

Praktikum Simulationstechnik





Dieses Dokument wird Studenten, die am Hochschulpraktikum „Simulationstechnik“ des Lehrstuhls für Automatisierungstechnik teilnehmen, als Download zur Verfügung gestellt. Es dient zur Vorbereitung für das Blockpraktikum und ermöglicht einen selbstständigen Einstieg in das Arbeiten mit MATLAB, Simulink und Stateflow.

Dieser Umdruck wurde mit Hilfe von Studenten erstellt und ersetzt nicht den Besuch des Hochschulpraktikums.

Dieser Umdruck stellt keinesfalls einen Ersatz für einschlägige Lehrbücher dar.

Garching bei München, 06.2020



Inhaltsverzeichnis

Versuch 1	5
Versuch 2	59
Versuch 3	83
Versuch 4	103
Versuch 5	111



WICHTIGE INFORMATIONEN

Das Skript ist in fünf Versuche unterteilt, die in weiten Teilen aufeinander aufbauen. Damit vermieden wird, dass sich Fehler aus vorherigen Aufgaben in aufbauenden Aufgaben fortziehen, sind folgende Punkte **ausdrücklich gefordert**.

Software:

- Um die zur Verfügung gestellten Vorlagen öffnen zu können, benötigen Sie die **Matlab** Version **R2021a** oder neuer
- Bitte installieren Sie neben Matlab und Simulink die folgenden Toolboxes vor Antritt des Praktikums:
 - **Aerospace Blockset**
 - **Aerospace Toolbox**
 - **Signal Processing Toolbox**
 - **Stateflow**

Bearbeitung der Aufgabe:

- Speichern Sie Dateien auf dem lokalen Laufwerk und laden Sie dann die **vollständigen Versuche online auf Moodle hoch**. (nur online eingereichten Modelle werden bewertet)
- Arbeiten Sie ordentlich und strukturiert, d.h. nutzen Sie in Matlab die Möglichkeit Ihren Code zu kommentieren. In Simulink sollten Sie Signale, Ein- und Ausgänge, sowie Systeme eindeutig bezeichnen. Wenn Bezeichnungen in der Aufgabenstellung gegeben sind, nutzen Sie diese.
- Testen Sie die Modelle. Überlegen Sie sich wie sich das Modell bei bestimmten Eingangssignalen verhalten sollte und überprüfen Sie dies. Versuchen Sie auch Eingänge zu wählen, die nicht trivial sind, in anderen Worten, versuchen Sie Ihr Modell zu „zerstören“.
- Versichern Sie sich mehrmals, dass ihre Modelle korrekt implementiert sind. Bei Unklarheiten wird die Sprechstunde mit den Tuto ren empfohlen. Dadurch ist sichergestellt, dass Sie keine Folgefehler mitziehen. Nichts ist demotivierender, als wenn Sie durch alte Fehler nicht zum gewünschten Ziel kommen.
- Zeigen Sie, dass Ihr Modell voll funktionstüchtig ist. Bereiten Sie verschiedene Eingangssignale vor mit denen die Funktionalität des Systems überprüft werden kann.
- Eine Abnahme erfolgt ausschließlich wenn,
 - alle Signale und Ports eindeutig benannt sind,
 - die Ergebnisse übersichtlich dargestellt sind und auf einem Blick nachvollzogen werden können,
 - Graphen eine eindeutige Benennung und Legende aufweisen,
 - und erklärt werden kann, warum das Modell funktioniert.

Die Tutoren sind bei Fragen gerne bereit Ihnen zu helfen. **Sie werden jedoch ausschließlich Hilfestellungen zur Lösungsfindung geben und Ihnen keine fertige Lösung präsentieren.**

Viel Erfolg!



Praktikum Simulationstechnik



Versuch 1



Inhaltsverzeichnis

1 PRAKTIKUM SIMULATIONSTECHNIK – ORGANISATORISCHES	8
1.1 ZIEL DES PRAKTIKUMS.....	8
1.2 AUFBAU DES PRAKTIKUMS	8
1.3 MATLAB ALS DOWNLOAD	8
1.4 WEITERFÜHRENDE LITERATUR UND ÜBUNGEN	8
2 EINFÜHRUNG	9
2.1 WAS IST MODELLBILDUNG UND SIMULATION	9
2.2 VORGEHENSWEISE DER MODELLBILDUNG UND SIMULATION.....	9
2.3 SIMULATIONSTECHNOLOGIEN.....	11
2.3.1 ABLAUFsimulation.....	11
2.3.2 3D-KINEMATIKSIMULATION.....	11
2.3.3 FEM	12
2.3.4 MEHRKÖRPERSIMULATION	12
2.4 KLASIFIZIERUNG VERSCHIEDENER SIMULATIONSARTEN.....	13
3 MATLAB	14
3.1 GRUNDLAGEN ZU MATLAB	14
3.1.1 BASIS-FUNKTIONALITÄT.....	15
3.1.2 HILFEFUNKTION	18
3.1.3 M-FILES: SKRIpte UND FUNKTIONEN	19
3.2 AUFGABEN.....	21
3.2.1 EINGABE VON VARIABLEN	21
3.2.2 ERZEUGEN EINES SKRIPTS	21
3.2.3 ERZEUGEN EINER FUNKTION.....	21
4 SIMULINK.....	22
4.1 BESCHREIBUNG VON SIMULINK	23
4.1.1 STARTEN VON SIMULINK	23
4.1.2 WICHTIGE SHORTCUTS.....	23
4.1.3 SIMULINK BIBLIOTHEKEN.....	23
4.1.4 SUBSYSTEME.....	27
4.1.5 DAS MODELLFENSTER	29
4.1.6 SIMULATIONSPARAMETER	30
4.1.7 DIAGNOSEFENSTER.....	31
4.2 VORGEHEN ZUR ERSTELLUNG EINES KONTINUIERLICHEN SYSTEMS	32
4.2.1 FEDERPENDEL	32
4.2.2 DREHTELLERBEWEGUNG	33
4.3 AUFGABEN.....	35
4.3.1 EINFACHE SYSTEME	35
4.3.2 SUBSYSTEME.....	35
4.3.3 MODELLIERUNG DES DREHTELLERS	36
5 STATEFLOW.....	37



5.1	GRUNDLAGEN	37
5.1.1	ELEMENTE EINES ZUSTANDSAUTOMATEN.....	37
5.1.2	HIERARCHISIERUNG UND BEHERRSCHUNG VON KOMPLEXITÄT.....	43
5.1.3	EIGENSCHAFTEN VON CHARTS.....	45
5.1.4	DATA DICTIONARY UND MODEL EXPLORER	46
5.1.5	ACTION LANGUAGE	47
5.1.6	VARIABLEN UND EVENTS	49
5.1.7	BEISPIEL ZUR ALLGEMEINEN VORGEHENSWEISE.....	52
5.1.8	PROGRAMMBEISPIEL	53
5.1.9	CHECKLISTE: HÄUFIGE FEHLER.....	54
5.2	AUFGABEN.....	55
5.2.1	WICHTIGE HINWEISE	55
5.2.2	MODELLIERUNG EINER EINFACHEN TEMPERATURREGELUNG.....	55
5.2.3	TIMER	56
5.2.4	WINKELSENSOR	56
5.2.5	VEREINFACHTE DREHTELLERSTEUERUNG.....	57



1 Praktikum Simulationstechnik – Organisatorisches

1.1 Ziel des Praktikums

Ziel des Praktikums Simulationstechnik ist die Vermittlung von praktischen Erfahrungen bei der Modellierung und Simulation technischer Produkte und Prozesse. Dazu wird mit Hilfe des Simulationswerkzeugs MATLAB/Simulink und der auf Zustandsautomaten basierenden Toolbox „Stateflow“ zunächst eine kleine automatisierungstechnische Anlage simuliert, bei der auch Optimierungsansätze vermittelt werden. Die daraus gewonnenen Erkenntnisse werden für die Simulation einer größeren Anlage genutzt, die sowohl aus Steuerungssoftware als auch aus Hardware (Aktorik und Sensorik) und Mechanik (Fließband, Greifer, etc.) besteht.

Es werden schrittweise die kontinuierlichen Simulationsanteile Hardware und Mechanik in Simulink modelliert, sowie die ereignisorientierte Steuerung mit der Toolbox „Stateflow“ nachgebildet. Die erstellten Einzelmodelle werden anschließend zu einem hybriden System verknüpft.

1.2 Aufbau des Praktikums

- | | |
|-------|--|
| Tag 1 | Grundlagen zur Simulation mit MATLAB und Simulink und zur ereignisorientierten Simulation mit Stateflow |
| Tag 2 | Virtuelle Inbetriebnahme von mechatronischen Systemen |
| Tag 3 | Modellieren der HW- und Mechanikkomponenten der Abfüllanlage (Simulink), Modellierung des Prozessguts (kontinuierliche Simulation) |
| Tag 4 | Modellieren der Anlagensteuerung (Stateflow) |
| Tag 5 | MATLAB, Modellbildung, Optimierung und Abschlusstest |

1.3 MATLAB als Download

Die TUM hat mit MathWorks einen Rahmenvertrag für den Bezug von MATLAB abgeschlossen. Dies ermöglicht MATLAB, Simulink und alle Toolboxen mit einer Studentenlizenz zu benutzen.

Download und Lizenz

- Der Einstieg zum Bezug dieser Produkte erfolgt über folgendes Portal: <http://matlab.rbg.tum.de>
- Weitere Informationen zu Bezug und Aktivierung der Produkte erhalten Sie nach dem Login mit Ihrer TUM-Kennung auf dem genannten Portal.
- Die aktuellen Versionen von MATLAB, Simulink und den Toolboxen sind über das MathWorks License Center <http://www.mathworks.com> erhältlich. Eine Download-Anleitung gibt Hilfestellung für das Herunterladen der Software.

1.4 Weiterführende Literatur und Übungen

Die folgenden Grundlagen zu MATLAB, Simulink und Stateflow sind speziell auf die Fragestellungen und Aufgaben, die während des Praktikums bearbeitet werden sollen, ausgerichtet. Weiterführende Fachliteratur und Onlinehilfe sind unter folgendem Link zu finden: <http://www.mathworks.de/support/books/>

Im **Lehrbuchbestand der TUM-Bibliothek** finden Sie z. B.: A. Angermann: *MATLAB, Simulink, Stateflow – Grundlagen, Toolboxen, Beispiele*. München, Oldenbourg, 2014.

Die zugehörigen **Übungsbeispiele zum Download** gibt es unter: <http://www.matlabbuch.de>

2 Einführung

2.1 Was ist Modellbildung und Simulation

Mit Modellbildung und Simulation wird der Problemlösungsprozess von der Realität auf ein abstrahiertes Abbild der Wirklichkeit verlagert und somit unterstützt. Es wird ein existierendes oder gedachtes System virtuell nachgebildet, um damit experimentieren und schließlich Aussagen über die Wirklichkeit zu bekommen. Simulationen erlauben es, Erkenntnisse über Systeme zu gewinnen, die in der Realität nicht oder nur mit wesentlich höherem Aufwand experimentierbar sind (zu groß: z.B. Galaxien, zu teuer: z.B. Raumfahrttechnik, reales System nicht vorhanden: z.B. zu entwickelndes Produkt) bzw. deren zu untersuchendes Verhalten in der Zukunft liegt (z.B. Wettervorhersage).

Ein Beispiel für eine Simulation ist in Abbildung 1 zu sehen. Der zu entwickelnde Roboter wurde simuliert, um beispielsweise Erreichbarkeits- und Kollisionsvermeidungskontrollen durchzuführen.

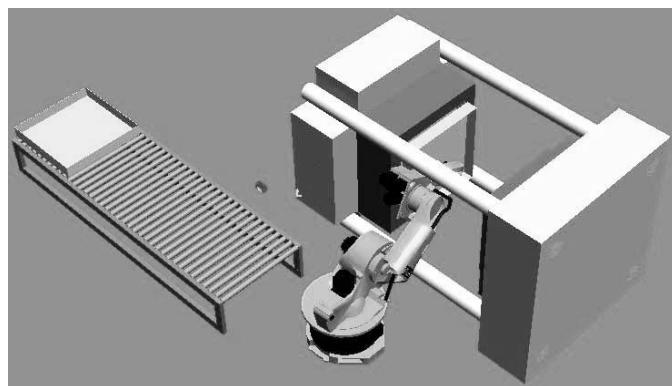


Abbildung 1: 3D-Simulation eines Industrieroboters

2.2 Vorgehensweise der Modellbildung und Simulation

Vor der eigentlichen Durchführung von Simulationsexperimenten muss das zu untersuchende System in einem Simulationssystem nachgebildet werden. Im Folgenden sind die Hauptschritte der Modellbildung und Simulation kurz erläutert:

Problemspezifikation: Wie bei der Entwicklung eines Produkts muss zu Beginn genau spezifiziert werden, was das eigentliche Ziel (Modellierungszweck) der Simulation ist. In der zu erstellenden Anforderungsspezifikation ist eindeutig zu beschreiben, welche Probleme mit Hilfe der Simulation gelöst bzw. welche Aussagen gewonnen werden sollen. Dabei sind neben dem Modellierungszweck Randbedingungen, wie z.B. die notwendige Genauigkeit der Simulation und Zeitbedingungen, die durch das reale System vorgegeben sind, anzugeben.

Modellbildung (Modellieren): In der Phase der Modellbildung wird das zu untersuchende **System analysiert** und ein **konzeptionelles Modell** erstellt. Dabei ist unter Modell allgemein eine vereinfachte Nachbildung eines existierenden oder gedachten Systems in einem anderen begrifflichen oder gegenständlichen System zu verstehen. Die Nachbildung des Systems erfolgt vor dem Hintergrund des Modellierungszwecks. Im Gegensatz zu einem Simulationsmodell, welches bereits ein ablauffähiges Modell darstellt, dient das konzeptionelle Modell dazu, im Vorfeld der Implementierung des Modells die innere Struktur und das Verhalten des nachzubildenden Systems zu beschreiben, um somit einen Einblick in die Funktionsweise des Systems zu erhalten. Die Phase der Modellbildung ist vergleichbar mit der **Entwurfsphase** (Grob- und Feinentwurf) bei der Produkt- bzw. Softwareentwicklung.

Simulation: In der Phase der Simulation wird zum einen das konzeptionelle Modell in ein Simulationsmodell überführt (Implementierung), zum anderen werden an dieser Stelle die eigentlichen Experimente durchgeführt. Die Implementierung der Simulationsmodelle kann mit konventionellen Programmiersprachen („C“), mit simulationsspezifischen Sprachen („MATLAB“) oder mit graphik-orientierten Simulationsentwicklungsumgebungen („Simulink“, „Dymola“) erfolgen.

Gesamtauswertung und Präsentation: Nach dem erfolgreichen Experimentieren sind die gewonnenen Ergebnisse auszuwerten und dem Modellierungszweck entsprechend darzustellen.

In Abbildung 2 ist der Zusammenhang zwischen der nachzubildenden Realität, dem konzeptionellen Modell und dem Simulationsmodell dargestellt.

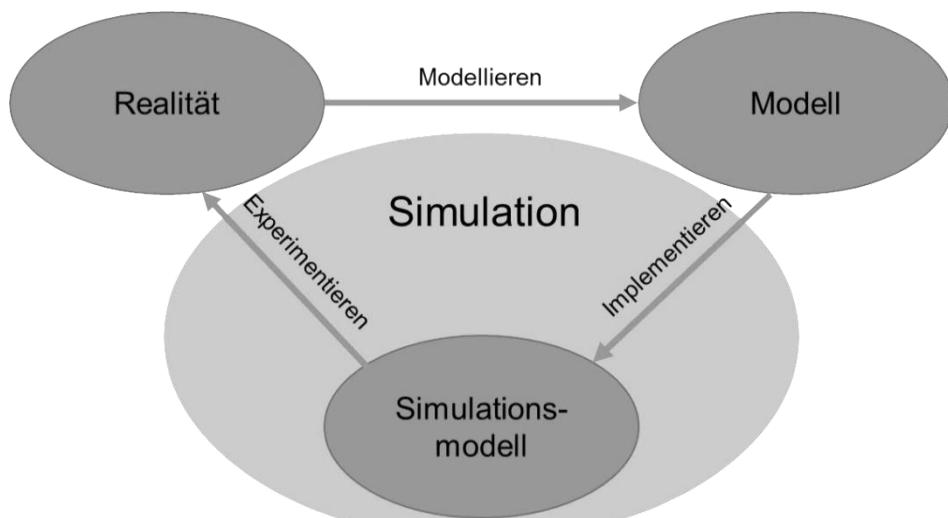


Abbildung 2: Gesamtansicht Modellbildung und Simulation

2.3 Simulationstechnologien

Abhängig vom Ziel und Zweck einer Simulationsstudie (Modellierungszweck) sind die entsprechenden Simulationssysteme unterschiedlich aufgebaut. Im Folgenden werden vier verschiedene Arten von Simulationen, sogenannte Simulationstechnologien, vorgestellt.

2.3.1 Ablaufsimulation

Durch Ablaufsimulationen werden Abläufe von Prozessen (technische, wirtschaftliche, ...) nachgebildet. Dabei sind vor allem die Reihenfolge sowie die Zeitanforderungen der Abläufe im Fokus der Betrachtung. Ziel der Ablaufsimulation ist in erster Linie die Funktionsauslegung und – optimierung, der Funktionsnachweis (\rightarrow Funktionstests) und Risikominimierung technischer Prozesse.

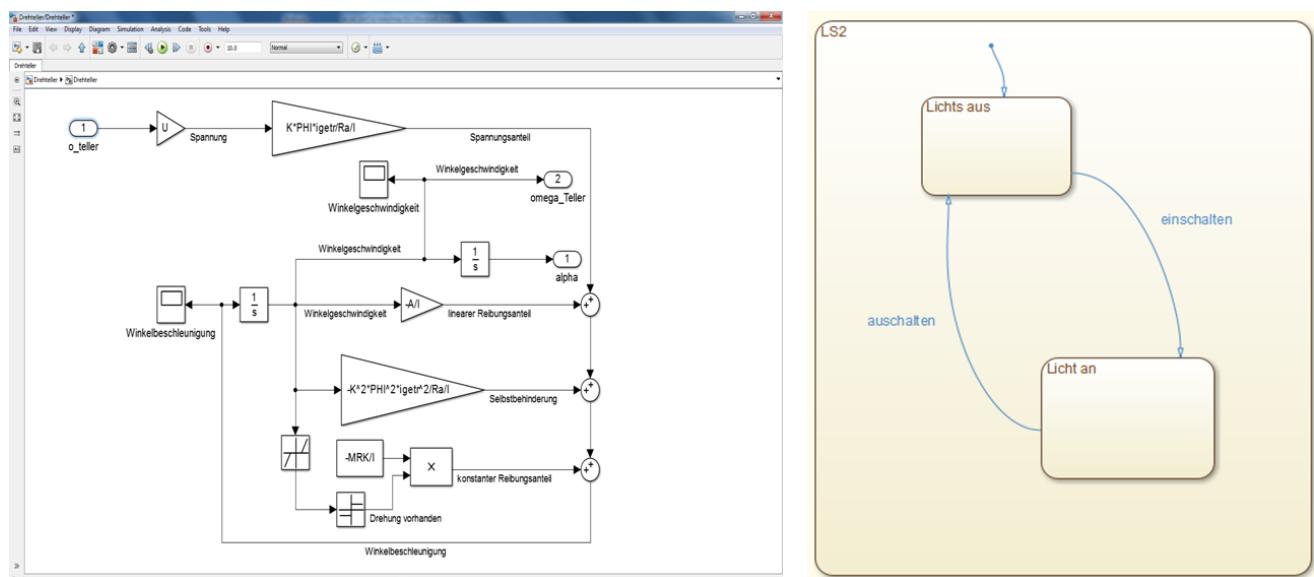
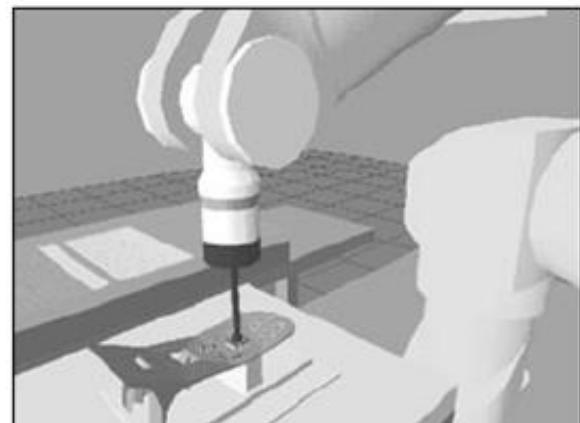


Abbildung 3: Beispiel einer Ablaufsimulation in Simulink / Stateflow

2.3.2 3D-Kinematiksimulation

Die 3D-Kinematiksimulation dient der Analyse und Optimierung von Komponenten eines Produktionsystems. Ziel ist der Test und die Optimierung von geometrischen Auslegungen und Bewegungsabläufen, sowie die Detektierung und Vermeidung von Kollisionen. Die Hauptanwendungsgebiete sind in der Produktentwicklung und speziell auch im Digital Mock-Up zu sehen.



2.3.3 FEM

FEM-Simulation eignet sich zur Nachbildung physikalischer Werkstoffeigenschaften. Dadurch ist es beispielsweise möglich, die mechanische Beanspruchung oder das Schwingungsverhalten einzelner Bauteile bzw. ganzer Baugruppen am Rechner zu analysieren. Das Simulationstool wird als etabliertes Berechnungswerkzeug im Rahmen der Produktentwicklung in der Regel für Werkstoffuntersuchungen, Festigkeitsnachweise und geometrische Auslegungen von Mechanikkomponenten eingesetzt.

Neben mechanischen können auch thermische oder strömungsmechanische Einflüsse und Belastungen nachgebildet werden.

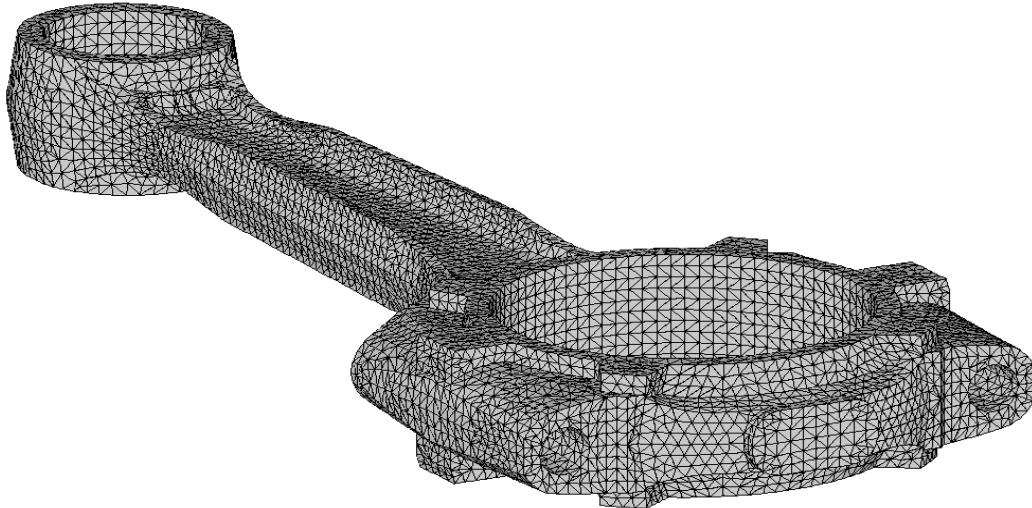


Abbildung 4: FEM-Modell eines Pleuels

2.3.4 Mehrkörpersimulation

Mehrkörpersimulation wird eingesetzt, um das dynamische Verhalten, d.h. die Starrkörperbewegungen sowie das Schwingungsverhalten von technischen Systemen zu analysieren.

