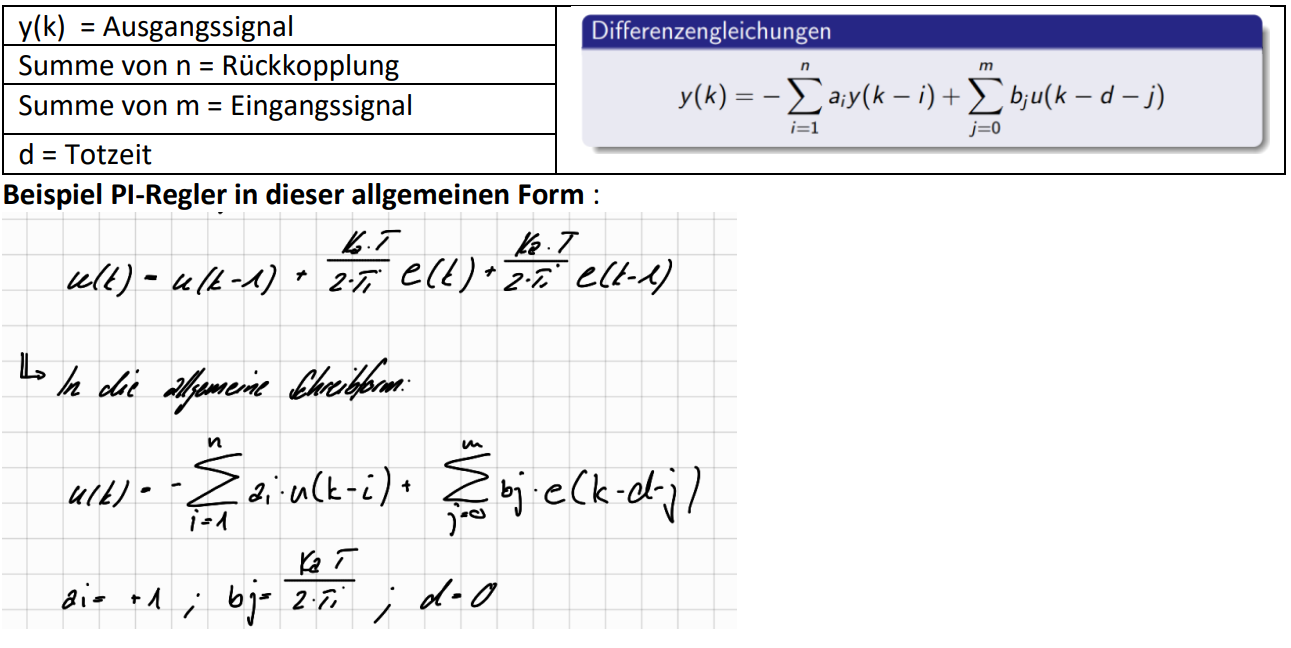
Die Stellgrösse des Reglers muss begrenzt werden, da der Steller auch nur einen begrenzten Wert ausgeben kann