Typische Anwendungsgebiete sind:

- Schwingungs-Stabilitätsuntersuchungen in Mehrkörpersystemen
- Analyse und Optimierung von Starrkörperbewegungen
- Berechnung elastischer Bauteilverformungen in Folge der Bewegungsdynamik der Komponente

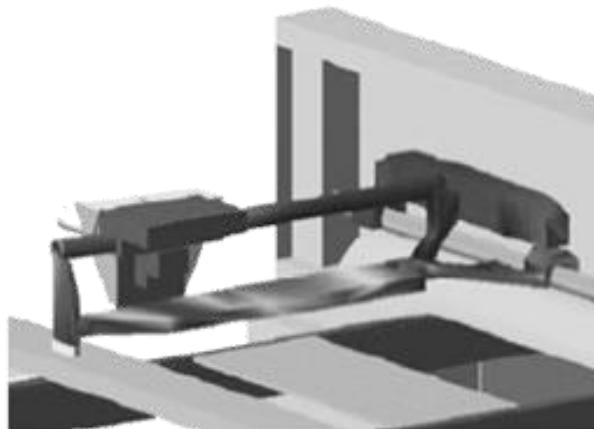


Abbildung 5: Beispiel für Mehrkörpersimulation

2.4 Klassifizierung verschiedener Simulationsarten

Dynamische Simulationen, also Simulationen deren Zustand sich im Laufe der Simulationszeit ändert, können danach klassifiziert werden, ob sich ihre Zustandsvariablen und Simulationszeit diskret oder kontinuierlich ändern:

- **Simulationszeit**

- *Zeitkontinuierlich*: beliebige Zeitpunkte werden simuliert (unendlich viele Zeitpunkte in einem Intervall)
- *Zeitdiskret*: nur dezidierte Zeitpunkte werden simuliert (endlich viele Zeitpunkte in einem Intervall)

- **Zustandsvariablen**

- *Zustandskontinuierlich*: Zustandsvariablen können beliebige Werte annehmen
→ unendliche Anzahl an Zuständen
- *Zustandsdiskret*: es wird nur endliche Anzahl an Systemzuständen simuliert

Diskrete Zustände

Werden von einem System nur bestimmte Zustände (diskrete Zustände) betrachtet, spricht man von einer ereignisorientierten Simulation. Bei einer ereignisorientierten Simulation hätte ein Motor beispielsweise die möglichen Zustände „an“ und „aus“. Sämtliche Zwischenschritte zwischen diesen beiden Zuständen (stetige Geschwindigkeitszu- bzw. -abnahme) werden vernachlässigt.

Kontinuierliche Zustände

Systeme mit unendlich vielen Zuständen (kontinuierliche Zustände) werden nochmals unterteilt. Kann in einer Simulation das nachgebildete System seine Zustände zu jeder beliebigen Zeit ändern, d.h. wird die Simulationszeit als kontinuierlich angenommen, so spricht man von kontinuierlicher (numerischer) Simulation (z.B. ein Motor, bei dem sich kontinuierlich über die Zeit der Geschwindigkeitsverlauf ändert). Wird stattdessen angenommen, dass sich die Zustände in einer Simulation nur zu bestimmten Zeiten ändern können, wird von einer zeitdiskreten Simulation gesprochen.

Hybride Simulationen

Werden ereignisorientierte Simulationen, numerische Simulationen und zeitdiskrete Simulationen miteinander kombiniert, so spricht man von hybriden Simulationen.

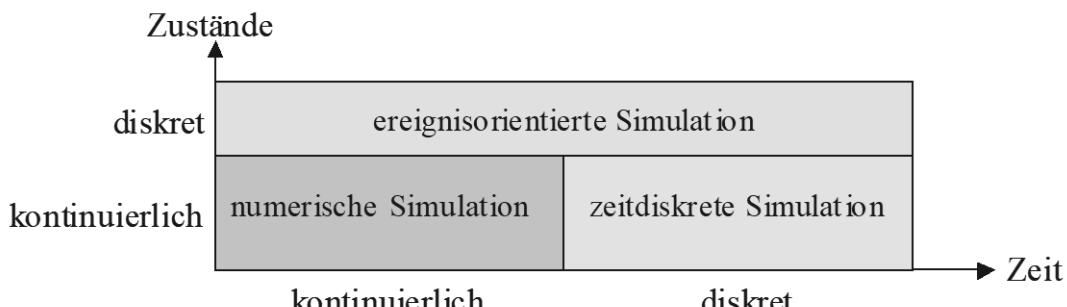


Abbildung 6: Klassifizierung von Simulationen



3 MATLAB

„MATLAB ist ein umfangreiches Softwarepaket für numerische Mathematik. Wie der Name MATLAB, abgeleitet von MATrix LABoratory, schon zeigt, liegt seine besondere Stärke in der Vektor- und Matrizenrechnung. Unterteilt in ein Basismodul und zahlreiche Erweiterungspakete, so genannte Toolboxen, stellt MATLAB für unterschiedlichste Anwendungsgebiete ein geeignetes Werkzeug zur simulativen Lösung der auftretenden Problemstellungen dar.“ [A. Angermann, MATLAB-Simulink-Stateflow].

3.1 Grundlagen zu MATLAB

Im Fokus des Praktikums steht die Modellierung mit Simulink und Stateflow. Trotzdem sollen im Folgenden kurz die Grundlagen zu MATLAB und dessen Benutzung vermittelt werden.

Command Window

Im Command Window können direkt alle Befehle aufgerufen oder Variablen deklariert werden.

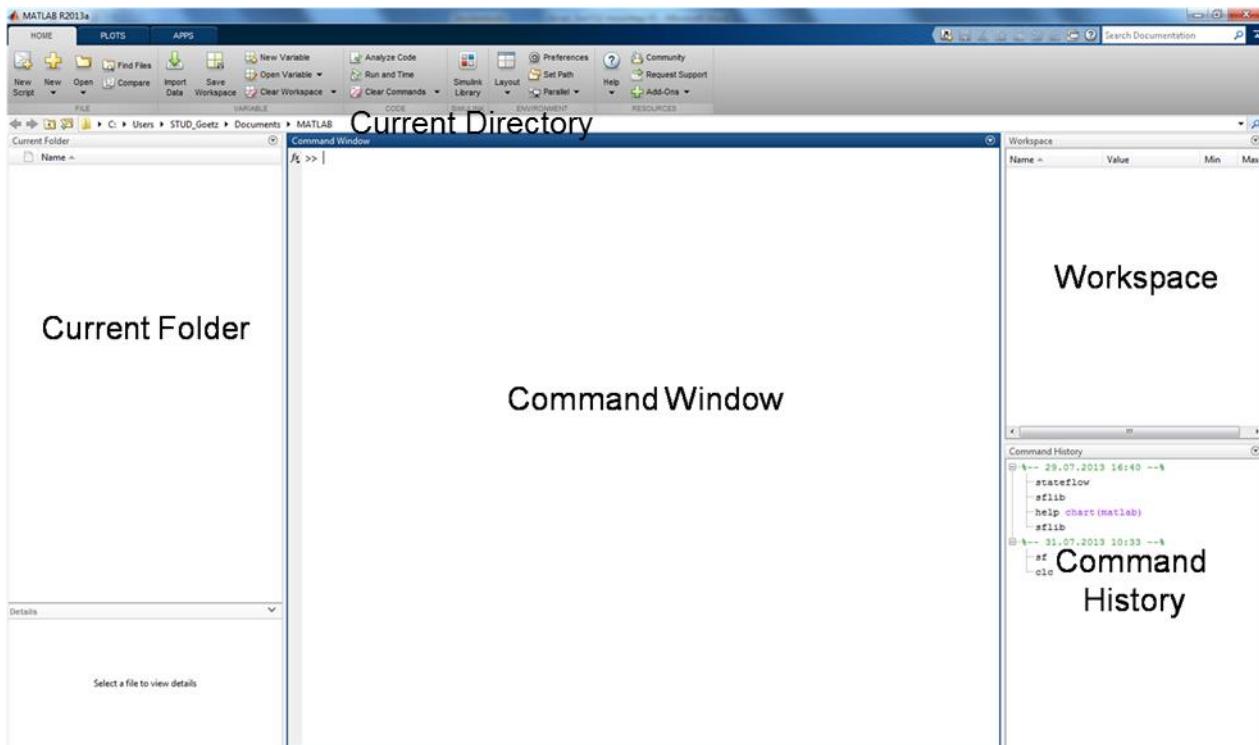


Abbildung 7: MATLAB Desktop

Unter „Current Directory“ sollte während des Praktikums stets das eigene Datenverzeichnis (Z:\) stehen. „Current Directory“ gibt den Pfad an, unter dem MATLAB alle erstellten Funktionen und Modelle sucht, die ausgeführt werden sollten.

Workspace

Im Workspace werden alle Variablen hinterlegt. Es gibt einen globalen Workspace und lokale Workspaces.



3.1.1 Basis-Funktionalität

Variablen

Ähnlich wie in der Programmiersprache C können in MATLAB Variablen zum (Zwischen-) Speichern von Werten verwendet werden.

Die Variablen werden nach einer üblichen Zuweisung eines Wertes (über Eingabe im Command Window) direkt im MATLAB-Workspace deklariert.

```
>> a = 28
```

Hier kann man unterscheiden zwischen globalen Variablen (abgelegt im globalen Workspace) und lokale Variablen (lokaler Workspace). Globale Variablen können direkt über die Eingabe im Command Window in den globalen Workspace abgelegt werden. Variablen, die innerhalb einer Funktion verwendet werden (einschließlich der Ein- und Ausgabevariablen), sind üblicherweise lokale Variablen.

- Einige Variablennamen, wie pi (Kreiszahl π), i bzw. j (imaginäre Einheit) und inf (unendlich) sind in MATLAB reserviert und können entsprechend verwendet werden (siehe `help → elfun`)
- Das Semikolon am Ende der Zeile unterdrückt die Ausgabe der eben eingegebenen Zuweisung.

Beispiel mit Semikolon:

```
>> a = 10 + 5 / 5;  
  
>>  
  
>> a = ((5 + 4) * 2 - 8) / 5;
```

Beispiel ohne Semikolon:

```
>> a = 10+ 5 / 5  
a =  
11
```

- Die Eingabe des Variablennamens ohne Zuweisung gibt den aktuellen Wert der Variable aus.
- In MATLAB gelten die gewöhnlichen Rechenoperationen ($,$, $-$, $*$, $/$, $.$, $()$).

Vektoren und Matrizen

Ein Vektor bzw. eine Matrix werden direkt zwischen eckigen Klammern eingegeben.

- Die Elemente einer Zeile werden durch Komma oder Leerzeichen getrennt und eine neue Zeile wird durch Strichpunkt oder Zeilenumbruch gekennzeichnet.
- Vektor oder eine Matrix durch die Eingabe von ' nach dem Variablennamen zu transponieren.
- Durch ein Semikolon (;) kann eine neue Zeile beim Initialisieren erzeugt werden.
- Mit dem „%“-Zeichen wird ein Kommentar eingeleitet. Alles, was in der Befehlszeile nach dem „%“-Zeichen steht, wird als Kommentar interpretiert.

Im Folgenden sind einige Beispiele zum Befüllen und Auslesen von Vektoren und Matrizen aufgelistet:

Vektor initialisieren

```
>> Zeilenvektor = [1 2 3]  
  
Zeilenvektor =  
  
1 2 3
```

```
>> Spaltenvektor = [1 ; 2 ; 3]  
  
Spaltenvektor =  
  
1  
2  
3
```



Vektor transponieren

```
>> Vektor_trans = Zeilenvektor' %Das Hochkomma erstellt die Transponierte  
Vektor_trans =  
1  
2  
3
```

Matrix initialisieren

Die Initialisierung einer Matrix kann sowohl durch manuelle Zuordnung erfolgen oder über interne MATLAB Funktionen:

- eye(zeilen) % Erzeugen einer Einheitsmatrix (quadratisch)
- ones(zeilen, spalten) % Initialisierung einer Matrix mit „1“
- zeros(zeilen, spalten) % Initialisierung einer Matrix mit „0“
- rand(zeilen, spalten) % Initialisierung einer Matrix mit Zufallswerten zwischen 0 und 1
- randn(zeilen, spalten) % Initialisierung einer Matrix mit normalverteilten Zufallswerten

```
>> Matrix = [Zeilenvektor; 4 5 6]
```

```
Matrix =
```

1	2	3
4	5	6

```
>> Einheitsmatrix = eye(3)
```

```
Einheitsmatrix =
```

1	0	0
0	1	0
0	0	1

```
>> Matrix = ones(3,2)
```

```
Matrix =
```

1	1
1	1
1	1

```
>> Matrix = zeros(2,3)
```

```
Matrix =
```

0	0	0
0	0	0

Elemente auslesen

```
>> erstes_Element = a(1) % Index 1 für das erste Element (nicht 0!!)  
erstes_Element = 1  
  
>> letztes_Element = Matrix(end)  
letztes_Element =  
6  
  
>> b = Matrix(3,2) % Element der dritten Zeile und zweiten Spalte  
b = 1
```



Doppelpunkt-Operator

Der Doppelpunkt Operator ist einer der wichtigsten Operatoren in MATLAB und findet in vielen Fällen Verwendung. Mit seiner Hilfe können spezielle Zeilenvektoren erzeugt werden, z.B. bei der Indizierung in for Schleifen oder beim Plotten. Dabei wird ausgehend von einer Zahl so lange ein Inkrement addiert und in dem Vektor gespeichert, bis ein vorgegebenes Ende erreicht oder überschritten wurde. Wird kein Inkrement angegeben so wird automatisch eine Schrittweite von 1 verwendet. Die allgemeine Syntax ist

Anfangswert:Endwert oder Anfangswert:Inkrement:Endwert

Matrix initialisieren

```
>> Matrix = [1:3;4:6;6:3:12]
```

Matrix =

1	2	3
4	5	6
6	9	12

Zeile auslesen

```
>> ZweiteZeile = Matrix(2,:) %Alle Elemente der 2. Zeile
```

ZweiteZeile =

4	5	6
---	---	---

Bestimmte Bereiche auslesen

```
>> Matrix2 = Matrix(2:end,2:3)
```

Matrix2 =

5	6
9	12

for-Schleife

Durchläuft die Variable i von Anfang bis Ende durch und erhöht ihn in jedem Schritt um den Wert Inkrement. Dabei werden alle Anweisungen zwischen for und end so oft ausgeführt, bis i=ende.

```
>> for i = 1:0.1:1.3
```

```
    Ergebnis = 3*i
```

```
end
```

Ergebnis:

3	3.3000	3.6000	3.9000
---	--------	--------	--------



3.1.2 Hilfefunktion

Das selbstständige Arbeiten im Praktikum kann durch die Hilfefunktion deutlich gesteigert werden.

Der Befehl

```
help FUNKTIONSNAME
```

bietet im Eingabefenster Hilfe zu Funktionen, deren Namen bekannt ist. Die Suche nach Hilfstexten über das Command Window ist dadurch nur empfehlenswert, wenn der Befehl bekannt ist. Falls man für ein Problem eine passende Funktion sucht kann man mit folgendem Befehl eine Suche beginnen:

```
lookfor SUCHBEGRIFF
```

Noch einfacher ist es, über die Taskleiste „Help“ -> Documentation (Shortcut F1) die komplette MATLAB Dokumentation nach einem bestimmten Suchbegriff (Englisch!) zu durchsuchen. Dies sowohl für MATLAB und die im Praktikum verwendeten Toolboxen Simulink und Stateflow.

Beispiel:

```
>> help sin  
  
sin - Sine of argument in radians  
  
This MATLAB function returns the sine of the elements of X.  
  
Y = sin(X)  
  
Reference page for sin  
  
See also asin, asind, sind, sinh  
  
Other functions named sin  
  
fixedpoint/sin, symbolic/sin  
  
>> help %Aufruf aller Hauptkategorien  
  
>> help elfun %Aufruf elementarer mathematischer Funktionen
```

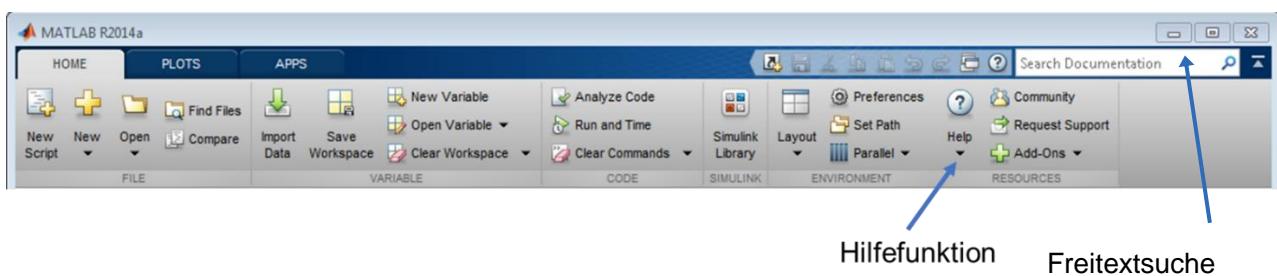


Abbildung 8: Hilfefunktion in der Toolbar

```
>> lookfor eigenvalues  
  
eigshow - Graphical demonstration of eigenvalues and singular values.  
expmdemo3 - Matrix exponential via eigenvalues and eigenvectors.  
condeig - Condition number with respect to eigenvalues.  
eig - Eigenvalues and eigenvectors.  
ordeig - Eigenvalues of quasitriangular matrices.
```



3.1.3 m-Files: Skripte und Funktionen

In MATLAB erstellte Programme liegen als sogenannte m-files (Dateiendung .m) vor. Es existieren zwei Arten von Programm- oder m-files: **Skripte (Script)** und **Funktionen (Function)**.

Allgemeines:

- Aufgerufen wird ein m-File wie jede andere Funktion im Command-Window oder in einem beliebigen anderen m-File.
- Ein neues m-File kann man unter **New→Script** für Skripte oder **New→Function** für Funktionen erzeugen. Ein Editor öffnet sich anschließend automatisch. Beim Speichern wird die Endung „.m“ automatisch angehängt.
- Ein M-File kann im Command-Window gestartet werden, indem man dort den Namen des M-Files eingibt (ohne die Endung „.m“). Außerdem kann der M-File im Current Directory durch Rechtsklick→Run gestartet werden.
- Der Dateiname darf keine Umlaute und Sonderzeichen außer Unterstrich erhalten.

Skripte

Ein Skript ist eine externe Datei, die eine Folge von Anweisungen enthält. MATLAB arbeitet diesen Skript genauso ab, wie wenn diese Anweisungen direkt nacheinander «*von Hand*» im Command Window eingegeben würden.

- **Programmzeilen:** Meistens wird man auf jeder Programmzeile nur einen Befehl notieren. Ein **Strichpunkt** nach dem Befehl verhindert, dass die erzeugten Werte im Command Window ausgegeben werden.
- **Kommentare** werden mit dem **Prozentzeichen %** gekennzeichnet. Alles, was auf einer Programmzeile nach diesem Zeichen erscheint, wird von MATLAB nicht interpretiert.

Beispiel:

```
%Parameterdatei für die Simulationsanlage  
%Anlage  
MaxFlaschen = 25;           %Anzahl der Flaschen, die max. durchlaufen dürfen  
%Drehteller  
PHI = 0.001;                 % Hauptfluss im Elektromotor  
U = 24;                      % [V] angelegte Spannung am Elektromotor  
Ra = 40;                     % [Ohm] Ankerwiderstand des Elektromotors  
I = 0.5;                     % [kgm2] Massenträgheit des Tellers  
K = 8;                       % Wicklungsanzahl  
MRK = 1;                     % Konstante Reibungsmoment  
A = 5;                       % Konstante für geschwindigkeitsabhängige Reibung  
igetr = 1080;                 % Getriebeübersetzung
```



Funktionen

Selbst programmierte Funktions-m-Files werden auf die gleiche Weise angewandt wie die von MATLAB bereitgestellten Funktionen wie z.B. cos (Cosinusfunktion), d.h. sie werden mit ihrem Namen (= Name des m-Files) und entsprechenden Argumenten aufgerufen:

The screenshot shows the MATLAB Editor with a file named 'addiere.m'. The code is as follows:

```
1 function [ output_args ] = addiere( input_args )
2 %ADDIERE Summary of this function goes here
3 % Detailed explanation goes here
4
5
6 end
```

Annotations with arrows point to specific elements:

- 'Funktionsname' points to the function header 'function [output_args] = addiere(input_args)'.
- 'Rückgabeparameter' points to 'output_args'.
- 'Eingabeparameter' points to 'input_args'.
- 'Warnings' points to the status bar at the bottom right.
- 'Breakpoints' points to a red circular marker at line 6.

Eine Funktion zeichnet sich dadurch aus, dass ihr (meistens) Werte übergeben werden, die sie verarbeiten soll, und dass sie die Resultate an den «Aufrufer» (aus dem Command Window oder aus einem anderen Skript bzw. Funktion) zurückgibt.

- Der **Name der Funktion** muss mit dem Dateinamen übereinstimmen, unter dem die Funktion als m-file abgespeichert ist.
- **Eingabeparameter** (inputargs in1, in2,...) legen fest welche Werte der Funktion übergeben werden
- **Rückgabeparameter** (outputargs out1, out2,...) spezifizieren die Werte, durch die die Funktion die Resultate der Berechnung zurückgeben kann
- Neben selbst erstellten Funktions-m-Files gibt es schon in **MATLAB implementierte Funktionen** wie z.B. mathematische Funktionen (*help elfun*) oder die zahlreichen Toolboxen
- Neben der Scrollbar findet man ein **farbiges Quadrat**. Dies kann die Farben „grün“ (=alles in bester Ordnung), „orange“ (=Warning. Programm hat keine Fehler, kann aber noch optimiert werden) oder „rot“ (=Syntaxfehler).
- **Breakpoints** können gesetzt werden um ein besseres Debuggen zu ermöglichen, um z.B. eine Funktion Schritt für Schritt zu debuggen bei Fehlersuche

Beispiel für eine Funktion:

The screenshot shows the MATLAB Editor and Command Window. The Editor contains the following code:

```
1 % Function "addiere"
2 function [ergebnis] = addiere(sum1, sum2)
3 - ergebnis = sum1 + sum2;
4 - end
5
```

The Command Window shows the execution:

```
>> [ergebnis] = addiere(sum1, sum2)

ergebnis =
```

The result is 15. To the right, a table displays variable values:

Name	Value	Min	Max
ergebnis	15	15	15
sum1	5	5	5
sum2	10	10	10

3.2 Aufgaben

3.2.1 Eingabe von Variablen

Folgende Operationen sind im Command-Window auszuführen

- Erzeugen sie folgende Variablen im MATLAB-Workspace

$$a = \begin{bmatrix} 11,5 \\ 20,9 \\ 13 \end{bmatrix} \quad b = [5 \ 10,5 \ 20]$$

- Berechnen Sie das Skalarprodukt
 - 1) von a mit b
 - 2) von b mit a
- Erzeugen Sie ohne direkte Eingabe der Einzelwerte die Matrix:

$$M = \begin{bmatrix} & b_1 \\ a & b_2 \\ & b_3 \end{bmatrix}$$

3.2.2 Erzeugen eines Skripts

Erzeugen Sie ein M-File, das aus gespeicherten Variablen a und b die Matrix M berechnet und diese dann im Workspace ablegt.