|  |  |
| --- | --- |
|  |  |

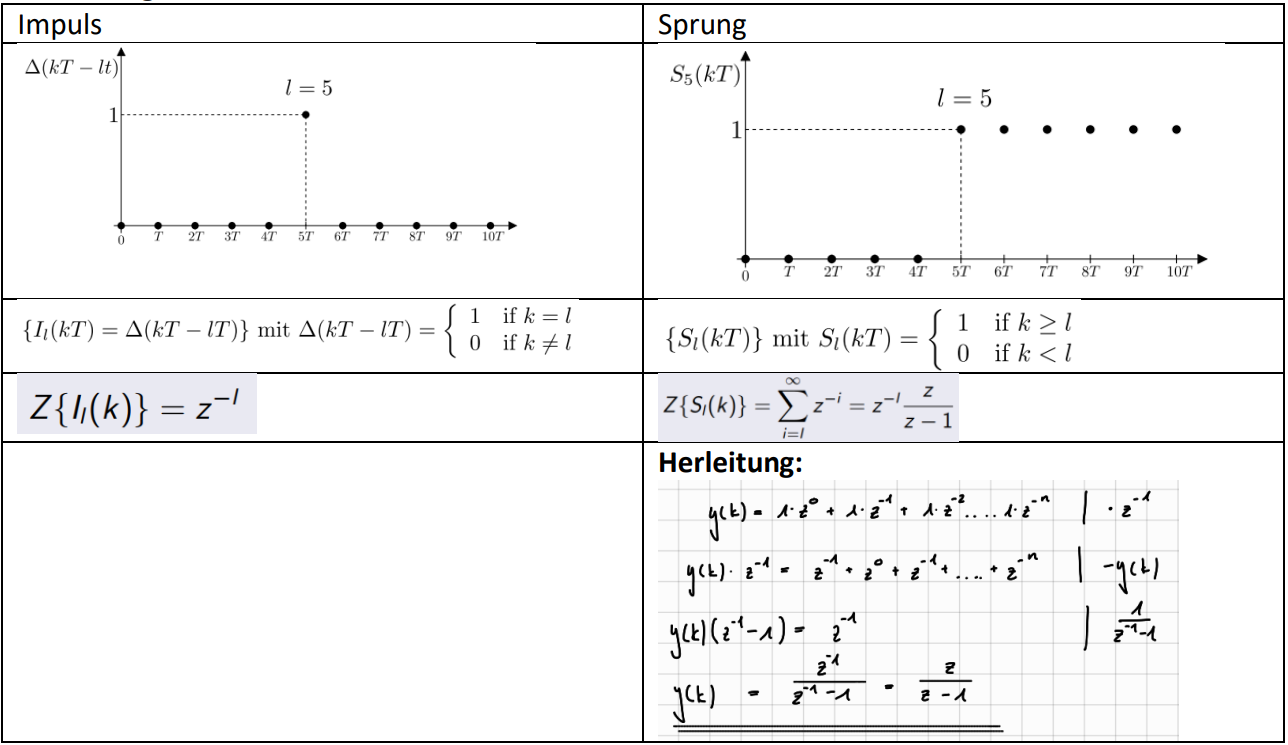
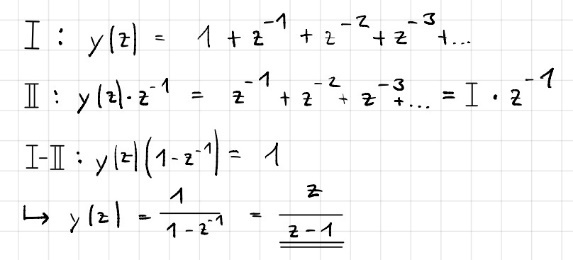
### Pseudocode

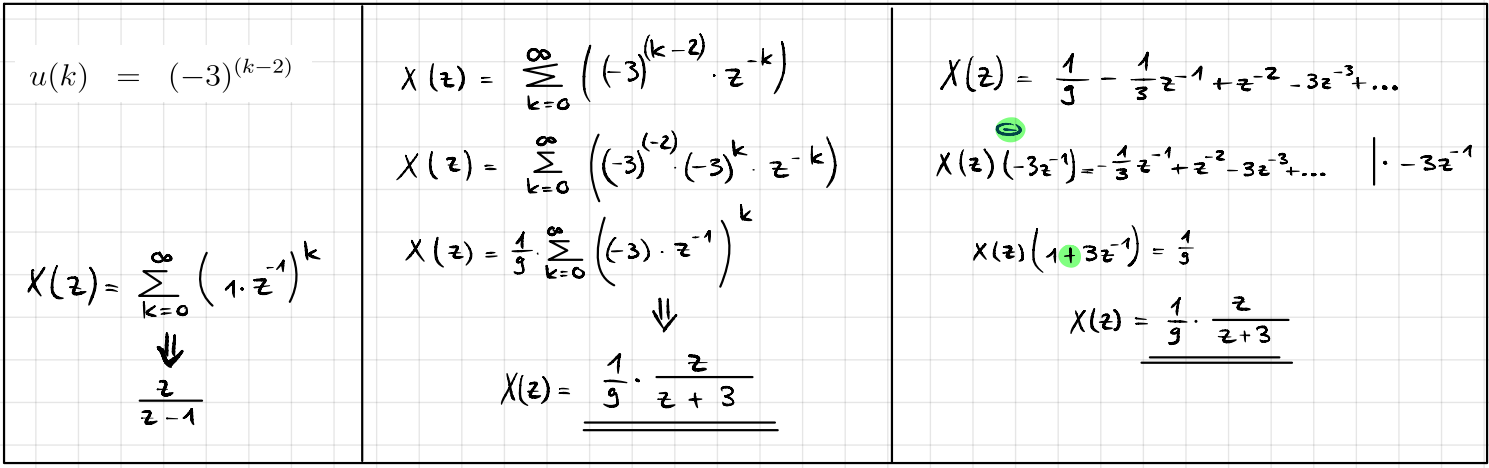
|  |  |
| --- | --- |
|  |  |

## Differenzengleichung

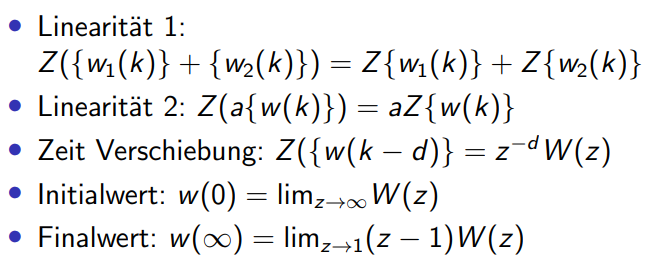


## Z-Transformation

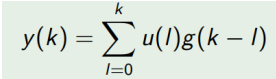




### Eigenschaften der Z-Transformation



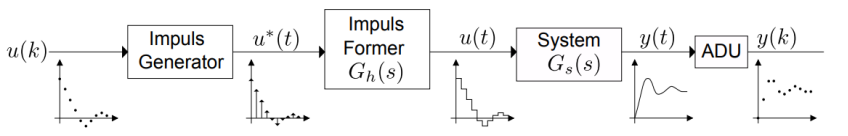
### Impulsantwort beschreibt das System

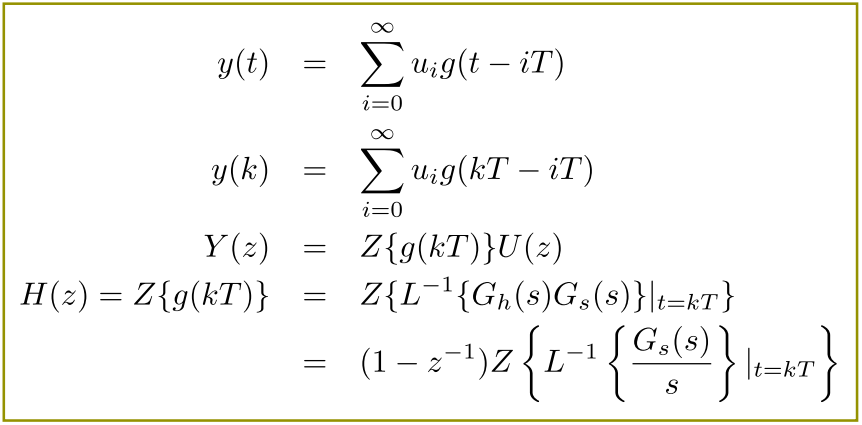


## Die Ordnung eines Systems

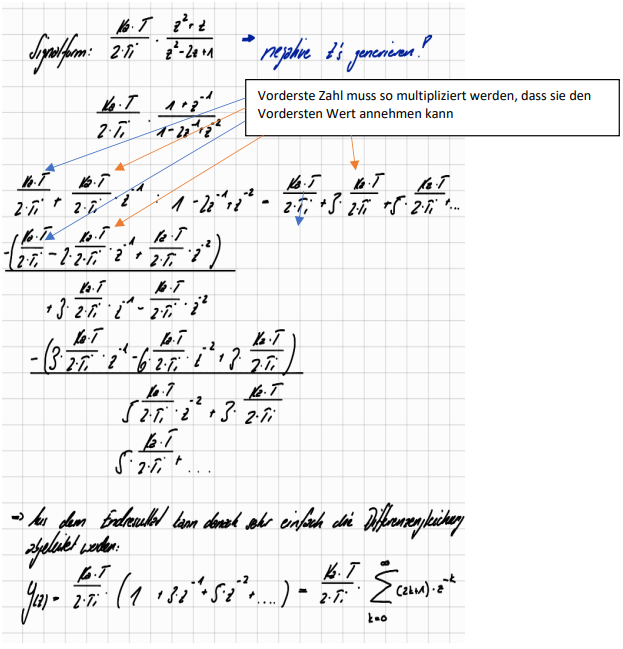
Die Ordnung eines Systems ist die höchste Potenz im Nenner