Speichern Sie die Datei unter dem Namen ‚CalcMatrix.m‘ ab.

3.2.3 Erzeugen einer Funktion

Schreiben Sie das gerade erzeugte M-File zu einer Funktion um. Dabei sollten die beiden Vektoren a und b beim Funktionsaufruf mit übergeben werden und die Funktion soll M zurückliefern.

Speichern Sie die Datei unter dem Namen ‚FunkCalcMatrix.m‘ ab.



4 Simulink

Simulink ist ein Erweiterungspaket, eine sog. Toolbox von MATLAB. Es ermöglicht eine graphische Modellierung von physikalischen Systemen mittels Signalflossgraphen. Die Basis eines Signalflossgraphen sind Funktionsbausteine. Solche Bausteine sind durch Ein- und Ausgänge, sowie Namen und Blockicon gekennzeichnet (vgl. Abbildung 9: Integrator Funktionsbaustein). Je nach Funktion werden innerhalb eines Bausteins Signale erzeugt oder verarbeitet. Signalfüsse werden über Flussrelationen (Pfeile), die Blöcke miteinander verbinden, dargestellt.

Signale haben außerdem Datentypen, welche durch die Funktionsblöcke implizit festgelegt werden. Nicht alle Funktionsblöcke akzeptieren alle Datentypen als Eingang, weshalb in einigen Fällen ein vorgeschalteter „Converter“-Block notwendig ist.



Abbildung 9: Integrator Funktionsbaustein

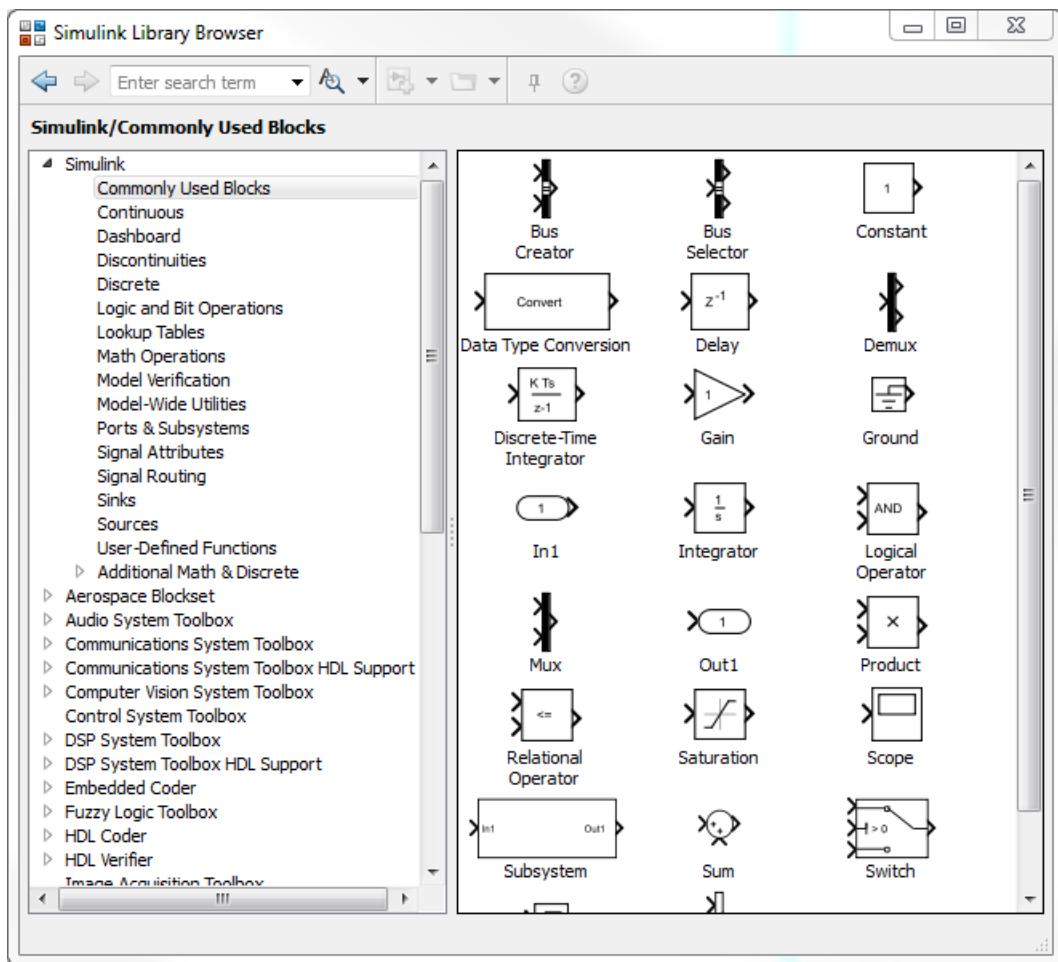


Abbildung 10: Der Library Browser



4.1 Beschreibung von Simulink

Simulink ist in seiner Funktionalität sehr umfangreich. Deshalb wird in den folgenden Abschnitten nur ein Teilbereich von Simulink betrachtet. Die dadurch gewonnenen Kenntnisse lassen sich jedoch sehr gut übertragen.

4.1.1 Starten von Simulink

- Gestartet wird Simulink durch die Eingabe von „simulink“ in das MATLAB Command-Window oder durch Klicken auf das Simulinksymbol („Simulink Library“) in der MATLAB Toolbar. Durch die Auswahl von „Blank Model“ auf der „Simulink Start Page“ wird ein neues Model erstellt.
- Alle Funktionsbausteine findet man „Library Browser“, den Sie durch Klicken des entsprechenden Icons in der Toolbar öffnen können.
- Funktionsbausteine können per Drag and Drop zum Model hinzugefügt werden.
- Durch einen Doppelklick auf einen Funktionsbaustein lassen sich alle nötigen Parameter einstellen. An diesem Punkt sei noch einmal auf die ausführliche Hilfefunktion von Matlab verwiesen.

4.1.2 Wichtige Shortcuts

Tastenkombination	Funktion
Strg + R	Rotiert einen Block
Strg + I	Invertiert einen Block
Rechte Maustaste + ziehen	Erstellt ein Duplikat des Ursprungsblocks/ Abzweigen eines Signals
Doppelklick	Öffnet das Einstellungsmenü
Leertaste	Passt Model an Fenstergröße an
Rechte Maustaste -> help	Aufruf der Hilfe zum Funktionsblock

4.1.3 Simulink Bibliotheken

Die Simulink Bibliotheken beinhalten einzelne, elementare Bausteine zum Bilden von Systemen. Im Folgenden werden die wichtigsten Bibliotheken vorgestellt. Eine ausführliche und vollständige Beschreibung der Bibliotheken finden Sie unter „Block Libraries“ in der Matlab Hilfe.

Die Funktionsblöcke sind immer so universell wie möglich gehalten. Eine Spezifizierung muss daher in den Einstellungen des Blocks vorgenommen werden, z.B. kann der Block „trigonometric function“ den Sinus, Cosinus, etc. vom Eingang bilden. Ähnlich kann der Block „logical Operator“ ein AND, OR, XOR, etc. darstellen.

Sources

In der Sources Library finden Sie ausschließlich Funktionsblöcke, die Signale direkt erzeugen oder aus dem Workspace einlesen. Ein Auszug ist in der folgenden Tabelle dargestellt.

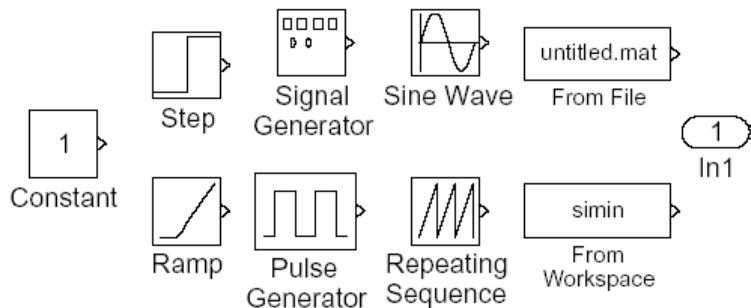


Abbildung 11: Beispiele für Sources

Name	Funktion
Constant	Erzeugt ein konstantes Signal
Clock	Gibt den aktuellen Wert der absoluten Simulationszeit aus
Ground	Erdet unverbundene Eingangssignale
Ramp	Gibt ein kontinuierlich steigendes Signal aus
Step	Erzeugt ein Sprungsignal
Signal Builder	Erzeugt ein benutzerdefiniertes Signal

Sinks

Funktionsbausteine aus Sinks (Becken, Spüle) haben keinen Ausgang. Sie dienen meistens der Anzeige von Signalen und der Ausgabe von Signalen in den Workspace oder einer Datei. Wie immer bieten auch diese Blöcke oftmals vielfältige Einstellungsmöglichkeiten.

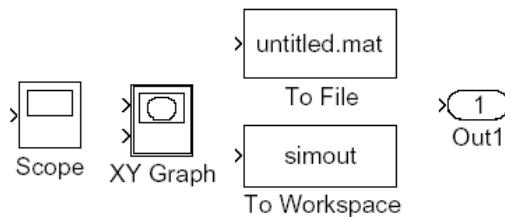


Abbildung 12: Beispiele für Sinks

Name	Funktion
Scope	Virtueller Oszilloskop, plottet Eingänge über die Zeit
To Workspace	Schreibt Daten in den Matlab Workspace
Display	Zeigt den Wert seiner Eingänge
Terminator	„Abstellgleis“ für unverbundene Blockausgänge



Math Operations

Mittels der Math Operations lässt sich ein Signal mathematisch und logisch verändern oder mit anderen vergleichen.

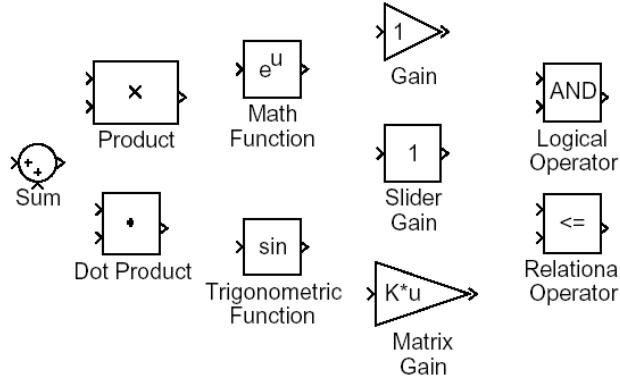


Abbildung 13: Beispiele für Math Operations

Name	Funktion
Gain	Skaliert das Signal
Product	Multipliziert, dividiert Eingänge miteinander
Sum	Addieren und Subtrahieren der Eingänge
Abs	Bildet Absolutwert des Eingangs
Trigonometric Function	Berechnet spezifizierte trigonometrische Funktion
Sign	Bestimmt Vorzeichen des Eingangs
Math Function	Wertet in den Einstellungen parametrisierte Funktion aus

Continuous

Funktionsblöcke innerhalb dieser Bibliothek können zur Modellierung von kontinuierlichen Systemen verwendet werden.

Name	Funktion
Integrator	Integriert das Eingangssignal über die Simulationszeit
Derivative	Bestimmt Änderung des Signals über die Zeit (Ableitung)



Logic and Bit Operations

In dieser Bibliothek befinden sich sämtliche Funktionsbausteine für logische Operatoren.

Name	Funktion
Logical Operator	Wertet die spezifizierte logische Operation aus
Compare To Constant	Vergleicht Eingang mit spezifiziertem Wert
Intervall Test	Testet, ob Eingang in spezifiziertem Intervall liegt

Signal Routing

In Abbildung 14 sind Funktionsblöcke zur Datenspeicherung und Datenmanagement, sowie zur Verknüpfung und Auswahl von Signalen dargestellt.

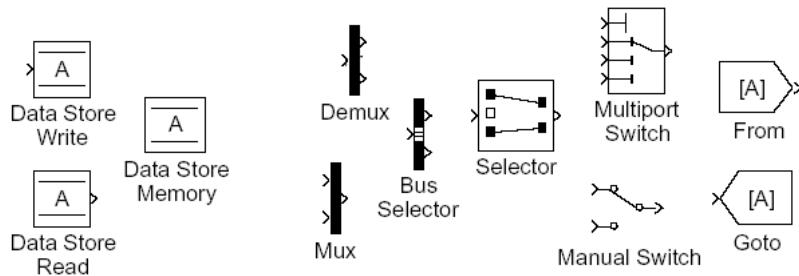


Abbildung 14: Beispiele für Signal Routing

Name	Funktion
Mux	Multiplexer, erstellt aus Eingangssignalen einen Vektor
Demux	Konvertiert einen Vektor in einzelne Signale
Bus Creator	Erstellt einen Bus aus Signalen
Bus Selector	Wählt definiertes Signal aus einem Bus aus
Goto	Transportiert Signal zu entsprechendem „From“ Block
From	Empfängt Signal von entsprechendem „goto“ Block

Ports & Subsystems

Beinhaltet Blöcke für Subsysteme. Auf Subsysteme wird an anderer Stelle ausführlich eingegangen.

Name	Funktion
Subsystem	Stellt ein Untersystem in einem System dar (Gruppierung)
Enabled Subsystem	Subsystem, aktiviert durch externes Signal
Triggered Subsystem	Subsystem, ausgelöst durch externes Signal

4.1.4 Subsysteme

Subsysteme dienen zum übersichtlichen Gestalten komplexer Modelle. Durch sie kann man aus mehreren Funktionsblöcken und deren Verknüpfungen einen Block gestalten, der das gleiche Verhalten besitzt.

- Dadurch ist ein Aufbau in hierarchischen Strukturen möglich (Systemgedanken).
- Um Schnittstellen nach außen zu erhalten, sind In- und Out-Ports im Subsystem anzulegen.
- Man kann jedoch auch automatisch ein Subsystem erzeugen lassen, indem man alle Blöcke und Verbindungen markiert, die in das Subsystem enthalten soll, auf ein markiertes Element mit der rechten Maustaste klickt und aus dem Kontextmenü Create Subsystem auswählt.
- Bereits bestehende Subsysteme können durch einen Doppelklick geöffnet werden.

Normale Subsysteme

Dienen zur besseren Übersicht und zur einfacheren Handhabung einzelner Teilsysteme. Diese Systeme laufen mit der gleichen Simulationsschrittweite wie das übergeordnete System ab. Durch einen Doppelklick auf das Subsystem kann dieses geöffnet werden.

Getriggerte Subsysteme (Triggered Subsystem)

Getriggerte Subsysteme verfügen zusätzlich über einen Triggereingang. Dieser überwacht das anliegende Signal auf auftretende Flanken. Dieser Triggereingang löst ein **Event** aus, das einen **einmaligen Durchlauf** des Systems bewirkt.

Dieses Event kann sowohl bei einer steigenden Flanke (*rising edge*), einer fallenden Flanke (*falling edge*) oder durch beide (*both*) ausgelöst werden. Die Einstellung hierfür wird in den Blockparametern des „Trigger“-Blocks vorgenommen. Durch Doppelklick auf den Block „Trigger“ im Subsystem gelangt man zum Parameterfenster.

Enabled Subsysteme (Enabled Subsystem)

Ein Enabled Subsystem besitzt zusätzlich einen Enabler-Port. Das Subsystem wird **so lange** ausgeführt, wie das **Enable-Signal True** ist (>0). Solange ein Eingangssignal anliegt, wird die Funktion im enabled Subsystem entsprechend des eingestellten Simulationstakts wiederholt. **Beispiel**

Das folgende System (vgl. Abbildung 15) besteht aus einem normalen, einem getriggerten und einem enabled Subsystem. Die Subsysteme führen allesamt den Eingang direkt weiter in den Ausgang. Das triggered Subsystem ist für „rising Edge“ konfiguriert.

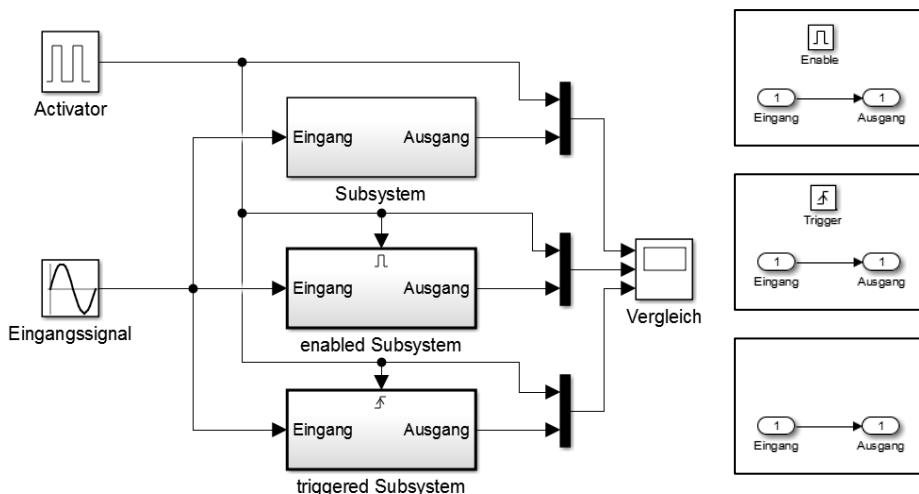


Abbildung 15 Beispielsystem mit mehreren Subsystemen

Somit zeigt der Vergleich im Scope (vgl. Abbildung 16) das distinktive Verhalten der jeweiligen Subsysteme. Ein besonderes Augenmerk ist dabei auf die Eigenschaft zu legen, dass die Ausgänge des enable/triggered Subsysteme bei Inaktivität auf einem konstanten Leven bleiben.

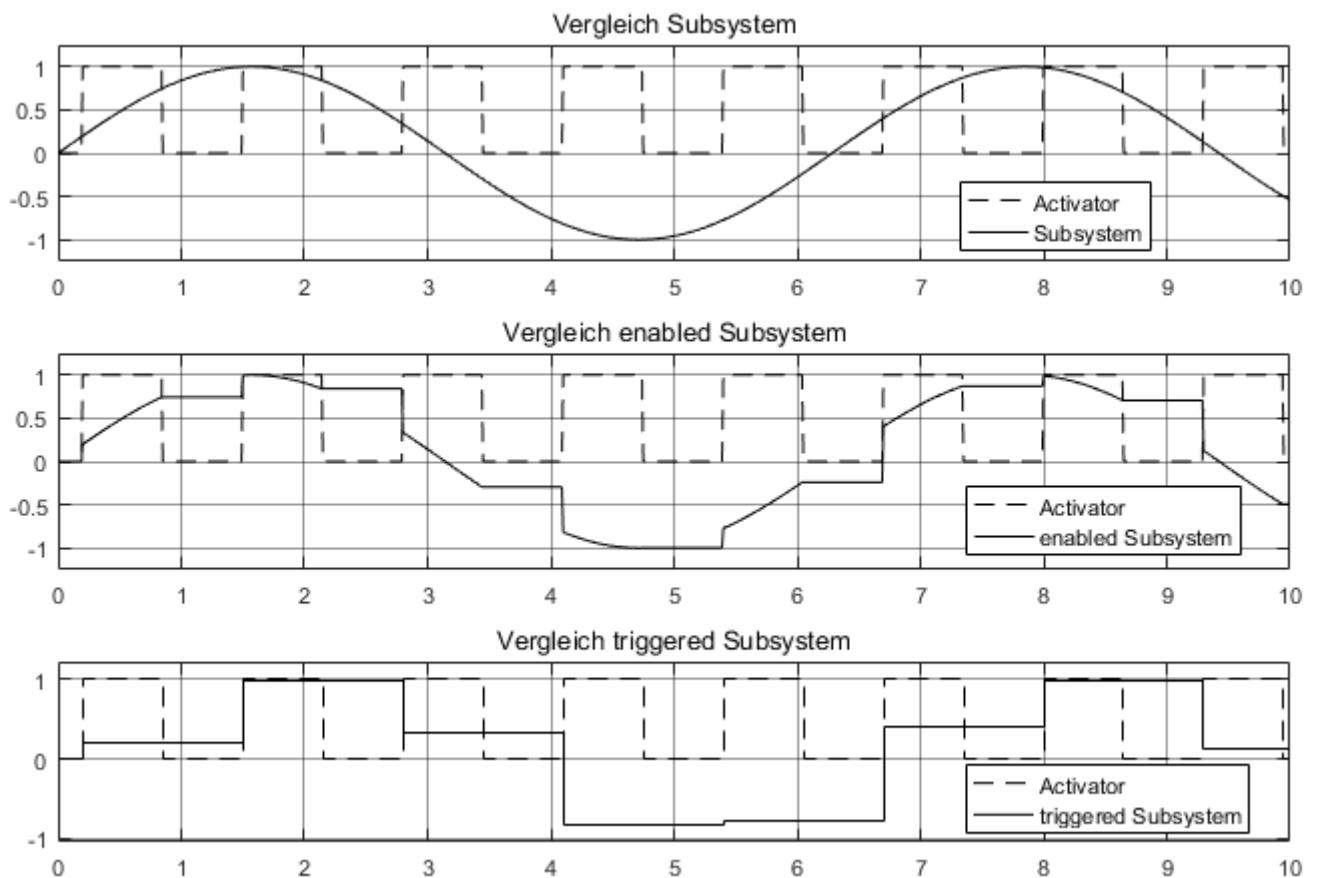


Abbildung 16 Ausgabe des Scopes „Vergleich“



4.1.5 Das Modellfenster

In Abbildung 17 ist ein Beispiel eines fertigen Simulink-Modells

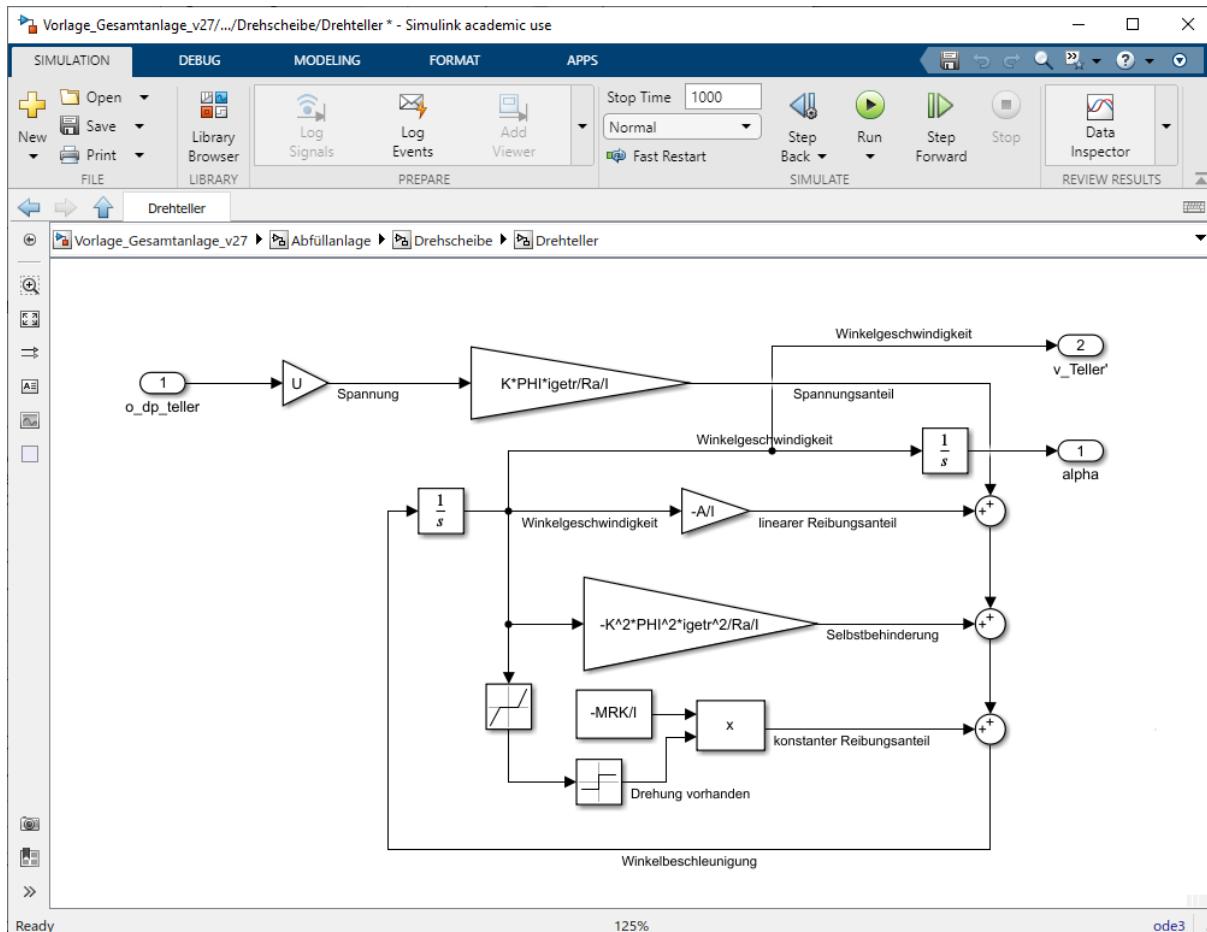


Abbildung 17: Beispiel eines fertigen Modells

- Durch einen Klick auf dieses Symbol kann das Modell gespeichert werden.
- Durch einen Klick auf dieses Symbol kann die Simulation gestartet werden. Nach dem Start verändert sich das Symbol in ein Pausezeichen, welches das Anhalten der Simulation ermöglicht.
- Durch einen Klick auf dieses Symbol kann man die Simulation abbrechen (nur während der Simulationsausführung möglich).
- Durch einen Klick auf dieses Symbol wird der Library Browser geöffnet.
- Durch einen Klick auf dieses Symbol wird ein neues Simulink Modell geöffnet