## Regelstrecke Diskretisieren



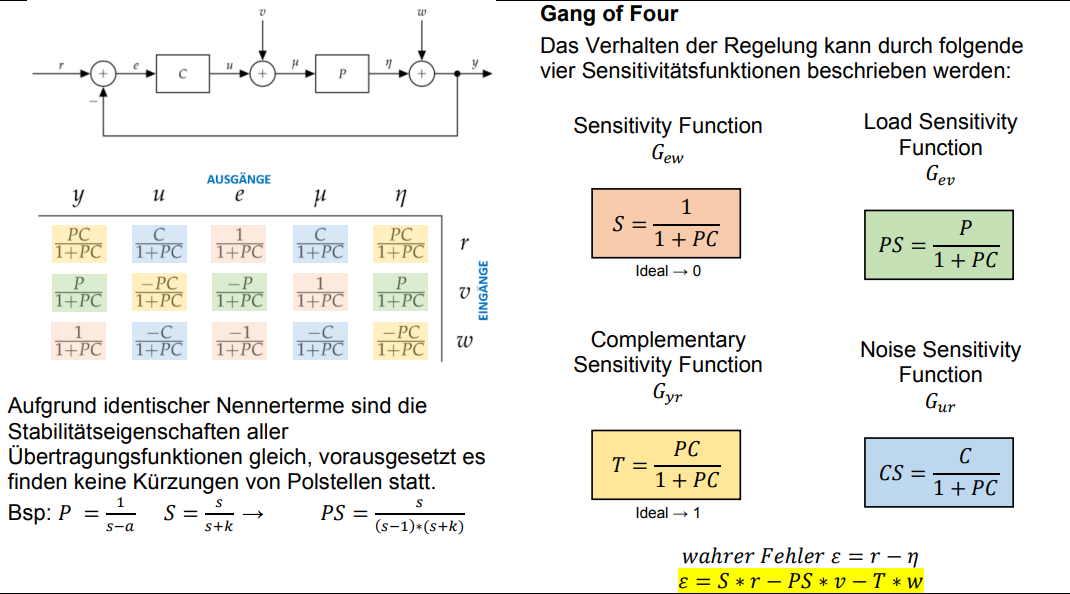


Polynomdivision

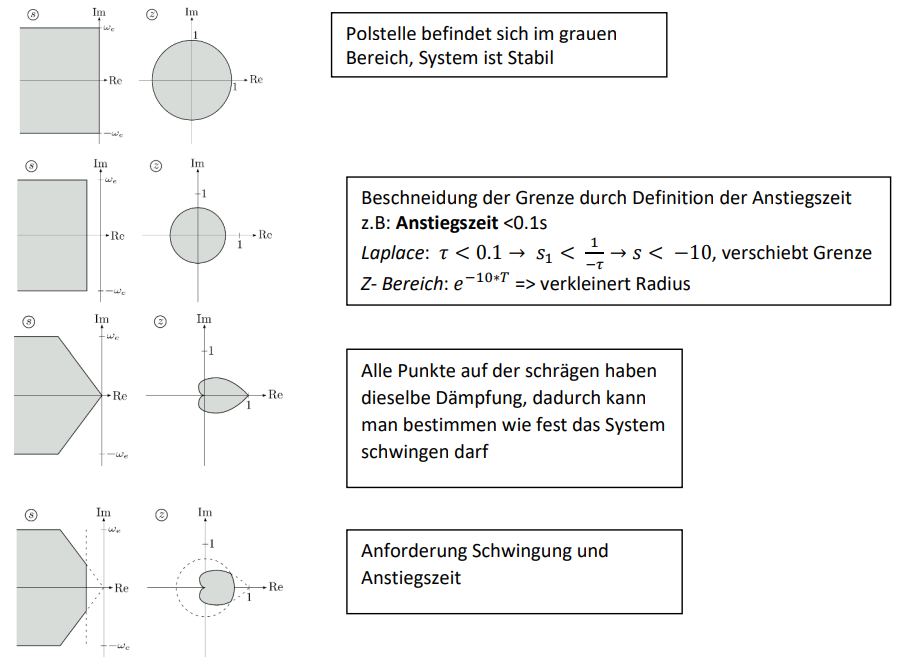


## Störungseinfluss

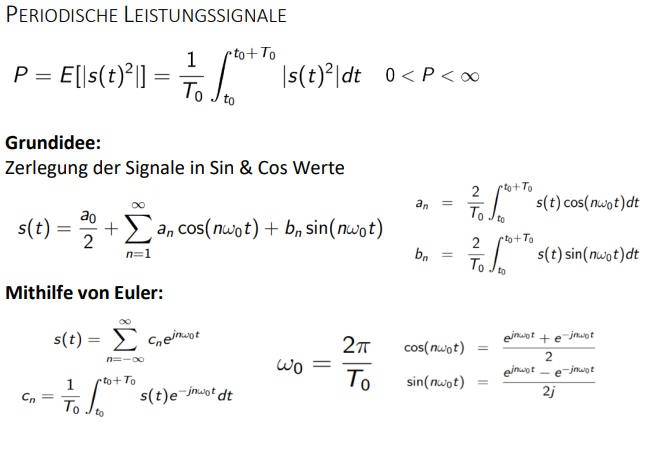
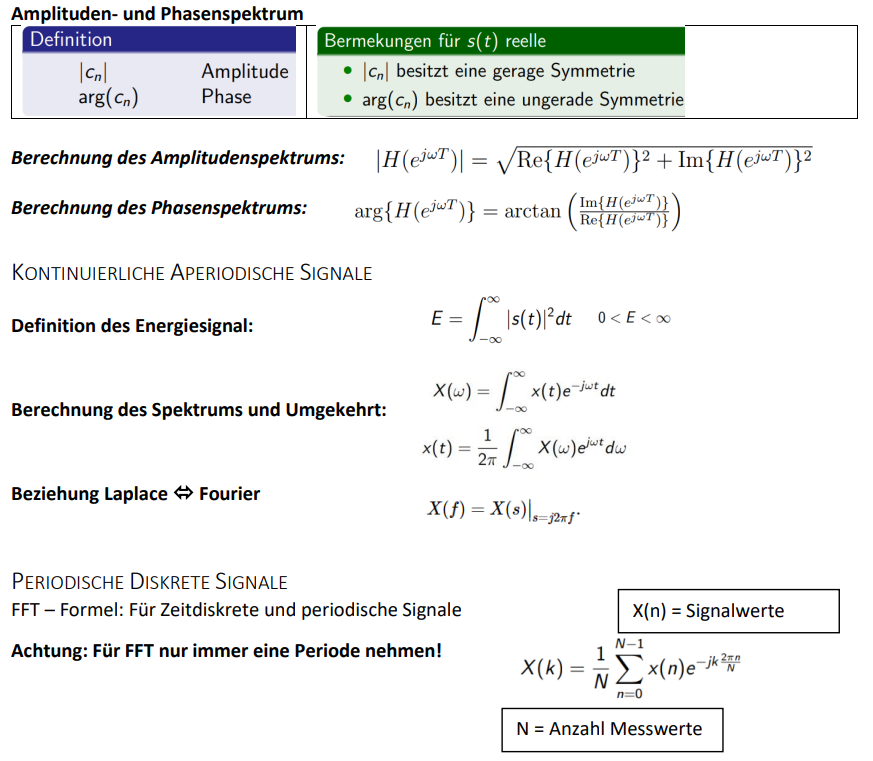
P muss diskretisiert werden und als H(z) angeschaut werden.