Im Menüpunkt „Debug → Information Overlays“ können verschiedene Eigenschaften von Signalen, wie den Datentyp („Base Data Types“) oder die Größe („Signal Dimensions“) direkt im Modell angezeigt werden. Es empfiehlt sich diese anzeigen zu lassen, da dies häufige Fehlerquellen sind



4.1.6 Simulationsparameter

Mittels der Simulationsparameter lassen sich alle für die Simulation wichtigen Eingaben tätigen. Hierunter fallen die Startzeit und das Ende der Simulation, die Auswahl des Algorithmus zur numerischen Integration, die Genauigkeit, die Schrittweiten und das Verhalten bei Fehlern.

Die Einstellung der Simulationsparameter erfolgt im Modellfenster unter dem Menüpunkt „Modeling→Model Settings“ oder über den Shortcut Ctrl+E (Abbildung 18).

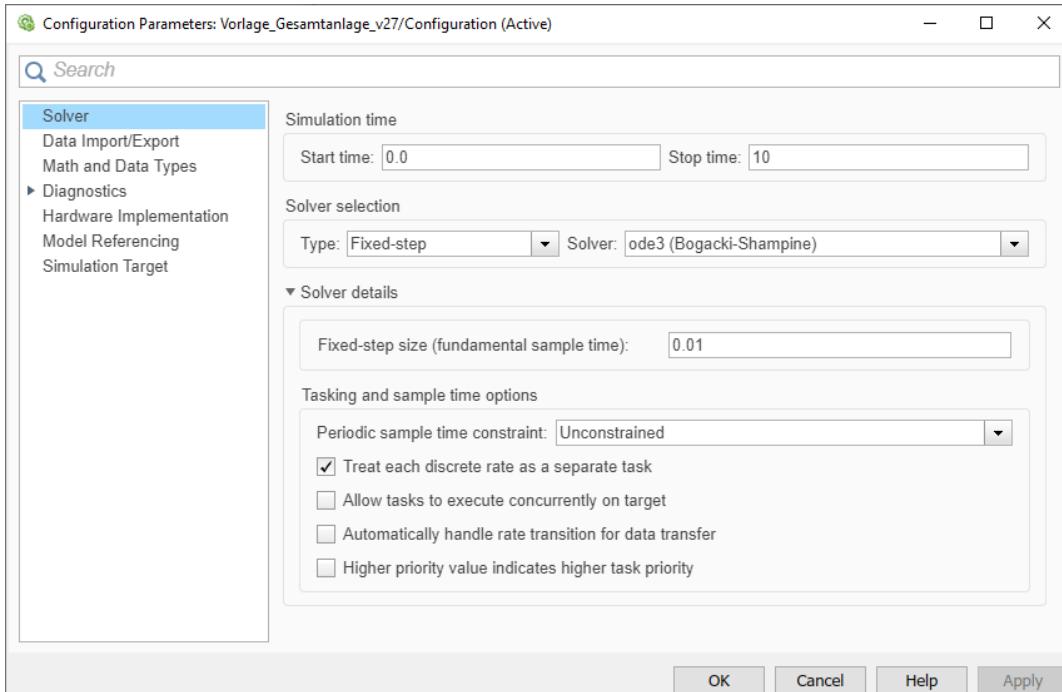


Abbildung 18: Der Simulationsparameterdialog

Simulation time: Einstellung der Start und Endzeit der Simulation

Solver options: Type: **Fixed Step**

Solver: **ode3 (Bogacki-Shampine)**

Fixed-step size: **0.01**

Hier muss während des Praktikums immer „Fixed-step“ und eine maximale Schrittweite von 0.01s eingestellt werden, da es ansonsten zu Fehlern in der Modellberechnung kommen kann.

Hintergrund:

Das Problem in der kontinuierlichen Simulation besteht darin, dass ein Rechner nur diskret arbeiten kann. Um ein möglichst gutes Ergebnis zu erhalten, kann die Schrittweite verkleinert und der Fehler verringert werden. Dadurch erhöht sich Rechenaufwand und -zeit (Fixed-step).

Eine weitere Möglichkeit besteht darin, die Schrittweite an kritischen Punkten zu verkleinern und an unkritischen Punkten zu vergrößern (Variable-step). Dadurch erhält man ein zufriedenstellendes Ergebnis, ohne eine Erhöhung der Rechenzeit in Kauf nehmen zu müssen. Um die kritischen Punkte zu erhalten, ist bei Variable-step eine Fehlerüberwachung und eine zero-crossing Erkennung durchzuführen. Die Genauigkeit kann man auch durch Ändern der Ansatzfunktion erhalten (z.B. kubisch statt quadratisch). Dies geschieht in Simulink durch die Auswahl der verschiedenen Solver. Mit der Ansatzfunktion wird die zu berechnende Funktion in einem kleinen Teil angenähert.

Je höher der Grad einer Ansatzfunktion ist, desto besser kann die Ursprungsfunktion in der Regel abgebildet werden. Es sind aber dadurch auch mehr Rechenoperationen nötig.



4.1.7 Diagnosefenster

Sollten vor und während der Simulation Fehler auftreten, werden diese im Diagnosefenster ausgegeben. Die Fehler werden durch deren Art, Ursache, Ort und Komponente beschrieben (Abbildung 19).

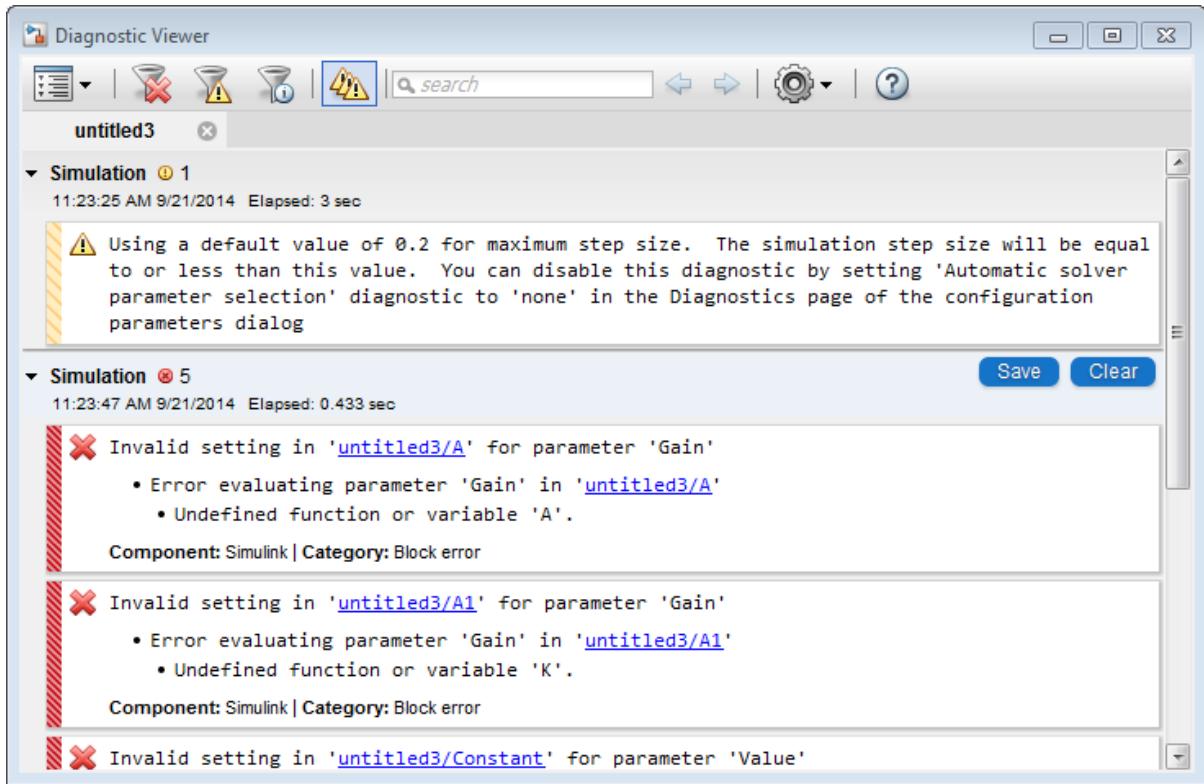


Abbildung 19: Das Diagnosefenster

- Man findet hier eine Auflistung aller aufgetretenen Fehler und Warnungen.
- Bei jedem Fehler oder jeder Warnung findet man eine detaillierte Beschreibung des ausgewählten Fehlers mit Link auf den Fehler verursachenden Block gegeben.
- Solange das Diagnosefenster geöffnet ist, sind Blöcke, die Fehler verursachen, innerhalb des Modells farblich markiert. **Die blauen Textteile in der Fehlerbeschreibung sind anklickbar und führen direkt zum fehlerhaften Block.**
- Typische Fehler
 - Fehlerhafter Datentyp, oder eine
 - Falsche Signalgröße
 - Syntaxfehler
 - Parameterfile nicht geladen oder Parameter nicht definiert

Hinweis:

Es ist wichtig, die Fehler oder Warnung zu lesen und interpretieren zu können. Fehlersuche und -behebung ist es ein fundamentaler Bestandteil beim Arbeiten mit MATLAB, Simulink und Stateflow!

4.2 Vorgehen zur Erstellung eines kontinuierlichen Systems

Anhand der folgenden drei Beispiele wird das prinzipielle Vorgehen zum Erstellen eines Modells dargestellt. Es soll hier nicht der schnellste, sondern ein möglichst allgemeingültiger Weg aufgezeigt werden.

4.2.1 Federpendel

In diesem Kapitel soll ein Federpendel modelliert werden. Es handelt sich dabei um eine ungedämpfte, freie Schwingung mit normierter Einheitsfrequenz $\omega_0 = 1$. Am Anfang ist das Pendel in Ruhe und um $x = 1$ ausgelenkt.

Aufstellen des mathematischen Modells

Mittels verschiedener Lösungsmethoden (Lagrange, Newton-Euler,...) kann man die Differentialgleichung des Systems erhalten. In unserem Fall lautet diese

$$\ddot{x}(t) + \omega_0^2 \cdot x(t) = 0$$

Anfangsbedingungen: $\dot{x}(t=0) = 0, x(t=0) = 1$

Umschreiben zur Verwendung in Simulink

Um die Differentialgleichung in Simulink zu modellieren, wird diese nach der höchsten Ableitung aufgelöst:

$$\ddot{x}(t) = -\omega_0^2 \cdot x(t)$$

Übertragung zu Simulink

- Zunächst erfolgt eine zweifache Integration der Beschleunigung \ddot{x} mit Hilfe zweier Integrator-Blöcke.

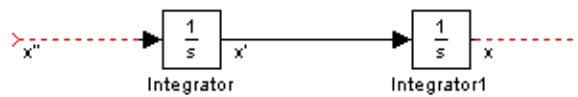


Abbildung 20: Blockschaltbild zu Schritt 1

- Es wird nun sukzessive die rechte Gleichungsseite der umgeformten Differentialgleichung mit Hilfe von Integratoren, Additionsstellen etc. aufgebaut. Im vorliegenden Falle wird der Ausgang des zweiten Integrators x im Block Gain mit $-\omega_0^2$ multipliziert.



Abbildung 21: Blockschaltbild zu Schritt 2

- Zum Schluss erfolgt die Rückkopplung. Da der Ausgang des Blocks Gain die Beschleunigung \ddot{x} der Differentialgleichung darstellt, wird er mit dem Integratoreingang verbunden.

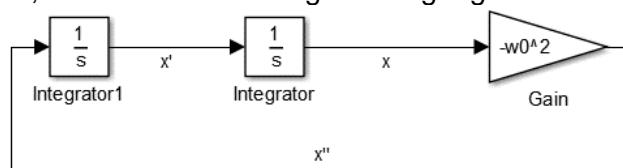


Abbildung 22: Blockschaltbild zu Schritt 3

Randbedingungen einbringen Nun sind noch die Anfangsbedingungen der beiden Integratoren zu setzen. Ausgang des letzten Integrators ist der Weg x , also wird dieser durch Doppelklicken und Eintrag des Wertes 1 in das Feld „Initial condition“ mit der Anfangsbedingung initialisiert. Der erste Integrator liefert die Geschwindigkeit. Dieser Block wird mit der Anfangsgeschwindigkeit 0 initialisiert.

4.2.2 Drehtellerbewegung

Der Drehteller ist eine Scheibe, auf der die Flaschen abgestellt werden können. Durch die Drehbewegung des Tellers können Flaschen auf einer Kreisbahn transportiert werden.

Ein- und Ausgangsgrößen

Vor der Modellierung eines Systems sind die Ein- und Ausgangsgrößen des zu modellierenden Systems zu bestimmen.

- Der Drehteller verfügt über die Eingangsgröße o_{teller} (logisches Signal, das angibt, ob die Stromzufuhr zum Motor an- oder ausgeschaltet ist)
- Die Ausgangsgrößen sind Drehgeschwindigkeit ω und den Drehwinkel α

Aufstellen der Bewegungsgleichung

Im vorliegenden Beispiel stellt sich diese Wirkkette wie folgt dar: Das Eingangssignal schaltet ein Relais (Verstärker). Die Relaisspannung wird an einen Elektromotor angelegt. Dieser wandelt die elektrische Energie in ein Moment um. Das Moment treibt den Drehteller an. Die Drehtellerbewegung ergibt, über die Zeit integriert, den aktuellen Drehwinkel α , der die Ausgangsgröße darstellt. Die Bewegungsgleichungen werden über die Momentengleichgewichte des Tellers und des Elektromotors ermittelt.

Momentengleichgewicht des Tellers:

$$M_{\text{el}} \cdot i = I \cdot \dot{\omega} + M_R = I \cdot \dot{\omega} + \text{sgn}(\omega) \cdot M_{RK} + A \cdot \omega$$

M_{el}	Drehmoment des Elektromotors
i	Gesamtübersetzung
I	Trägheitsmoment des Drehtellers
α	Drehwinkel
$\omega = \dot{\alpha}$	Drehgeschwindigkeit
$\dot{\omega}$	Drehbeschleunigung
M_{RK}	konst. Reibmoment
M_R	Reibmoment ($= M_{RK} + A \cdot \omega$)
A	geschwindigkeitsabhängige Reibung
sgn	Signumfunktion $\text{sgn}(x) = \frac{ x }{x}$ für alle $x \neq 0$, sonst $\text{sgn}(x=0) = 0$

Momentengleichung eines Gleichstrommotors:

$$M_{\text{el}} = \frac{K \cdot \Phi}{R_a} \cdot (U - K \cdot \Phi \cdot \Omega)$$

K	Wicklungsanzahl
Φ	Hauptfluss im Elektromotor
R_a	Ankerwiderstand des Elektromotors
U	angelegte Spannung
Ω	Drehzahl des Elektromotors
$\Omega = \omega \cdot i$	Koppelbedingung

Übertragen der Bewegungsgleichung nach Simulink

Um die Bewegungsgleichung nach Simulink zu übertragen, wird diese nach der höchsten, abgeleiteten Größe aufgelöst:

$$\dot{\omega} = \frac{M_{el} \cdot i - \text{sgn}(\omega) \cdot M_{RK} - A \cdot \omega}{I} = \frac{\frac{K \cdot \Phi \cdot i}{R_a} \cdot (U - K \cdot \Phi \cdot \omega \cdot i) - \text{sgn}(\omega) \cdot M_{RK} - A \cdot \omega}{I}$$



Abbildung 23: Aufbau des Drehtellers

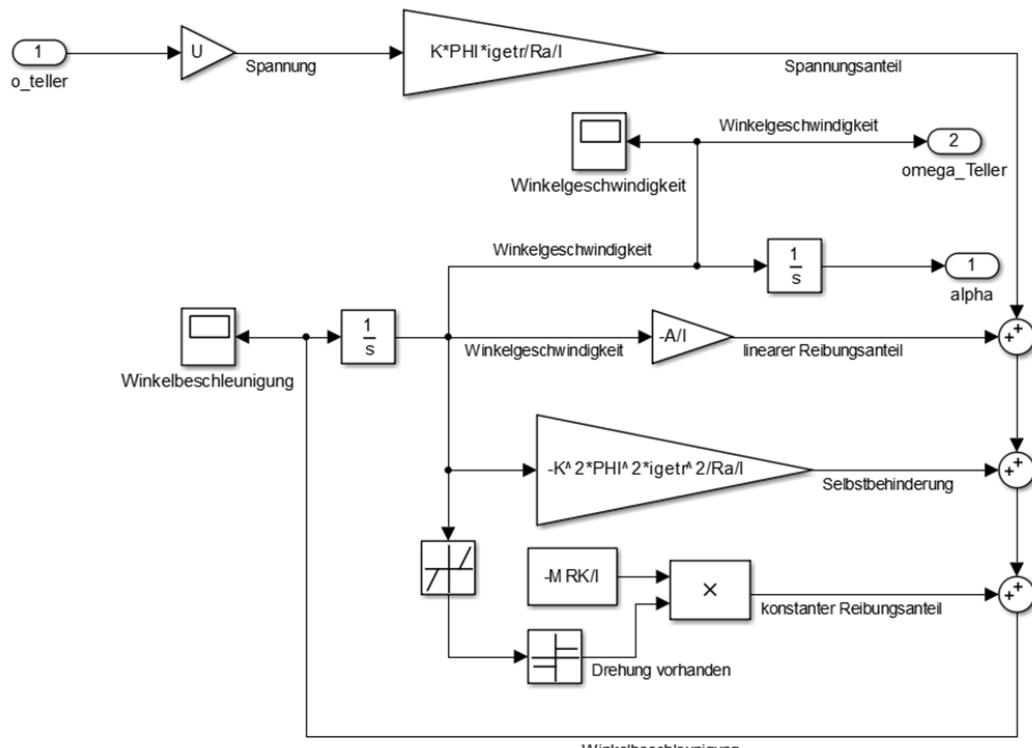


Abbildung 24: Aufbau des Drehtellers

Vor der Signumfunktion wurde ein **Totzonenglied** (nicht ein **Totzeitglied**) eingefügt.

Randbedingungen festlegen

Da sich die Anlage zu Beginn weder dreht noch irgendwelche Kräfte wirken, sind die Drehbeschleunigung, die Drehgeschwindigkeit und der Drehwinkel mit Null anzusetzen.

4.3 Aufgaben

Wichtig:

- Vor dem Erzeugen der Systeme müssen Sie verschiedene Simulationsparameter (Simulationszeit, Schrittweite, Solver, ...) einstellen.
- Lesen Sie sich bitte aufmerksam die Grundlagen zu Simulink durch.

4.3.1 Einfache Systeme

Erzeugen Sie ein Modell, das die Graphen folgender Gleichungen ausgibt:

$$\sin(2 \cdot t)$$

$$\sin^2(2 \cdot t)$$

$$\sin^2 t + \cos^2 t$$

$$\sin(\cos(2 \cdot t) + 0,1 \cdot t^2)$$

Wichtig: Nach dem Erzeugen der Systeme müssen Sie verschiedene Simulationsparameter (Simulationszeit, Schrittweite, Solver, ...) einstellen und erneut einen Simulationsversuch durchführen.

4.3.2 Subsysteme

Zähler

Erstellen Sie einen Zähler, der alle steigenden Flanken eines Eingangssignals (Eingang des getriggerten Systems) zählt. Verwenden Sie hierzu ein getriggertes Subsystem. Verwenden Sie als Triggersignal ein Rechtecksignal (Pulse Generator) unter Verwendung der oben genannten Einstellungen und lassen Sie den Zählerwert in einem Scope anzeigen.

Hinweis: Im Verzeichnis „Discrete“ gibt es einen Memory-Block. Dieser verzögert das eingegebene Signal um eine Schrittweite! Um diesen Memory-Block in ein getriggertes Subsystem einbauen zu können, muss man die Option „inherit sample time“ in den Parametern des Memory Blocks aktivieren, da ein getriggertes Subsystem nicht bei jedem Simulationsdurchlauf ausgeführt wird, der Memory-Block aber versucht, das Signal, welches nicht berechnet wurde, aufzuzeichnen.