## Grafischer vergleich Laplace und Z-Transformation

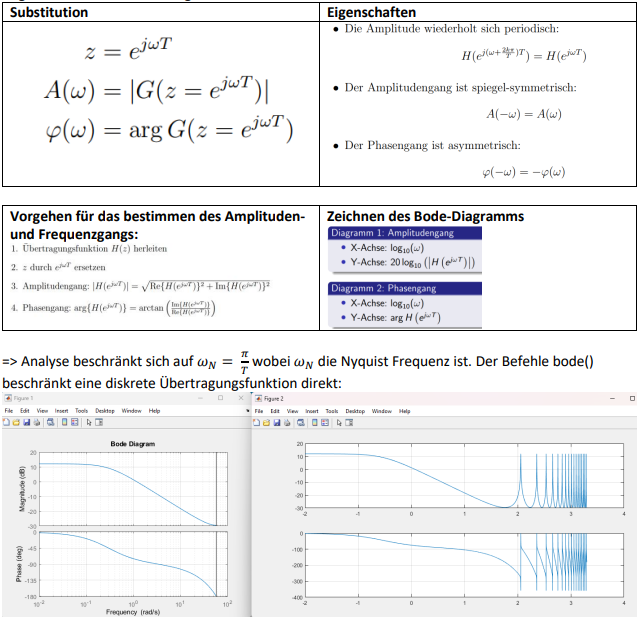
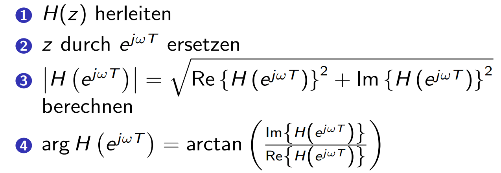
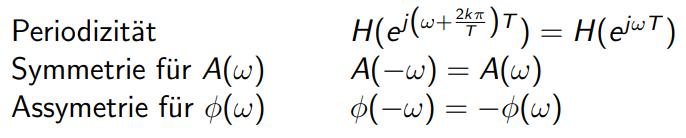