Denken Sie daran die Einstellungen des Blockes „Pulse Generator“ zu ändern:

Amplitude: 1 || **Period (secs):** 2 || **Pulse Width (% of period):** 50

Stoppuhr

Erstellen Sie eine Stoppuhr, die das Zeitintervall misst, während das Signal eines Pulse Generators den Wert true einnimmt. Verwenden Sie hierzu ein Enabled Subsystem. Als Eingang soll wieder ein PWM-Signal dienen. Lassen Sie auch dieses Mal den Wert der Stoppuhr in einem Scope anzeigen.

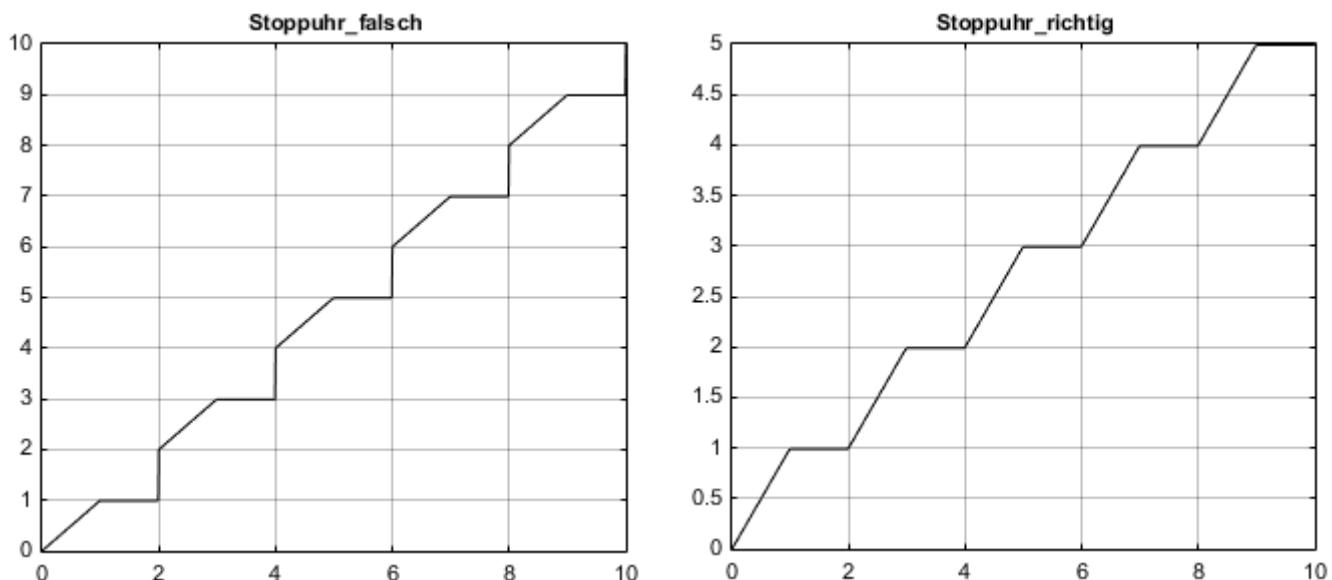


Abbildung 25: Falsche (links) und richtige Lösung (rechts) der Stoppuhr

Tipp: Die Simulationszeit ist global und läuft immer weiter und kann nicht einfach so verwendet werden (siehe „Stoppuhr_falsch“). Dies gilt es auszugleichen, um zu einem aussagekräftigen Ergebnis zu gelangen (siehe „Stoppuhr_rightig“).

4.3.3 Modellierung des Drehtellers

Erstellen Sie den Drehteller. Versuchen Sie zuerst, das Modell allein aus der Differentialgleichung zu modellieren und nicht sofort das Modell aus Abbildung 24 nachzuzeichnen. Wozu wird das Todzonenglied benötigt? Entwickeln Sie ein Parametrisierungs-m-File, dass den Parametern folgende Werte zuweist:

PHI = 0.001	Hauptfluss im Elektromotor
U = 24	[V] angelegte Spannung am Elektromotor
Ra = 40	[Ohm] Ankerwiderstand des Elektromotors
I = 0.5	[kgm ²] Massenträgheit des Tellers
K = 8	Wicklungsanzahl
MRK = 1	Konstantes Reibungsmoment
A = 5	Konstante für geschwindigkeitsabhängige Reibung
igetr = 1080	Getriebeübersetzung

Dieses m-File soll vor der Simulation **manuell** aufgerufen werden und die Werte in den globalen Workspace schreiben. Im globalen Workspace hinterlegte Variablenwerte werden von Simulink ausgelesen, wenn die Variable im Modell verwendet wird.

5 Stateflow

In diesem Teil des Versuchs werden Systeme behandelt, deren dynamischer Verlauf nicht zeit-, sondern **ereignisgesteuert** ist. Die möglichen Zustände, in denen sich das System befinden kann, sind **diskrete** Zustände, wie z. B. ein- oder ausgeschaltet. Zum **Wechseln zwischen den Zuständen** sind **Ereignisse** notwendig. Dies stellt einen großen Unterschied zu den *kontinuierlichen* Systemen da, die Sie gerade eben im Simulink-Teil kennengelernt haben.

Die auftretenden Ereignisse sind meist Eingangssignale des Systems, z. B. ein **Sensorsignal**. Es können jedoch ebenso Zustände aus einem anderen Programmteil abgefragt werden (Beispiel: Schalte Warnlampe an, wenn in der Steuerung der Zustand „Fehler“ aktiv ist.). Zur Modellierung dieser Art von Systemen stehen neben den universellen textbasierten Programmiersprachen auch graphische Modellierungstechniken zur Verfügung, z. B. Petri-Netz-Modelle oder Zustandsautomaten.

Für die Modellierung zustandsbasierter Systeme, wie sie beispielsweise auch Steuerungen darstellen, bietet sich die Verwendung von **Stateflow** an. Die Grundlagen von Stateflow werden auf den folgenden Seiten näher erläutert.

5.1 Grundlagen

In diesem Abschnitt wird die Implementierung von Zustandsautomaten mittels Stateflow beschrieben. Aufgrund des sehr großen Funktionsumfanges dieser Software wird nur auf die für dieses Praktikum unmittelbar relevanten Bestandteile eingegangen. Weiterführende Informationen sind geeigneter Fachliteratur bzw. der Dokumentation in der Hilfefunktion zu entnehmen.

Stateflow ist eine zusätzliche Bibliothek von Simulink, deshalb können Stateflow-Charts nur **innerhalb eines Simulink-Modells** ausgeführt werden. Die Stateflow-Library kann durch Anklicken des entsprechenden Symbols im Simulink-Library Browser geöffnet werden.

Die Stateflow-Library (Abbildung 26) enthält verschiedene Blöcke, für dieses Praktikum ist jedoch nur der Block „Chart“ relevant. Dieser kann durch Drag'n'Drop in ein bestehendes Simulink-Modell gezogen werden. Bei Doppelklick auf das leere Chart öffnet sich der Editor von Stateflow (Abbildung 27).

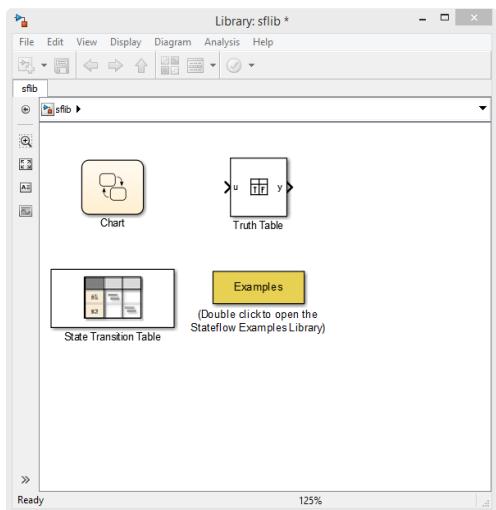


Abbildung 26: Stateflow-Bibliothek

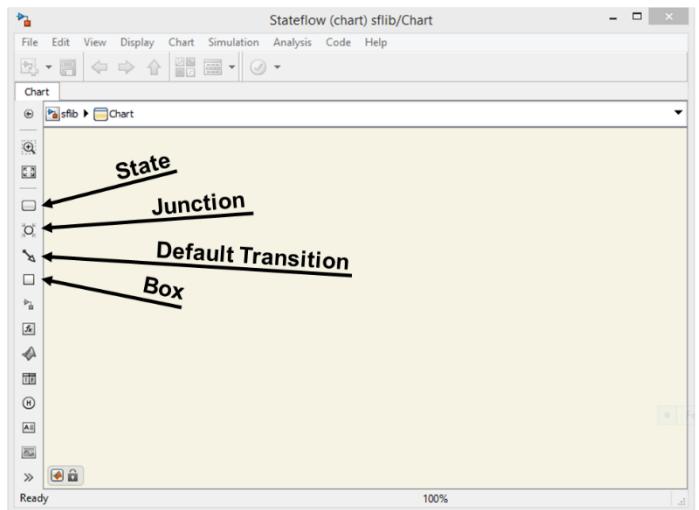


Abbildung 27: Stateflow-Chart

5.1.1 Elemente eines Zustandsautomaten

In den folgenden Abschnitten werden die in Stateflow verfügbaren Elemente beschrieben. Abbildung 28 gibt einen Überblick über den möglichen Aufbau eines Zustandsautomaten in Stateflow. Beachten Sie, dass es sich hierbei um ein Beispiel ohne Funktion handelt.

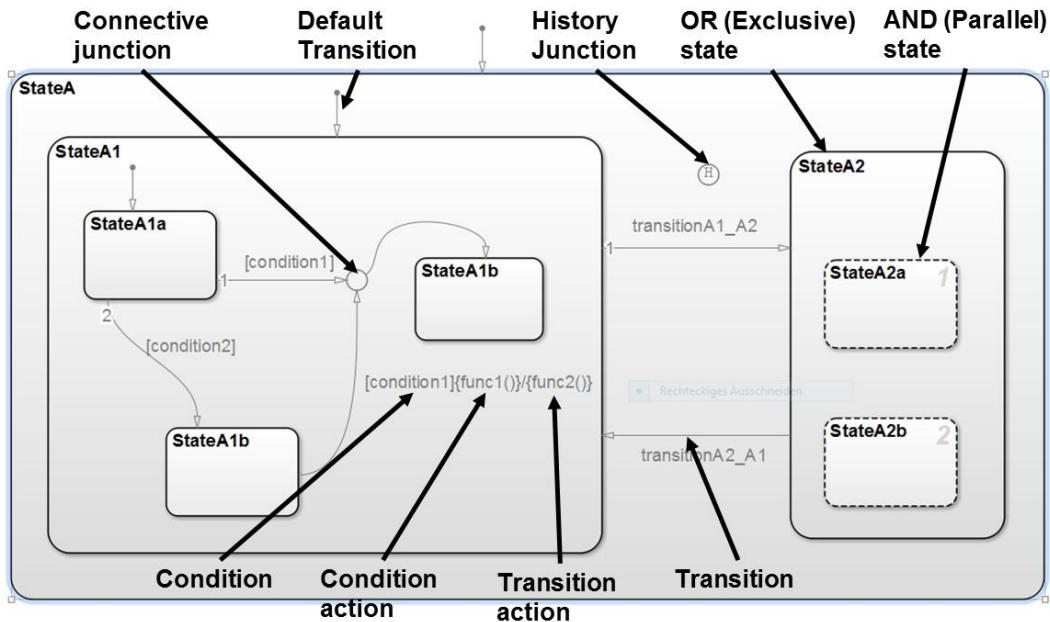


Abbildung 28: Elemente eines Charts in Stateflow

Zustände (states)

Ein „State“ stellt einen möglichen Zustand dar, in welchem sich ein dynamisches System befinden kann. Ein „Chart“ kann beliebig viele Zustände enthalten. Ein Zustand kann wiederum einen Zustandsautomaten – also weitere States – beinhalten, wodurch eine **Hierarchisierung** und **bessere Übersichtlichkeit** des Zustandsautomaten erreicht werden kann.

Ein Zustand kann durch Anklicken des entsprechenden Symbols der Werkzeugeiste im Chart platziert werden (siehe Abbildung 27). Jeder Zustand muss benannt werden. Dazu wird das Symbol „?“ mit den gewünschten Namen ersetzt. Es werden zwei Arten von Zuständen unterschieden, die im Folgenden erläutert werden (**für die grafische Darstellung siehe auch Abbildung 28**).

Zustandsarten	(OR) Exclusive states	(AND) Parallel states
Beschreibung	Nur <u>ein einziger</u> Zustand in der gleichen Hierarchieebene ist zu einem Zeitpunkt aktiv. Es <u>muss</u> pro Hierarchieebene <u>immer</u> einen aktiven Zustand geben. Der bei Beginn der Simulation aktive State <u>muss</u> mit einer Default-Transition gekennzeichnet werden. (mehr dazu auf S. 41)	<u>Alle</u> Zustände der Hierarchieebene sind <u>gleichzeitig</u> aktiv.
Darstellung	durchgezogene Rechtecke	gestrichelte Rechtecke
Einstellung	Klicken Sie mit der <u>rechten</u> Maustaste auf den markierten Zustand. Unter <i>Decomposition</i> wählen Sie die gewünschte Art von Zustand (Standardeinstellung: „OR State“). Achtung: Die Gliederung in exklusive oder parallele Zustände bezieht sich <u>auf die im gewählten State beinhalteten Zustände</u> , nicht auf den markierten Zustand!	

Tabelle 1: Parallele und exklusive Zustände



Ein komplettes Label, also die Beschriftung eines Zustands, enthält den Namen sowie die auszuführenden Aktionen. Durch **Anklicken der Beschriftung im jeweiligen State lässt sich das Label eingeben** (auch möglich mit der rechten Maustaste auf den markierten Zustand klicken und dann unter „Properties...“).

Abbildung 29 zeigt das Label eines Zustands und das zu diesem State zugehörige Properties-Fenster.

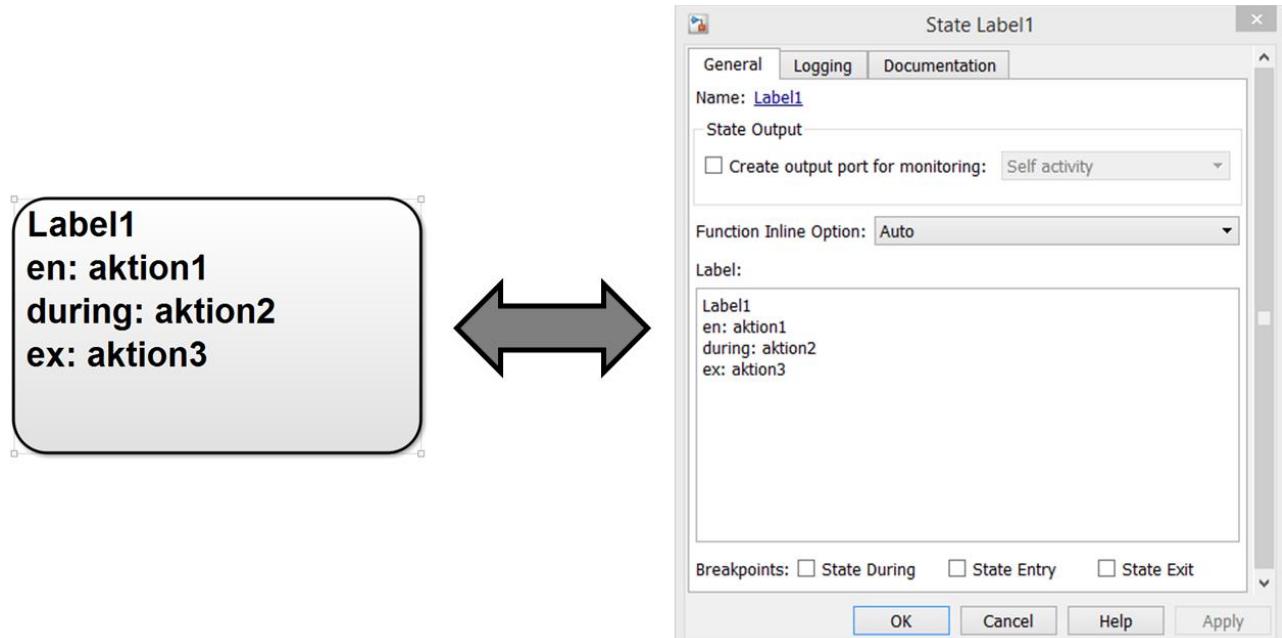


Abbildung 29: Zustand und Properties-Fenster

Hinweis: Der Name eines States muss im Modell einmalig sein und darf weder Leerzeichen noch Umlaute und Sonderzeichen enthalten. Alles andere führt zum Abbruch der Simulation. Um den Inhalt eines States dennoch klar zu bezeichnen, bietet sich ein Name mit Unterstrichen an.

Beispiel für zulässige und aussagekräftige Benennung eines States: *tuer_auf*

Die Aktionen werden in der sogenannten „Action Language“ implementiert. Wenn ein Zustand aktiv wird, werden alle im Label enthaltenen Aktionen (z. B. ändern des Wertes einer Variable) ausgeführt. Diese Aktionen werden bei der Aktivierung (entry) (Standardverhalten), während der Aktivität (during) oder beim Verlassen des Zustands (exit) ausgeführt. (weitere Informationen zur Action Language folgen im Kapitel 5.1.5 auf Seite 47).



Transitionen (transitions)

Eine Transition stellt einen **Übergang zwischen zwei States** im Stateflow-Chart dar. Eine Transition wird durch **Anklicken des Randes** der ersten States mit der linken Maustaste und **Ziehen** zum gewünschten State erzeugt. Die Transition erscheint nun als Pfeil (Abbildung 30) zwischen den beiden States und kann auch nur in **eine Richtung** durchlaufen werden. Wird die Transition in Abbildung 30 aktiv, springt der Chart von *State1* in *State2* und setzt die Variable *on* gleich 1.



Abbildung 30: Beispiel einer Transition mit komplettem Label

Über einer neu gezogenen Transition erscheint ein Fragezeichen. Dieses kann angeklickt und durch ein „Transition Label“ ersetzt werden. Eine Transition mit Label wird nur dann durchlaufen, wenn **alle im Label genannten Events und Bedingungen erfüllt** werden. Im Umkehrschluss wird eine **Transition ohne Label immer sofort aktiv**.

In Abbildung 30 ist eine Transition mit allen möglichen Typen von Labels dargestellt. Es ist selbstverständlich möglich auch nur eine Art von Label oder mehrere Labels des gleichen Typs zu verwenden. Die verschiedenen Arten von Labels werden im Folgenden kurz vorgestellt.

Event	Nur wenn das genannten Event (auch mehrere Events sind möglich) auftritt, wird die evtl. vorhandene Bedingung (Condition) auf ihre Gültigkeit geprüft. Mehrere Events können durch Verknüpfung mit logischen Operatoren kombiniert werden. Die Abarbeitung von Events erfolgt in der zeitlichen Reihenfolge des Empfangs.
Condition	Bool'scher Ausdruck, z. B. [i_Temperature>20], der zu <i>true</i> werden muss, damit die Transition ausgeführt wird. Wenn keine Bedingung im Label steht, wird immer der Wert <i>true</i> für die Auswertung angenommen.
Condition Action	Die Aktion wird ausgeführt, sobald die Bedingung und das Event der jeweiligen Transition erfüllt werden. Gibt es keine Bedingung und wird auf kein Event gewartet, wird die Transition sofort aktiv und die Condition Action sofort ausgeführt.
Transition Action	Die Aktion wird erst ausgeführt, wenn ein gültiges Ziel der Transition erkannt wird und alle zum Erreichen notwendigen Events und Bedingungen, falls angegeben, erfüllt werden. Falls eine Transition aus mehreren Segmenten besteht, muss <u>der gesamte Pfad</u> der Transition gültig sein und der State am Ende des Pfades aktiv geworden sein.

Die Deklaration von Events sowie der für die logischen Abfragen in den Conditions nötigen Variablen erfolgt im Data Dictionary. Dies wird in Kapitel 5.1.4 genauer erläutert.

Default transition

Ist eine Hierarchieebene als Exclusive (OR) definiert, stellt sich für Stateflow die Frage, **welcher** der beinhalteten States **als erstes aktiv** ist. Dieser Sachverhalt wird deutlicher bei Betrachtung von Abbildung 31. Im State „On“ gibt es zwei weitere States „Lamp1“ und „Lamp2“. Mit Hilfe des Events „switch“ lässt sich zwischen den beiden Zuständen wechseln. Hierfür muss Stateflow jedoch wissen, in welchem der beiden States es sich **am Anfang der Simulation** befindet. Diese Aufgabe übernimmt die Default transition, die grafisch als Pfeil mit einem ausgefüllten Punkt erscheint.

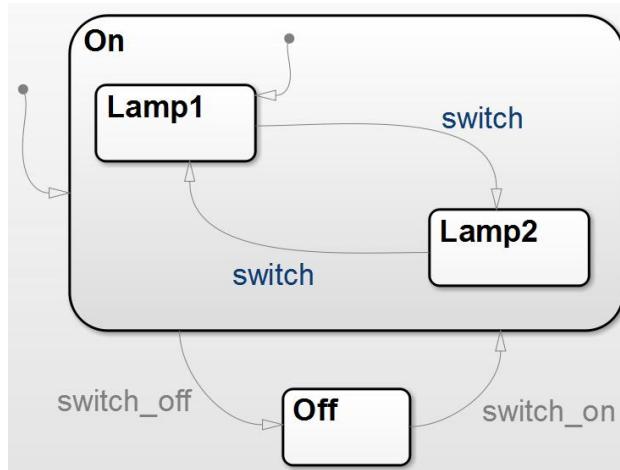


Abbildung 31: Default Transition ohne Label

Die Default transition wird immer dann aktiv, wenn ihr Superstate, also die **übergeordnete Hierarchieebene** aufgerufen wird. In Abbildung 31 gibt es zwei Default transitions. Die erste Default transition zeigt auf den State „On“ und befindet sich hiermit in der **höchsten Hierarchieebene**. Dies bedeutet sie wird nur **ein einziges Mal beim Start der Simulation** aufgerufen. Danach ist auf dieser Ebene immer entweder „On“ oder „Off“ aktiv. Dadurch ist der **Zustand dieser Ebene eindeutig definiert**. Die zweite Default transition befindet sich **innerhalb des States „On“**. Sie wirkt daher immer dann, **wenn dieser State aktiv wird**.

Hieraus ergibt sich, dass bei einer Exclusive-(OR)-Ausführung immer genau eine Default transition pro Hierarchieebene geben muss. Alles andere führt zu Fehlern und dem Abbruch der Simulation. Soll das Ziel der Default transition variabel sein, so ist zunächst eine leerer State per Default transition zu aktivieren, von dem dann die Bedingungen für den ersten „echt“ aktivierte State abgehen.

Verbindungspunkte (Junctions)

Ein Verbindungspunkt wird als ein Kreis dargestellt und kann nach Anklicken des entsprechenden Symbols der Werkzeugeiste in den Chart eingefügt werden. Er stellt eine **Entscheidungsmöglichkeit zwischen mehreren Pfaden einer Transition** bzw. eine **Zusammenführung von mehreren Pfaden einer Transition** dar. Die Transition wird nur dann gültig, wenn ein kompletter Pfad vom Quellzustand zum Zielzustand gültig ist. Zudem kann immer nur ein Pfad gültig sein.

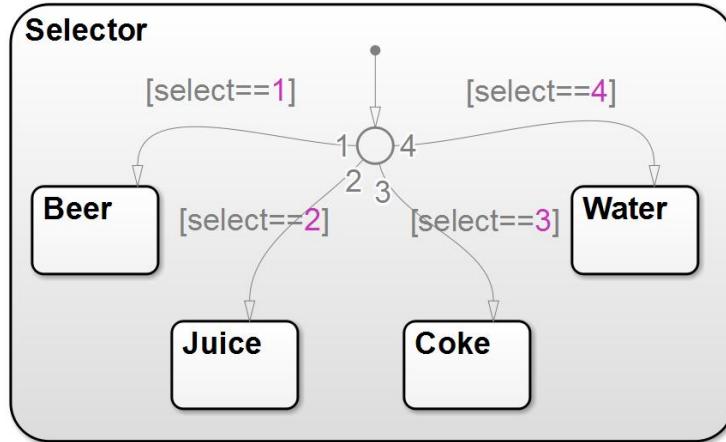


Abbildung 32: Verzweigung mit Connective Junction

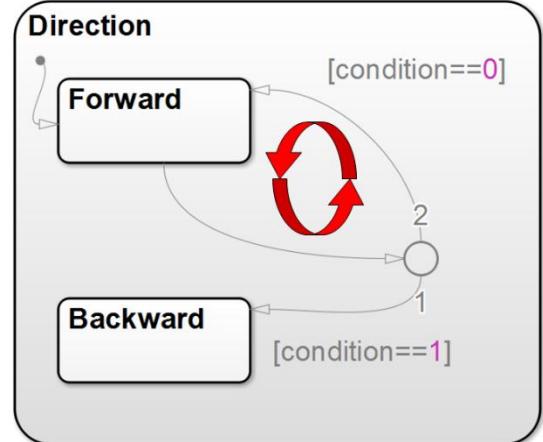


Abbildung 33: Self Loop Transition

Abbildung 32 stellt ein Beispiel dar, in dem eine Default transition zu einem Verbindungspunkt führt. Dieses Beispiel entspricht also einer aus der textbasierten Programmierung bekannten if-Abfrage.

Abbildung 33 zeigt den Fall einer „**self-loop-transition**“. Eine self-loop-transition ist eine Transition, bei der Quell- und Zielobjekt identisch sind. Dies sollte man **unter allen Umständen vermeiden**, da die Gefahr besteht, dass die so entstandene Schleife in jedem Simulationsschritt durchlaufen wird, was zu hoher Prozessorlast und damit einer **sehr deutlichen Verlangsamung der Simulation** führt.

5.1.2 Hierarchisierung und Beherrschung von Komplexität

Bei der Simulation komplexer Systeme treten eine Vielzahl verschiedener Zustände auf, die zu unübersichtlichen und damit fehleranfälligen Modellen führen. Aus diesem Grund bietet Stateflow verschiedene Möglichkeiten zur **Hierarchisierung**, wodurch sich ein großes, komplexes Modell in **kleinere und leicht zu durchschauende Teile** strukturieren lässt.

Superstates

In den vorhergehenden Beispielen wurde schon eine Art der Hierarchisierung vorgestellt, die Superstates. Ein **Superstate** ist ein Zustand, der wiederum einen Zustandsautomaten (also weitere durch Transitionen verknüpfte States) enthält. Wird ein Superstate aktiv, dann wird die entsprechende Default transition des Zustandsautomaten innerhalb des Superstates ausgeführt. Während der Aktivität des Superstates ist einer der zugehörigen Unterzustände immer aktiv. **Superstates können sowohl exklusiv als auch parallel eingeordnet werden**. Die Einführung von Superstates in einen Chart führt zu weiteren Elementen, die im Folgenden erklärt werden (siehe auch Abbildung 34).

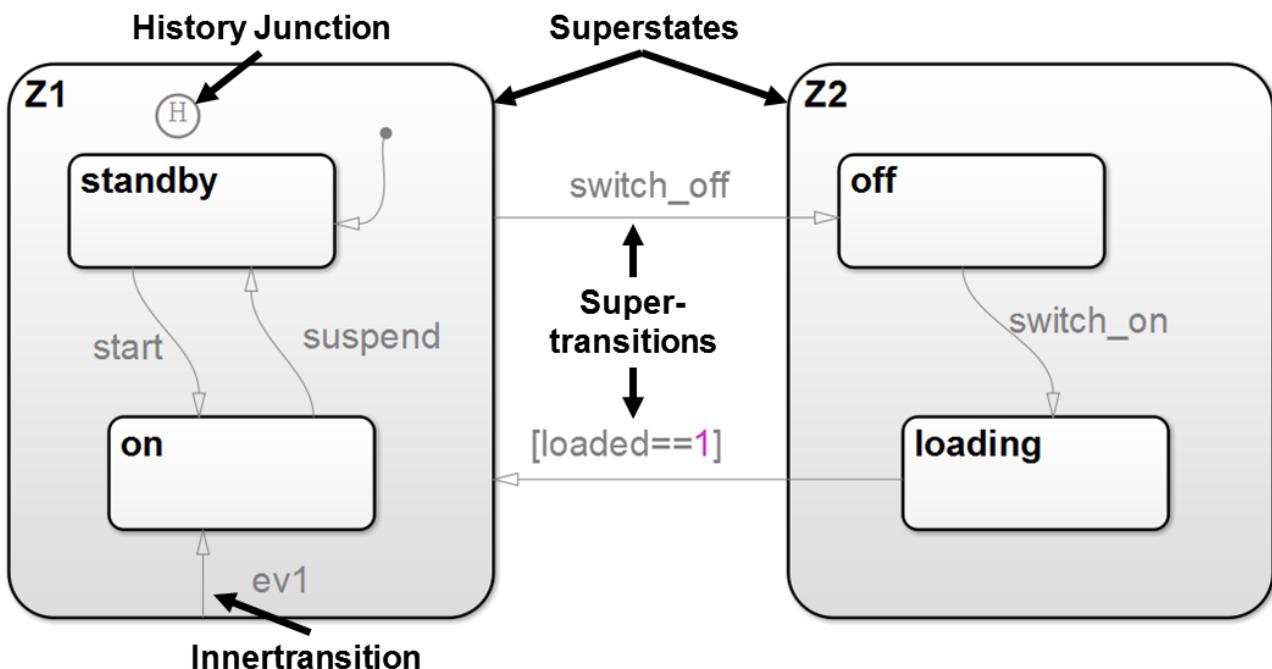


Abbildung 34: Superstates, Super- & Innertransitions und History Junction

Supertransitions: Supertransitions sind **hierarchieübergreifende Transitionen**. Das bedeutet, dass Quell- und Zielobjekt nicht zu der gleichen Hierarchieebene gehören. Dadurch kann man Superstates direkt von einem Unterzustand ohne Bedingung verlassen.

Innertransitions: Das Quellobjekt dieser Transitionen ist immer ein Superstate und das Zielobjekt ist einer der zugehörigen Unterzustände. Damit kann z. B. nach Auftritt des Events ev1 der Zustand „on“ aktiviert werden, unabhängig davon, welcher State in Superstate Z1 vorher aktiv war.

History Junctions: Sie können durch Anklicken des entsprechenden Symbols in der Werkzeugleiste erzeugt werden und müssen **innerhalb der Grenzen eines Superstates** platziert werden. Befindet sich eine History Junction in einem Superstate, wird der **letzte aktive State bei Verlassen des Superstates gespeichert**. Bei erneutem Aktivieren des States **wird das Programm an der gespeicherten Stelle fortgesetzt**. Die History Junction ersetzt also die Default transition (diese wird jedoch immer noch für die erste Ausführung des States benötigt)

Groups: In einem Chart kann der Inhalt von Superstates gruppiert werden. Dies erfolgt durch den Rechtsklick auf einen markierten Zustand. Anschließend wird „Make Contents“ → “Grouped“ gewählt. Eine Group hat ausschließlich Auswirkungen auf das graphische Verhalten der enthaltenen Elemente. Während bei nicht-gruppierten Superstates die Elemente einzeln verschoben und ihre Größe verändert werden können, können Groups und ihre Größe nur als Ganzes verschoben bzw. geändert werden. Eine Group ist im Gegensatz zu einem nicht gruppieren Superstate grau hinterlegt.

Subcharts

In einem Chart können Superstates zu Subcharts umgewandelt werden. Dies erfolgt durch den Rechtsklick auf einen markierten Zustand. Anschließend wird „Make Contents“ → “Subcharted“ gewählt. Der Inhalt des States ist nun verborgen und kann durch einen Doppelklick aufgerufen werden. Dies hat **keinerlei Auswirkungen auf den Programmablauf** und dient lediglich der **Übersichtlichkeit**.

Ein Beispiel ist in Abbildung 35 zu sehen. Hierbei wurde der Zustand „Z2“ aus Abbildung 34 auf „Subcharted“ gestellt. Diese Vorgehensweise macht vor allem Sinn bei sehr großen Modellen, die ansonsten mehrere Bildschirme einnehmen würden.

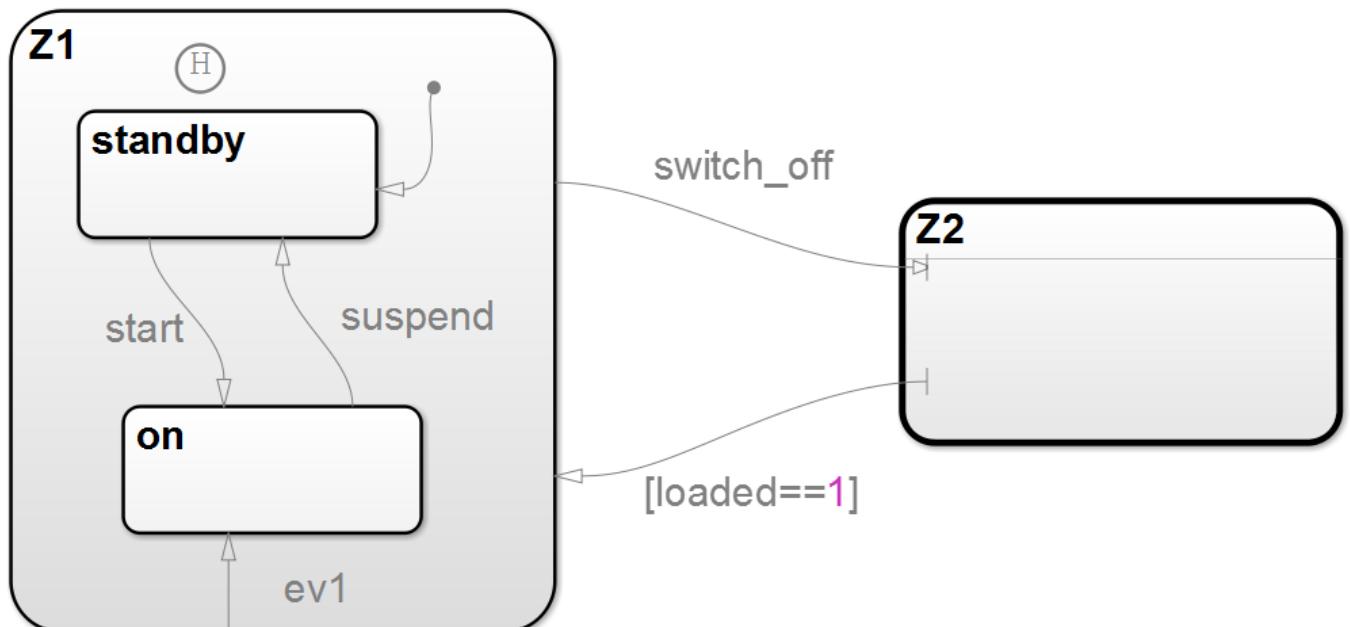


Abbildung 35: Subcharts

Boxes

Boxes verhalten sich analog zu Superstates. Sie stellen allerdings keinen eigenen Zustand dar, sondern erscheinen lediglich als frei platzierbares Rechteck ohne weitere Auswirkungen auf die Ausführung ihrer Inhalte. Auch sie lassen sich in einen Subchart verwandeln und dienen ausschließlich der Übersichtlichkeit.

Zur Erstellung einer Box wird auf das entsprechende Symbol in der Werkzeugleiste geklickt (siehe Abbildung 27 auf Seite 37) und die Box anschließend auf die richtige Größe gezogen.



5.1.3 Eigenschaften von Charts

Ein Chart wird beim Auftreten von Events aktiviert. Events können innerhalb des Charts oder auch extern erzeugt werden. Durch die Aktivierungsmethode (=Update method) wird festgelegt, wie Simulink die Charts in einem Modell steuert. Simulink bietet vier Arten von Update methods: **Triggered**, **Inherited**, **Discrete** und **Continuous**, die im Menü File → Model Properties → Chart properties entsprechend ausgewählt werden können (siehe Abbildung 36). Die Standardeinstellung ist „Inherited“.

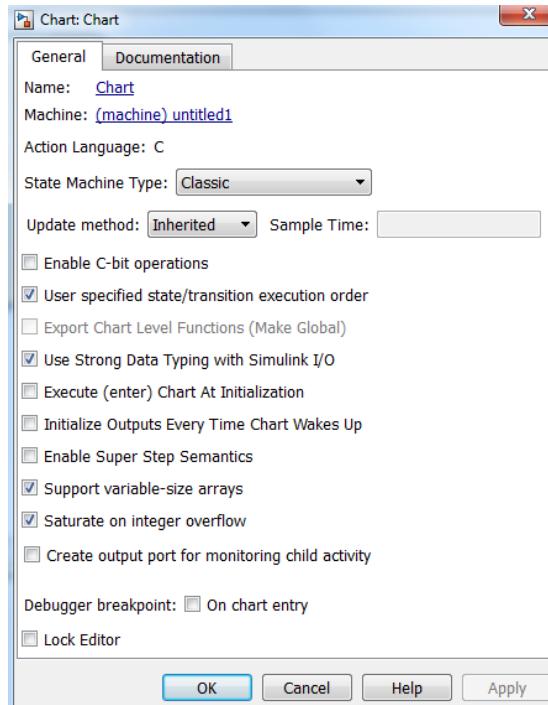


Abbildung 36: Chart Properties-Fenster

Inherited: Hier wird der Chart durch die Abstrakte seiner Eingangssignale aus Simulink gesteuert. Das heißt, jedes Mal, wenn mindestens eines der Eingangssignale aktualisiert wird, werden die Bedingungen auf den Transitionen überprüft. Die Signale müssen dementsprechend im Data Dictionary als data (Input from Simulink) deklariert werden (Kapitel 5.1.4).

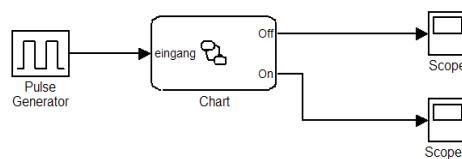


Abbildung 37: Chart mit inherited Update Method

Triggered: Ein triggered Chart wird nur dann aktiviert, wenn eine entsprechende Flanke des Trigger-Signals erkannt wird. Eine Flankenerkennung ist analog zur Erkennung eines Nulldurchgangs. Ändert sich ein Signal von Null zu einem positiven Wert, wird eine steigende Flanke erkannt. Ändert sich ein Signal von einem positiven Wert zu Null, wird eine fallende Flanke erkannt. Das Input-Trigger-Signal muss in der Data Dictionary als event deklariert werden (näheres zu Data Dictionary in Kapitel 5.1.4)

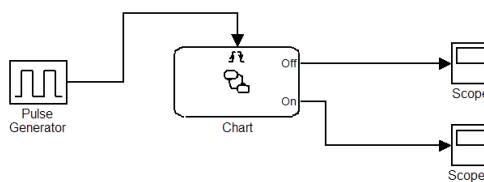


Abbildung 38: Triggered Chart



Discrete: Der Chart wird mit einer angegebenen und festen Rate (sample time) periodisch aktiviert. Diese Rate kann sich von der Integrationsschrittweite des Simulinkmodells und den Eingängen unterscheiden. Ein derartig aktviertes Chart muss über keine Ein- oder Ausgänge zu Simulink verfügen.

Continuous: Der Chart wird mit der kontinuierlichen Simulationsschrittweite des Simulink-Solvers gesteuert. Die Aktivierung erfolgt auch zu Zwischenzeitpunkten, falls diese wegen der angegebenen Genauigkeit des Modells (refine factor) benötigt werden.

5.1.4 Data Dictionary und Model Explorer

Der „**Data Dictionary**“ enthält **alle im Chart benutzten Variablen und Events** und ist **Teil des „Model Explorers“**. Der Model-Explorer von Stateflow wird durch das Menü „Modeling → Model Explorer“ (im Abschnitt „Design Data“) geöffnet. Hier lassen sich sowohl die Hierarchisierung des Charts als auch die zu jeder Hierarchieebene gehörenden Variablen und Events betrachten.

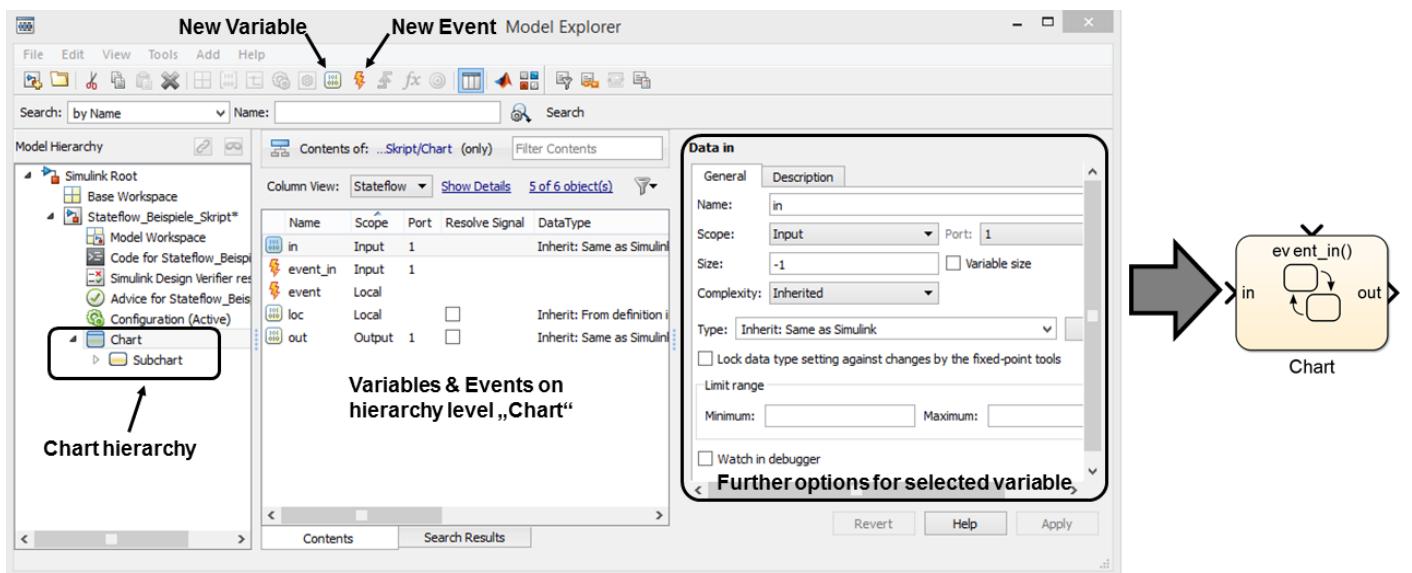


Abbildung 39: Stateflow Model-Explorer

Das **Vorgehen bei der Definition neuer Variablen und Events** ist in Abbildung 39 gezeigt und besteht aus folgenden Schritten:

1. Wähle im linken Fenster „Model Hierarchy“ die gewünschte Ebene, auf der die neue Variable definiert werden soll.
Der Zugriff auf Variablen und Events erfolgt immer nur **innerhalb der eigenen und der unterlagerten Hierarchieebenen**. Sollen Variablen und Events global **im gesamten Chart** benutzt werden, müssen sie daher auf der **obersten Ebene** definiert werden.
2. Klicke in der oberen Leiste auf die Symbole für „Add Data“ bzw. „Add Event“.
3. Benenne die Variable bzw. das Event und lege unter „Scope“ fest, ob diese als Input, Output oder Lokal definiert sein soll.

Hinweis: Lokale Events/Data können in allen Hierarchieebenen definiert werden. **Ein- und Ausgänge nach Simulink müssen immer auf der obersten Ebene definiert sein**. Wird eine Variable oder ein Event als In-/Output definiert, entsteht am Chart ein entsprechender Port (Abbildung 39).



Wird eine Variable bzw. Event im mittleren Fenster **durch Anklicken markiert**, erscheinen im rechten Fenster weitere Optionen. Weitere für dieses Praktikum relevante Einstellungen (neben den bereits genannten) sind:

- **Size:** Legt fest, ob die **Variable ein Skalar oder ein Vektor ist** (und dessen Größe). Die Elemente des Vektors werden in runden Klammern angesprochen (z. B. in (2)).
- **Type:** Legt den Datentyp fest (also Double, Integer, Bool usw.). Bei der Standardeinstellung „Inherit“ wird der Datentyp aus Simulink übernommen.

Einfacher lassen sich neue Variablen jedoch direkt im Stateflow-Modell definieren. Bei Beachtung der richtigen Syntax erkennt Stateflow automatisch neu anzulegende Variablen, wenn man die Simulation startet. Es erscheint ein Fenster, in dem die neu anzulegenden Variablen angegeben sind, per OK bestätigen legt die Variablen im Model Explorer automatisch an. Es empfiehlt sich jedoch, die so angelegten Variablen im Model Explorer nochmals zu überprüfen. Zudem sollte man sich, BEVOR man sein Modell erstellt, klar darüber sein, welche Variablen benötigt werden.

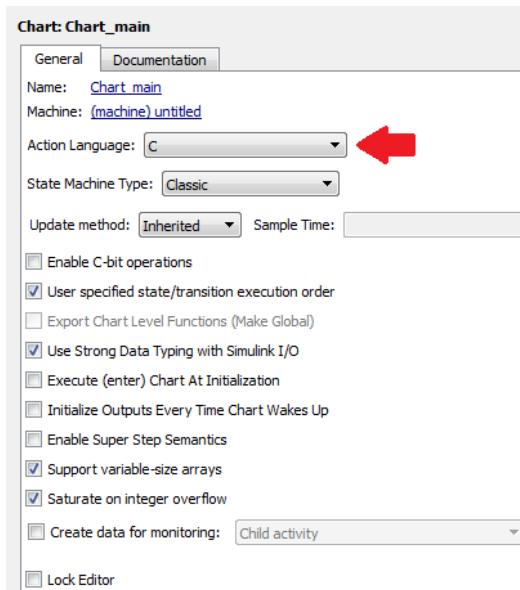
Neben diesen Grundeinstellungen bietet der Model Explorer viele weitere fortgeschrittene Funktionen, die für dieses Praktikum nicht relevant sind und daher an dieser Stelle nicht behandelt werden. Bei Interesse sei auf die Dokumentation in der Hilfefunktion verwiesen.