## Fourier Transformation

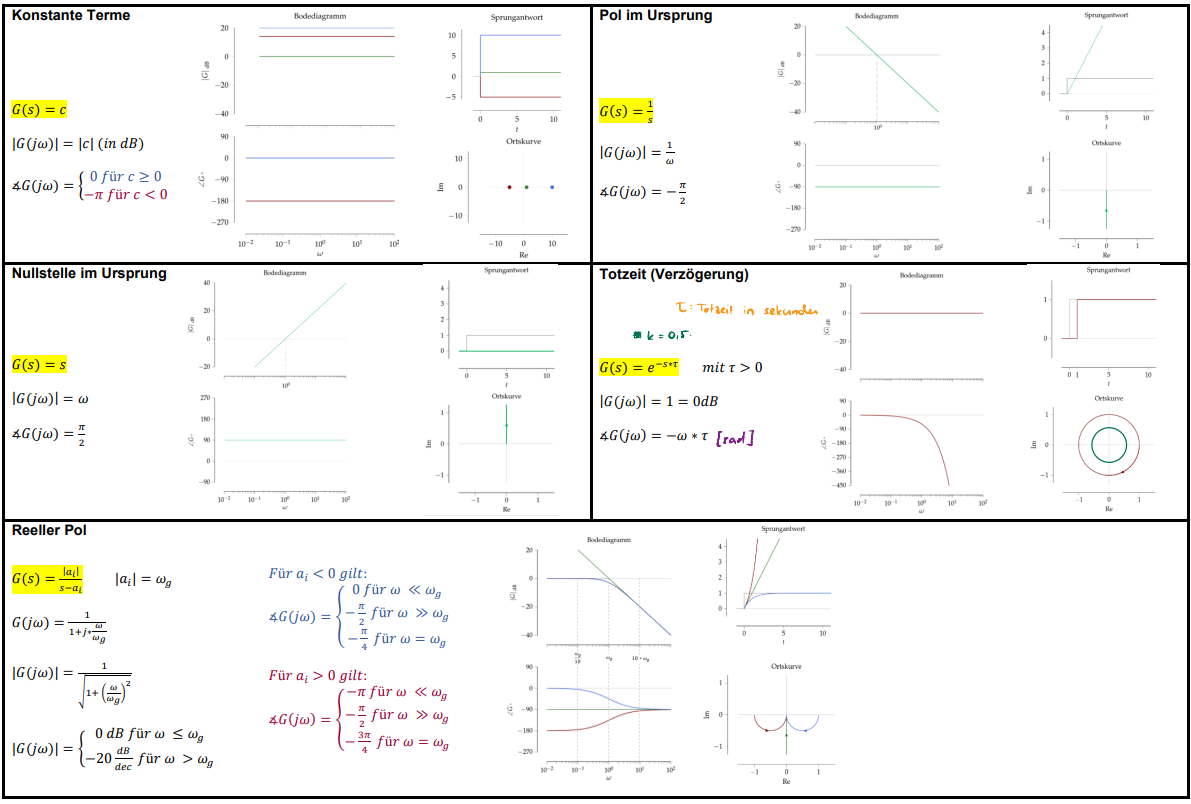
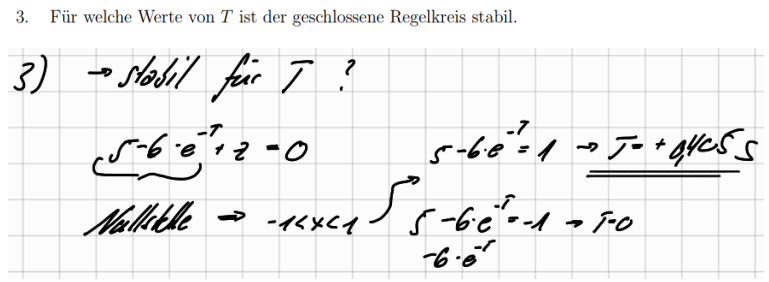
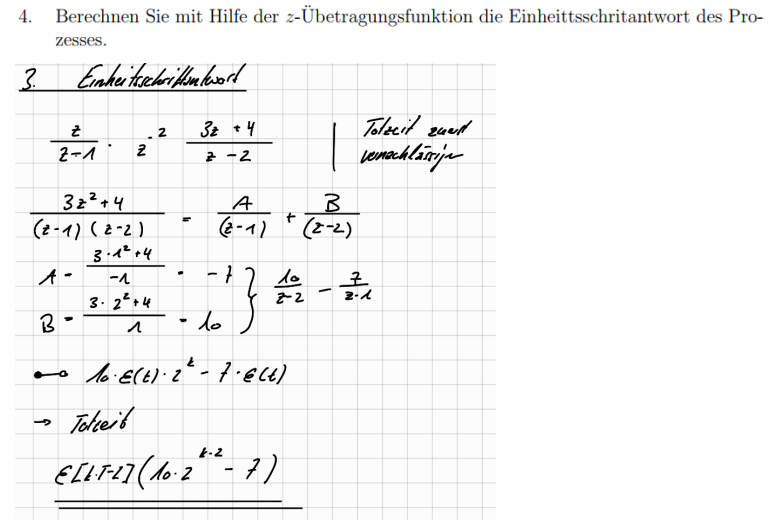
## Diskreter Frequenzgang