5.1.5 Action Language

Der Begriff „**Action Language**“ bezeichnet sämtlichen Syntax, der zur **Formulierung von Transitions- und Zustands-Labels** in Stateflow zur Verfügung steht. Dies können beispielsweise logische Ausdrücke, Aufrufe von Funktionen oder Formulierungen von Aktionen sein.

Seit Matlab R2012b unterstützt Stateflow neben der ursprünglichen Action Language C auch Matlab als Action Language. Neben der unterschiedlichen Syntax bestehen auch Unterschiede in der Funktionalität. Eine genaue Auflistung der Unterschiede sind in der Matlab Hilfe unter *Differences Between MATLAB and C as Action Language Syntax* beschrieben. Im Simulationstechnik Praktikum wird ausschließlich mit der MATLAB Action Language gearbeitet. Diese ist in jedem neu eingefügten Chart standartmäßig gesetzt.

Zu erkennen ist die eingestellte Action Language an einem Icon, welches sich links unten im Simulink / Stateflow Fenster befindet (direkt rechts neben der Toolbar am linken Rand). Möchte man sie ändern, ist dies in den Chart Properties (*Model Explorer -> Chart auswählen*) möglich.





Einen kleinen Teil der MATLAB Action Language haben wir bereits in Abbildung 30 auf Seite 40 kennengelernt. Hier wurde bei der Erfüllung einer definierten Bedingung der erste State verlassen, auf der Transition zwei Funktionen ausgeführt und schließlich bei Aufruf des zweiten States eine Variable gesetzt.

Einige (aber bei weitem nicht alle) Ausdrücke der Action Language werden im Folgenden aufgeführt.

Für dieses Praktikum relevante Ausdrücke sind grau hinterlegt. Selbstverständlich ist jedoch auch jegliche andere Syntax erlaubt. (Sofern sie das Richtige macht)

Schlüsselwörter und Elemente in Action Language

Schlüsselwort (Abkürzung)	Bedeutung
<code>entry</code>	Aktion bei <u>Eintritt</u> in den State (z. B. <code>en: variable_in = 1;</code>)
<code>during</code>	<u>Fortlaufende Aktion</u> während Simulation sich im State befindet.
<code>exit</code>	Aktion bei <u>Verlassen</u> des States (z. B. <code>ex: variable_out = 1;</code>)
<code>on event_name</code>	Aktion, wenn das Event „event_name“ auftritt.
<code>change(data_name)</code>	Erzeugen eines lokalen Events, wenn sich der Wert von <code>data_name</code> ändert bzw. bei der Aktivierung/Deaktivierung des Zustands <code>state_name</code>
<code>entry(state_name)</code>	Sendet das Event <code>event_name</code> an den Zustand <code>state_name</code> . Das Event kann auf der vom State wegführenden Transition abgefragt werden.
<code>exit(state_name)</code>	
<code>send(event_name,state_name)</code>	Sendet das Event <code>event_name</code> an den Zustand <code>state_name</code> . Das Event kann auf der vom State wegführenden Transition abgefragt werden.

Tabelle 2: Schlüsselwörter der Action Language

Logische und mathematische Operatoren

Eignen sich vor allem für WENN-DANN-Abfragen mittels Transitionen. (z. B. WENN Temperatur > 100 °C, DANN springe in State „Wasser kocht“). **Verschachtelungen** sind mittels **runden Klammern ()** möglich.

Operator	Beschreibung
<code>a * b</code>	Multiplikation
<code>a / b</code>	Division
<code>a %% b</code>	Restwert Division (Modulus), besser ist hier aber „ <code>ml.mod</code> “
<code>a + b</code>	Addition
<code>a - b</code>	Subtraktion
<code>a > b</code>	Größer-Vergleich
<code>a < b</code>	Kleiner-Vergleich
<code>a >= b</code>	Größer-Gleich-Vergleich
<code>a <= b</code>	Kleiner-Gleich-Vergleich
<code>a == b</code>	Gleichheit
<code>a ~= b</code>	Ungleichheit

Tabelle 3: Logische und mathematische Operatoren

Aufruf von MATLAB-Funktionen und Zugriff auf den Workspace

Aus Stateflow lässt sich auch auf im MATLAB-Workspace definierte Variablen zugreifen. Um sie benutzen zu können müssen sie als Variable des Typs „Parameter“ im Modelexplorer definiert werden. Stateflow kann dann die entsprechenden Variablen aus dem Workspace benutzen.

Des Weiteren können auch MATLAB-Funktionen ausgeführt werden. Hierzu einfach die normale Matlab-Syntax verwenden, z.B. `[mod(a,3)==1]`. (hätte man C als Action Language definiert, so sieht der Funktionsaufruf wie folgt aus `ml.funktion`, dies ist im Praktikum aber nicht notwendig).

Wie immer an dieser Stelle auch der Hinweis auf die Hilfefunktion, in der viele weitere Befehle und Einsatzmöglichkeiten zu finden sind.

5.1.6 Variablen und Events

Folgende grundsätzliche Dinge müssen beachtet werden:

- **Sensorsignale** können nur **abgefragt** und **Aktorsignale** nur **gesetzt** werden.
- Eine Komponente muss den Zustand der anderen Komponenten berücksichtigen. So darf z. B. ein Auto erst dann losfahren, wenn die Ampel grün wird. Dies wird in Stateflow mittels Bedingungen auf Transitionen abgefragt.

Namenskonvention für den Aufgabenteil dieses Praktikums

Namen von **Sensorsignalen** in der Form `(i_...)`. **Aktorsignale** in der Form `(o_...)`. Dies dient der Übersichtlichkeit und ist Voraussetzung für eine Abnahme der Aufgaben.

Beispiel 1:

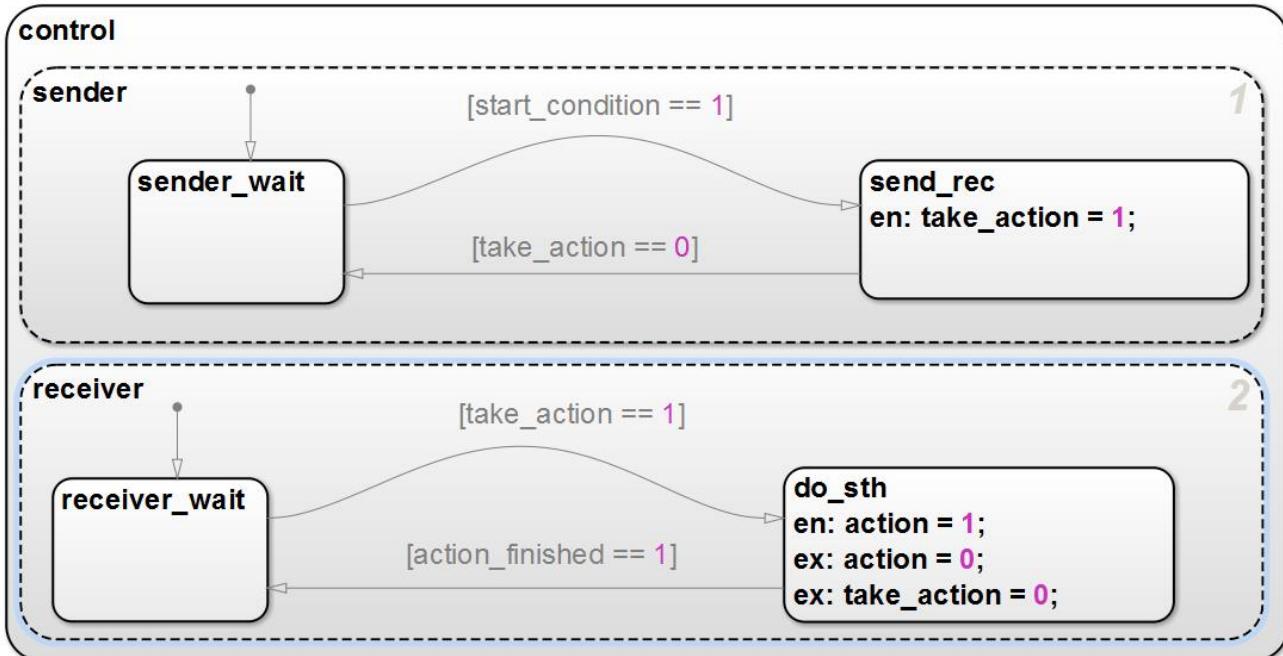


Abbildung 40: Beispiel zur Verwendung einer Kommunikationsvariablen

Der Empfänger (im State „empfaenger_befehl1“) ist so lange im Wartezustand, bis er durch den Sender über die Variable „take_action“ den Auftrag zur Ausführung einer Aktion bekommt. In diesem Beispiel ist das die Aktivierung von „action“. Über die Variable „action_finished“ wird dem Empfänger mitgeteilt, wann seine Aktion beendet ist. Als **Exit-Bedingung** müssen dann sowohl „action“ als auch „take_action“ zurückgesetzt werden. Sobald „take_action“ zurückgesetzt ist, weiß der Sender, dass der Empfänger seine Funktion ordnungsgemäß ausgeführt hat und kehrt in den Ausgangszustand zurück.

Wird diese Methode verwendet benötigt man **eine Kommunikationsvariable pro Komponente**. Andernfalls ist die Gefahr zu groß, dass aus Versehen mehrere Komponenten gleichzeitig aktiviert werden.

Im Rahmen dieses Praktikums ist folgende Regel zulässig:

Die Variable wird **immer im Sender gesetzt** und vom **Empfänger immer nach Beendigung der Aktion zurückgesetzt**.

Dies ist jedoch alles andere als elegantes Programmieren, für eine reine Simulation auf einem einzigen PC jedoch zulässig. In einer „echten“ Anlage wären die Steuersignale in der Regel jedoch Schaltsignale, die per SPS gesetzt werden. Es liegt als ein high oder low in einem Stromkreis an. Dies kann allerdings keinesfalls von einem empfangenden Gerät geändert werden. Eleganter wäre es also, eine sogenannte Hand-Shake-Variable einzuführen, durch welche der Empfänger den Empfang des Signals quittiert. Liegt dort ein high an, so löscht der Sender seine Variable, der Empfänger wartet auf die falling edge und ist danach wieder empfangsbereit.

Eine andere Möglichkeit der Kommunikation zwischen Sender und Empfänger sind Events. In diesem Fall entfällt die Notwendigkeit des Zurücksetzens der Kommunikationsvariablen (im letzten Beispiel war das „befehl1“).

Beispiel 2:

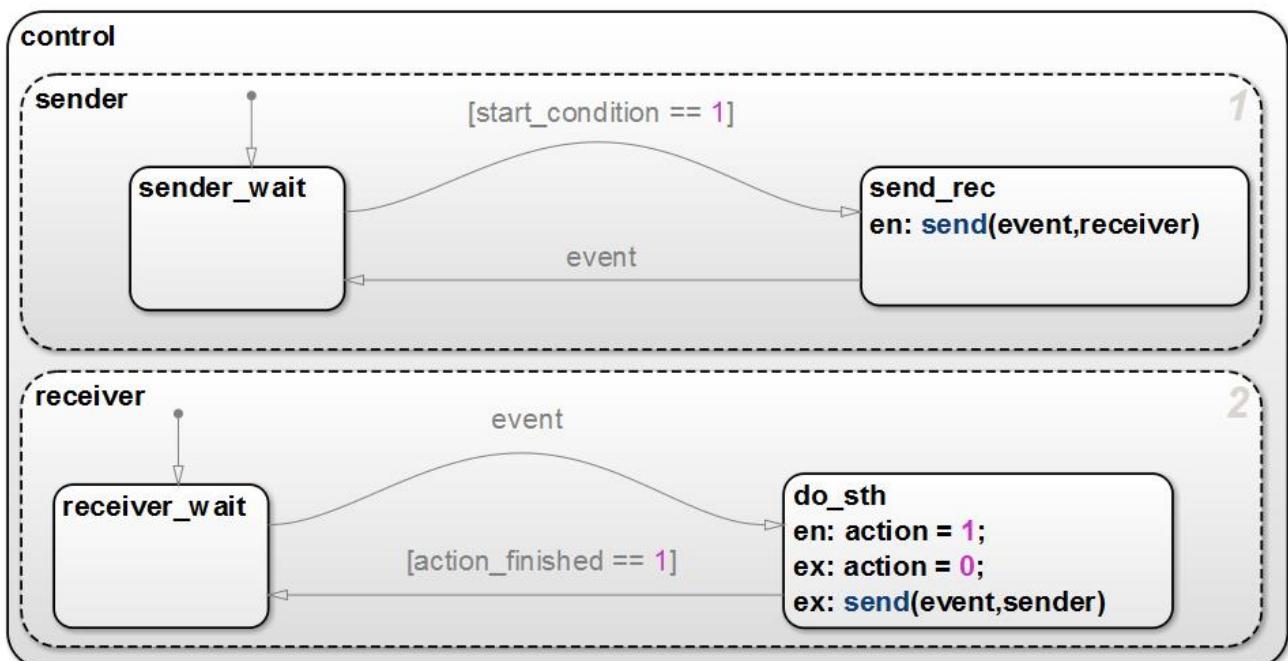


Abbildung 41: Beispiel zur Verwendung von Events

Hier **sendet der Sender das Event „event“ an den Empfänger**. Nachdem dieser seine Aktion beendet hat, teilt er dies dem Empfänger über das Event mit. Der Vorteil dieser Methode ist, dass das **Event gezielt an einen einzigen Empfänger** gesendet werden kann. Man muss sich also keine Gedanken über die versehentliche Aktivierung anderer Komponenten machen. **Zudem kann ein Event für mehrere Komponenten verwendet werden.**

ABER: Events werden nicht zwischengespeichert (asynchrone Kommunikation). Ist der Empfänger also noch in einem anderen State, in dem das Event nicht abgefragt wird, bekommt er von einem gesendeten Event nichts mit.

Beispiel 3:

Folgende Abbildung zeigt das **Grundprinzip, wie mechanische Aktoren bedient werden**. Nachdem der Aktor (hier Schieber) in Bewegung gesetzt wurde, wird auf den Endschalter (Sensorsignal „extended“) gewartet. Ist dieser erreicht, wird der Aktor wieder eingefahren und nach Beendigung des Vorgangs ein Event an den Sender geschickt, damit dieser weiß, dass die Aktion erfolgreich beendet wurde.

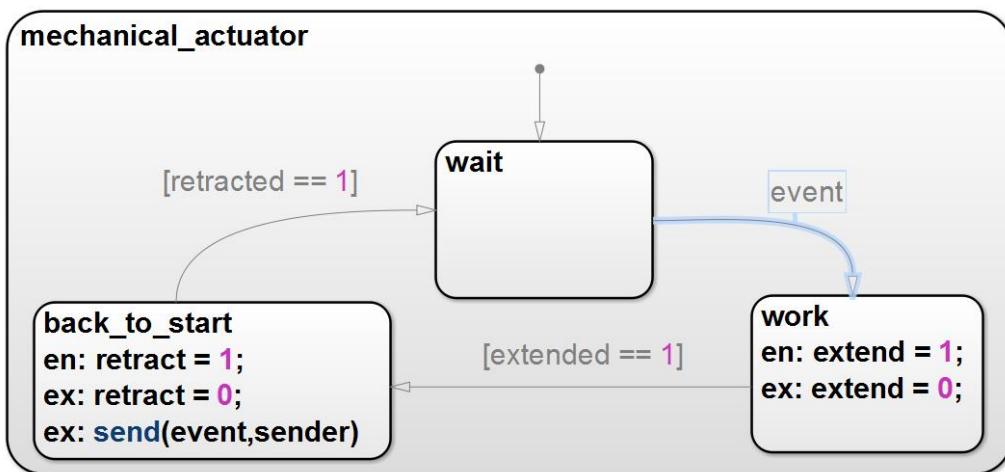


Abbildung 42: Steuerung eines Aktors



5.1.7 Beispiel zur allgemeinen Vorgehensweise

Eine zu simulierende Heizung hat **zwei Zustände** und soll über einen **Temperatursensor** gesteuert werden. Unterhalb von 16°C soll sich die Heizung einschalten, oberhalb von 20°C aus. Wie könnte die Temperatursteuerung für diese Heizung aussehen?

Dies ist nur ein Beispiel zum Verständnis, keine Praktikumsaufgabe!

1. Chart in das existierende Simulink-Modell der Heizung einfügen und mittels Doppelklick öffnen.
2. Überlegen, **welche Variablen benötigt werden und diese im Data Dictionary anlegen** (siehe Kapitel 5.1.4 auf Seite 46). In diesem Beispiel ist das die *Temperatur*, die über einen *externen Sensor* geliefert wird, sowie das eigentliche *Steuersignal für die Heizung (an-aus)*. Folglich ist hier eine Variable, die als Eingang deklariert werden muss, nötig (Temperatur), sowie ein Ausgang (Steuersignal).
3. Überlegen, **welche Zustände das System einnehmen kann** und diese Zustände (States) einfügen. Anschließend eindeutige und sinnvolle Namen für die Zustände vergeben. Hier gibt es nur zwei mögliche Zustände: „Heizung an“ und „Heizung aus“.
4. Überlegen, unter **welchen Bedingungen von einem Zustand in einen anderen Zustand gewechselt wird**. In diesem Beispiel wird zu „Heizung_an“ gewechselt, wenn die Temperatur kleiner als 16°C ist und zum Zustand „Heizung_aus“, wenn die Temperatur oberhalb von 20°C ist.
5. Transitionen mit den in Schritt 4 überlegten Bedingungen anlegen. Ebenso überlegen, **in welchem Zustand sich die Steuerung am Anfang befinden soll** und entsprechend die **Default transition anlegen**. In diesem Beispiel ist es egal, in welchem Zustand die Simulation beginnt, da es keinen logischen Startzustand gibt.
6. **Anweisungen in den Zuständen anlegen** (die erste Zeile eines Zustandes ist dessen Name, alle folgenden Zeilen sind Anweisungszeilen (Zuweisungen)). Im Zustand „Heizung_an“ wird die Heizung angeschaltet (en: heizung=1), im Zustand „Heizung_aus“ wird die Heizung abgeschaltet (en: Heizung=0).

7. **Testen!!!** (z. B. durch Betrachtung der Ein- und Ausgangssignale in einem Scope)

5.1.8 Programmbeispiel

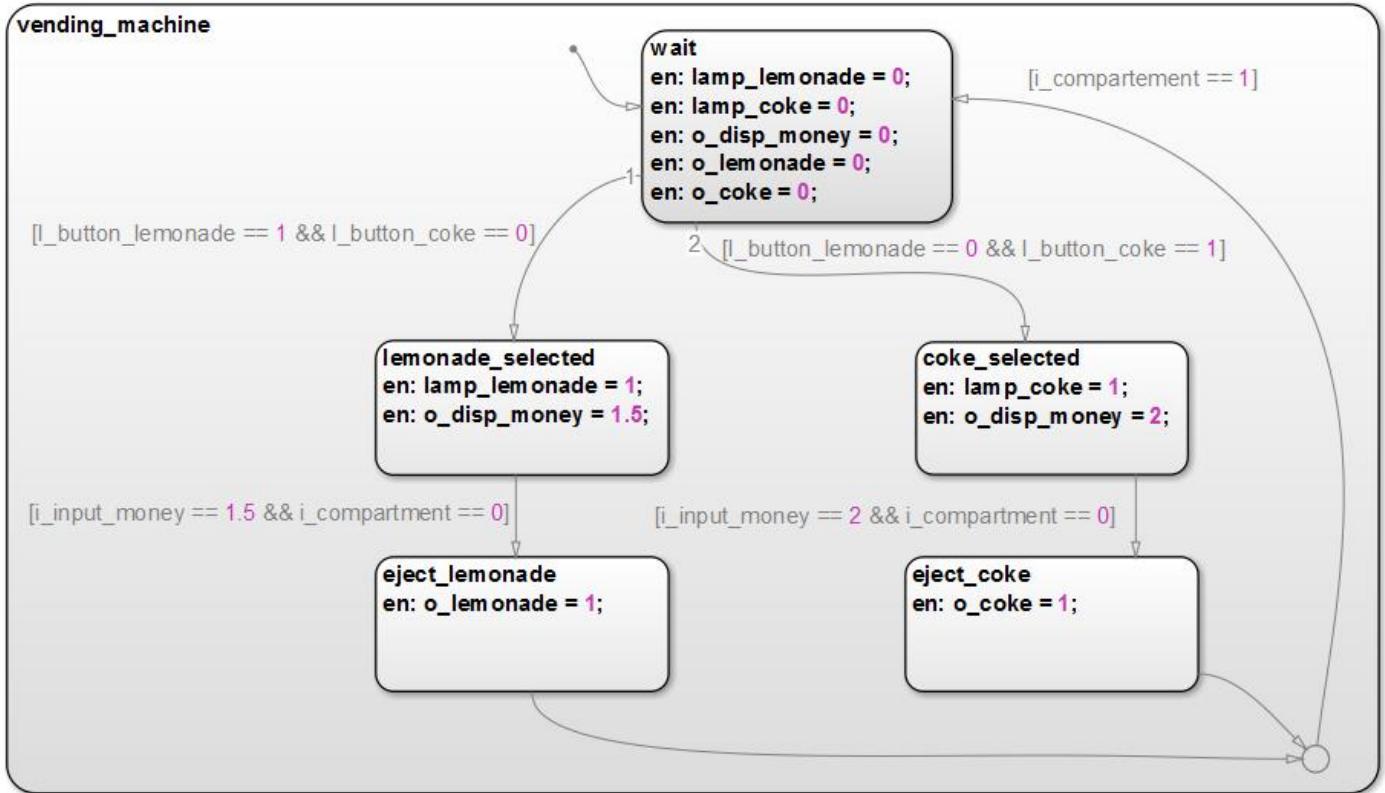


Abbildung 43: Programmbeispiel Getränkeautomat

Abbildung 43 zeigt die Programmierung der Steuerung eines (sehr rudimentären) Getränkeautomaten. Die Steuerung läuft in folgenden Schritten ab.

Dies ist nur ein Beispiel zum Verständnis, keine Praktikumsaufgabe!

1. Im ersten Schritt wird der Ausgangszustand hergestellt: Alle Ausgänge werden auf einen definierten Zustand gesetzt.
2. Anschließend wartet der Automat auf eine Tasteneingabe für Cola oder Limo. Hierzu werden laufend die Bedingungen auf den Transitionen geprüft.
3. Entsprechend der Eingabe des Benutzers (simuliert durch die Sensorsignale der Wahlstellen) wird eine Lampe angeschaltet und der passende Geldbetrag auf dem Display angezeigt.
4. Wenn der Kunde den Betrag bezahlt hat und sich im Auswurf keine Flasche befindet, wird das passende Fach geöffnet.
5. Ist die Flasche ausgegeben (erkennbar durch den Flaschensensor im Ausgabefach), kehrt der Automat in seinen Ausgangszustand zurück und der Programmzyklus beginnt von neuem.