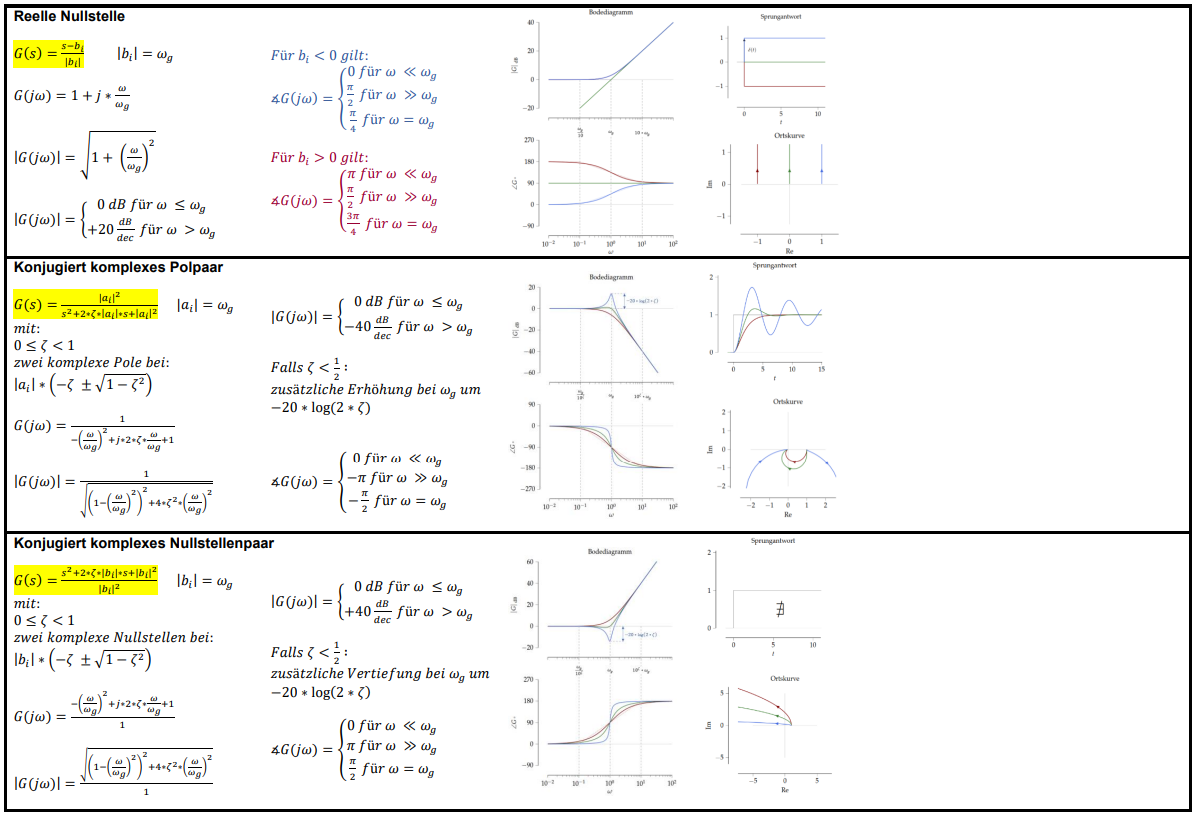
Amplitude- und Phasengang können direkt aus der Z-Übertragungsfunktion abgeleitet werden mit folgender Substitution und Eigenschaften:



### Beispiel







# Nsipre Befehle:

|  |  |
| --- | --- |
| **Beschreibung** | **Nspire Befehl** |
| Partialbruchzerlegung/Ausrechnen | expand() Alternative (propFrac()) |
| Faktorisiere | factor(Ausdr1, Var) bsp. factor(x^2-3,x) |
| Nullstellenberechnen | zeros([FUNKTIONSNAME],[VARIABLENNAME]) |
| Komplexe Nullstellenberechnen | czeros([FUNKTIONSNAME],[VARIABLENNAME]) |