5.1.9 Checkliste: Häufige Fehler

1. Keine bzw. mehrere gültige Default transitions innerhalb einer Hierarchieebene.
2. **Transition kreuzt Grenze des Superstates.** Hierdurch gilt der State als Verlassen und es kommt zu unvorhersehbarem Verhalten.

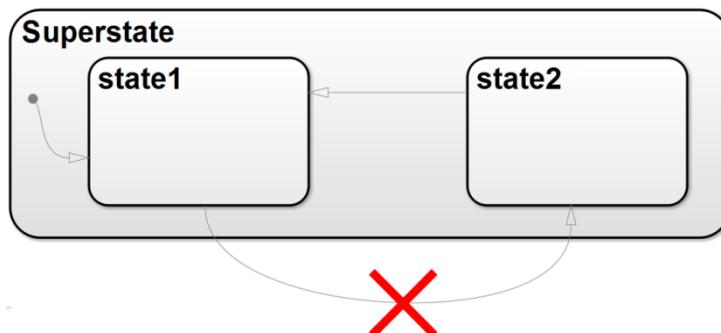


Abbildung 44: Transition verlässt State

3. **Sonderzeichen, Umlaute oder Leerzeichen** in jeglichen Benennungen (States, Variablen, Events).
4. **Sackgassen**, die zu einem Programmstillstand führen. In untenstehendem Beispiel kann das Wasser nie wieder abgeschaltet werden. Programme sollten daher immer als Kreisprozess angeordnet sein.



Abbildung 45: Sackgasse

5. **Mehrere, gleichzeitig erfüllte Transitionen** ausgehend von einem State. Stateflow entscheidet sich dann für den zuerst gesetzten Pfeil, dies ist durch die Nummerierung am Ausgang des States gekennzeichnet. Grundsätzlich ist dies zu vermeiden, da es zu ungewolltem Systemverhalten führen kann.

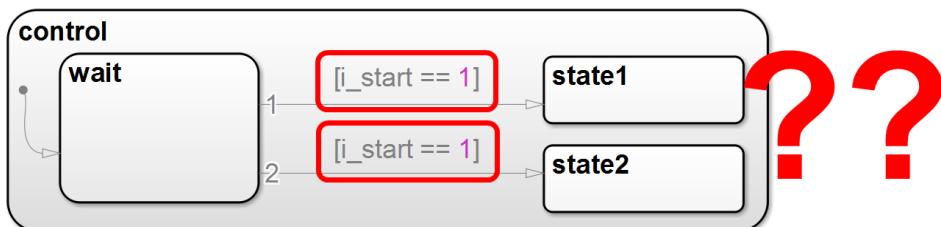


Abbildung 46: Zwei gleichzeitig gültige Transitionen

6. **Vergleich von Double Werten mit „==“.** Auf Grund numerischer Ungenauigkeiten wird diese Bedingung nie erfüllt. Versuchen Sie das Problem so zu formulieren, dass „>=“ oder „<=“ benutzt werden kann.
7. **Variablen werden nicht zurückgesetzt.** Lassen Sie Variablen nicht länger als notwendig gesetzt.
8. **Mehrere Zustände mit gleichen Namen.** Jeder State muss einen im Chart einmaligen Namen haben.
9. **[1 < x < 10] funktioniert nicht.** Verwenden Sie `[x > 1 && x < 10]`.
10. **Rechnen mit Integer-Werten.** Bei Verwendung von Integer-Werten ergibt sich: $8/5 = 1$! Für ein korrektes Ergebnis müssen 8 und 5 als Double definiert werden.

5.2 Aufgaben

5.2.1 Wichtige Hinweise

Einige Lösungen der folgenden Aufgaben werden an den folgenden Praktikumstagen noch benötigt. Lösen Sie daher **jede Aufgabe in einem neuen Modell und speichern Sie ihre Lösungen auf dem Laufwerk Z:**

Beachten Sie die **Namenskonvention** auf Seite 49 sowie die **häufigen Fehler** auf der vorherigen Seite.

Eine Aufgabe gilt erst dann als gelöst, wenn Sie die Funktionsfähigkeit ihres Programmes **NACHWEISEN** und **ERKLÄREN** können. Benennen Sie hierzu alle relevanten Signale und Blöcke. Betrachten Sie die Ein- und Ausgangssignale in EINEM Scope, in dem alle Signale benannt sind. Überlegen Sie sich sinnvolle Testsignale, um die Funktionalität zu verifizieren.

5.2.2 Modellierung einer einfachen Temperaturregelung

Die Temperatur T eines Gewächshauses soll automatisch geregelt werden. Hierfür werden bei **einer Temperatur von $T > 25^{\circ}\text{C}$ die Fenster geöffnet**. Fällt die **Temperatur unter 20°C** werden die Fenster wieder geschlossen. Die Fenster haben genau zwei diskrete Zustände (auf-zu).

Aufgaben:

- Beachten Sie die Hinweise und die allgemeine Vorgehensweise auf den vorigen Seiten!
- a. Öffnen Sie einen leeren Stateflow-Chart und definieren Sie im *Data Dictionary* eine **Eingangsvariable** "*i_Temperatur*" sowie eine **Ausgangsvariable** "*o_Fenster_auf*". *o_Fenster_auf*=1 bedeutet, dass das Fenster offen ist, bei *o_Fenster_auf*=0 ist es geschlossen.
- b. Modellieren Sie das gewünschte Verhalten des Fensters in Abhängigkeit der Temperatur innerhalb des States "Fenster".
- c. Benutzen Sie den **Signal-Builder** und ein **Scope** um die Funktion Ihrer Steuerung zu überprüfen. Die Funktionsweise dieser Blöcke entnehmen Sie Kapitel 4.1.1

Zusätzlich soll bei einer Temperatur von $T>30^{\circ}\text{C}$ eine Lüftung aktiviert werden. Für $T<25^{\circ}\text{C}$ wird diese wieder ausgeschaltet.

Aufgabe:

- a. Definieren Sie das zusätzlich notwendige **Ausgangssignal** "*o_Luefter_an*".
- b. Modellieren die Funktion des Lüfters innerhalb des States "Luefter".
- c. Der Lüfter arbeitet **unabhängig vom Fenster**. Die beiden States "Fenster" und "Luefter" müssen daher **parallel** ausgeführt werden.
Dies wird folgendermaßen eingestellt: Rechtsklick außerhalb der States "Fenster" und "Luefter"
→ Decomposition → AND (Parallel)
- d. Erweitern Sie ihr Scope aus der letzten Aufgabe um einen zusätzlichen Eingang und **überprüfen Sie erneut die korrekte Funktion**.

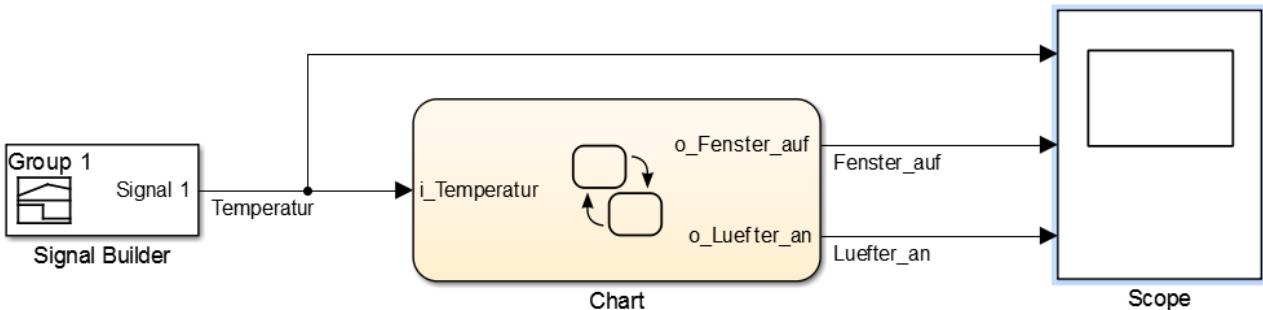


Abbildung 47: Lösung „Temperaturregelung“ auf Ebene Simulink

5.2.3 Timer

In dieser Aufgabe soll ein einfacher Timer erstellt werden, die Implementierung erfolgt wieder in Stateflow. Die **Eingangssignale** sind die **Wartezeit (wait_time)** und die **Simulationszeit** (Verwenden Sie den Simulink-Block „Clock“).

Das Programm soll folgende Funktion erfüllen:

Zu Beginn hat der Ausgang den Wert *null*. Sobald der Eingang *wait_time* einen Wert annimmt, wird die entsprechende Zeit gewartet und darauf der Ausgang für nur **einen Simulationsschritt auf eins gesetzt**. Ansonsten steht der Ausgang immer auf *null*. Wenn die Wartezeit abgelaufen ist und das Eingangssignal (*wait_time*) immer noch einen Wert hat, startet der Timer sofort wieder.

Hinweis: Verwenden Sie als Eingang für die Variable *wait_time* den Block „Pulse Generator“. Im untenstehenden Beispiel wird vom „Pulse Generator“ eine periodische *wait_time* von 1 alle 3 Sekunden mit einer Pulsweite von 50% vorgegeben. Zudem ist der Verlauf der Simulationszeit „Clock“ dargestellt.

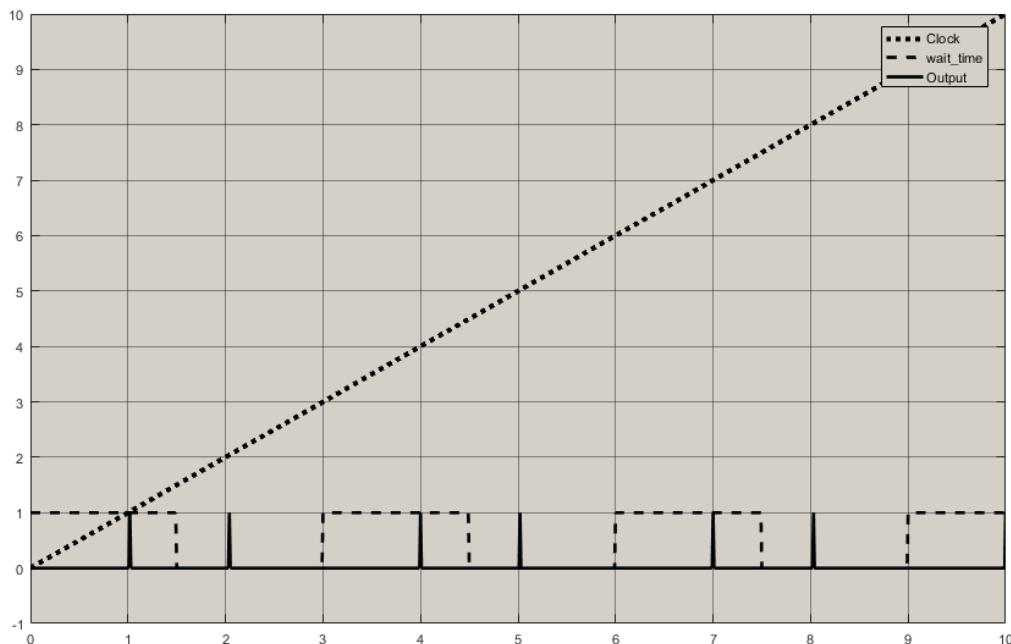


Abbildung 48: Scope mit Simulationszeit, *wait_time* und der Ausgabe des Timers

5.2.4 Winkelsensor

Sie haben heute bereits ein Modell für einen Drehteller erstellt. In dieser Aufgabe soll nun mit der Hilfe von Stateflow erkannt werden, **wann sich dieser Drehteller um 90° gedreht hat**.

Nehmen Sie als **Eingangssignal** für Ihren Chart den **Drehwinkel für eine Winkelgeschwindigkeit von $\dot{\alpha} = 45 \text{ }^{\circ}\text{s}^{-1}$** .

Das **Ausgangssignal „*i_teller_ausgerichtet*“** ist so lange **eins**, wie sich der Teller im Bereich von 90° ($\pm 2^\circ$) befindet. Der Sensor ist nur in diesem Bereich vollständig abgedeckt.

Abbildung 49 auf der nächsten Seite zeigt den Bereich, in dem der Sensor detektiert (blau). Die Flasche (kleiner Kreis) auf dem Drehteller deckt den Sensor gerade so vollständig ab.

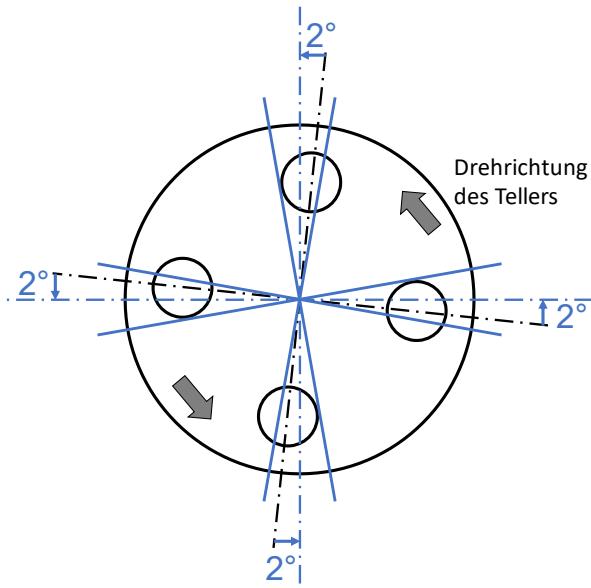


Abbildung 49: Funktionsweise Drehteller

Hinweise:

- Der Drehteller ist zu Beginn nicht zwangsläufig ausgerichtet, was bedeutet das **andere Startwerte als $\alpha = 0$ möglich sind**. Ihr Chart muss hiermit umgehen können. Dies lässt sich z.B. durch die **Verwendung der mathematischen Funktion "modulo"** (Division mit Rest) realisieren.
- Verwenden Sie für die **Rechnung mit Modulo in Stateflow** den Befehl „***mod(x,y)***“. Weitere Informationen hierzu finden Sie nach Eingabe von „**help mod**“ im MATLAB Command-Window.
- Beachten Sie, dass der Drehteller **alle 90°** ausgerichtet ist. Ihr Chart muss also auch für 180°, 270°, 360° usw. „Ausgerichtet“ melden.
- Die **Überprüfung der korrekten Funktion** erfolgt wieder mit einem **Scope**.

5.2.5 Vereinfachte Drehtellersteuerung

In dieser Aufgabe ist eine einfache Steuerung für den Drehteller zu implementieren. Der Drehteller steht als Simulink-Modell aus Aufgabe 4.3.3 (Seite 36) zur Verfügung.

Die Aufgabe der Steuerung ist es, den **Motor ein- bzw. auszuschalten**. Dies erfolgt über die Variable „***o_teller***“. Der Drehteller beginnt die Drehung nach **zwei Sekunden Wartezeit** und stoppt, wenn er sich um **90°** gedreht hat, d.h. wenn der Winkelsensor aus Aufgabe 5.2.4 das entsprechende Signal gibt (***i_teller_ausgerichtet == 1***).

Implementieren Sie die Steuerung unter der Beachtung folgender Punkte:

- Der Drehteller ist **am Anfang in Ruhe**, d.h. der Motor ist aus.
- Nach 2 Sekunden beginnt die Drehung**
- Jedes Mal, **wenn der Teller ausgerichtet** ist, wird der **Motor ausgeschaltet** und wieder 2 Sekunden gewartet.

Anmerkung: Warten Sie etwa eine halbe Sekunde, bevor Sie das Signal ***i_teller_ausgerichtet*** abfragen. Wenn Sie sofort nach dem Einschalten des Motors den Sensor abfragen, befindet sich der Teller noch innerhalb des 2° Toleranzbereiches und der Drehteller wird sofort wieder angehalten.

Erstellen Sie ein neues Modell in Simulink und fügen Sie die folgenden Komponenten aus den schon implementierten Modellen ein:

- Drehteller (Aufgabe 4.3.3)
- Winkelsensor (Aufgabe 5.2.4)
- Timer (Aufgabe 5.2.3)

→ Sie können die benötigten Teile direkt in Ihre Steuerung kopieren.

Abbildung 50 zeigt die logische Anordnung der Subsysteme, sowie den Signalfluss.

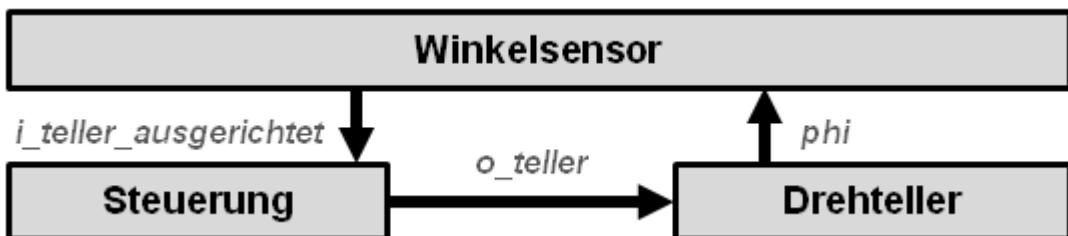


Abbildung 50: Ein- und Ausgänge der Komponenten

In Abbildung 51 ist eine mögliche Anordnung der Komponenten in Simulink zu sehen.

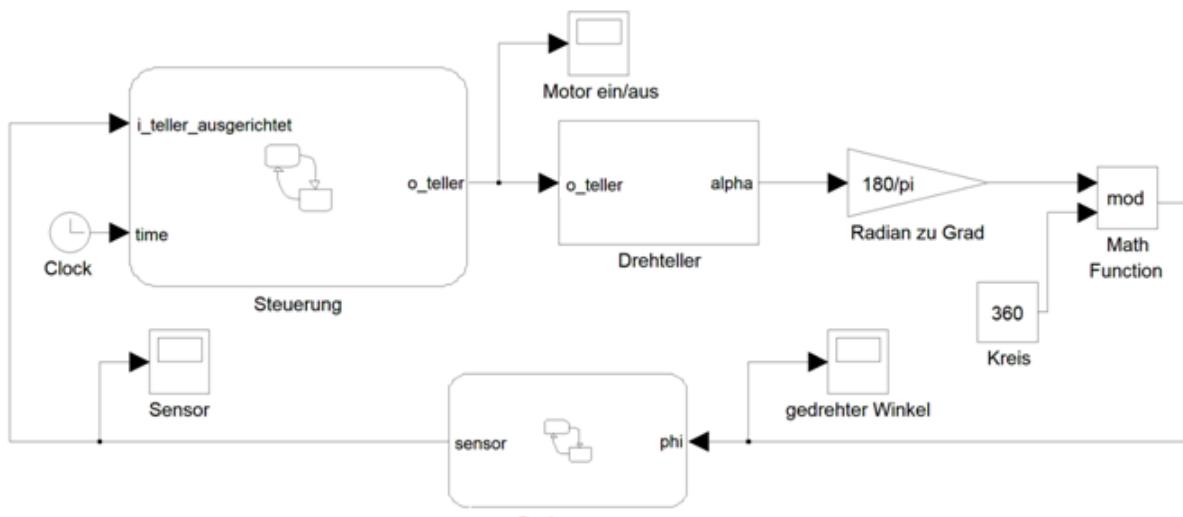


Abbildung 51: Modell und Steuerung eines Drehtellers

Überprüfung der Funktion:

Erstellen sie ein Scope, welches die Signale „*i_teller_ausgerichtet*“, den **gedrehten Winkel** sowie das Aktorsignal „*o_teller*“ als Eingänge hat.

Der Drehteller muss sich immer um 90° drehen, zwei Sekunden warten und anschließend den Vorgang wiederholen. **Überprüfen Sie sowohl den Drehwinkel als auch die Wartezeit!**

Was passiert, wenn die Toleranz des Winkelsensors auf $\pm 1^\circ$ reduziert wird?



Praktikum Simulationstechnik



Versuch 2



Inhaltsverzeichnis

6 SIMULATION EINER PRODUKTIONSANLAGE	61
6.1 AUFBAU.....	61
6.2 WERKSTÜCK ZYLINDER	62
6.3 FUNKTIONSWEISE DER ANLAGE.....	63
6.4 VORLAGE ZUR ERSTELLUNG DER SIMULATION UND DER STEUERUNG IN SIMULINK.....	64
6.5 EIN- UND AUSGÄNGE DER ANLAGE	65
6.6 UMWELT UND BEOBSAHTUNG VON ZUSTÄNDEN	66
7 SIMULATION DER MECHANIK.....	67
7.1 NÜTZLICHE BLÖCKE FÜR DIE NACHBILDUNG DER ANLAGE IN SIMULINK	67
7.2 BESCHREIBUNG DER MECHANISCHEN KOMPONENTEN	68
7.2.1 DER AUSSCHIEBER	68
7.2.2 DIE HEBEBÜHNE	69
7.2.3 DIE ZYLINDERAUFGNAHME	71
7.2.4 DIE ZYLINDERPRÜFUNG	72
7.2.5 DIE AUSSCHUSSRUTSCHE	73
7.2.6 DIE LUFTKISSEN RUTSCHE.....	74
7.2.7 ZUSAMMENBAU DER KOMPONENTEN	76
7.3 AUFGABE: ERSTELLUNG DES SIMULATIONS MODELLS DER ANLAGE PRÜFEN	77
8 STEUERUNG DER ANLAGE	78
8.1 BESCHREIBUNG DER STEUERUNG	78
8.2 AUFGABE: IMPLEMENTIERUNG DER STEUERUNG IN STATEFLOW UND VIRTUELLE INBETRIEBNAHME DER STATION.....	79
8.3 AUFGABE: INBETRIEBNAHME DER STATION (NICHT FÜR ONLINE PRAKTIKUM).....	81

6 Simulation einer Produktionsanlage

Dieser Teil des Praktikums soll Ihnen das Konzept der virtuellen Inbetriebnahme (VIBN) näherbringen. Automatisierte Produktionsanlagen haben in der Realität lange Fertigungszeiten, während derer die Steuerungssoftware bereits entwickelt, jedoch nicht an der Anlage getestet werden kann. Aus diesem Grund werden virtuelle Produktionsanlagen verwendet, um die Steuerung zu testen (daher der Name virtuelle Inbetriebnahme). Somit kann die Zeit bei der Entwicklung einer Anlage effizienter genutzt und Kosten gespart werden. Abbildung 52 veranschaulicht das Vorgehen zur virtuellen Inbetriebnahme. Zu Beginn steht die mechanische Entwicklung der Anlage. Diese liefert Spezifikationen für die Fertigung und die Entwicklung der Steuerungssoftware. Anhand dieser Spezifikationen wird für die virtuelle Inbetriebnahme eine Simulation der Anlage erstellt. Diese wird mit simulierten Steuerungssignalen getestet, da die Steuerungssoftware noch nicht vorhanden ist. Dieses Verfahren wird Software in the Loop (SIL) genannt. Mithilfe des Simulationsmodells wird die Steuerungssoftware erstellt. Das Ersetzen der realen Anlage durch ein Modell wird als Hardware in the Loop (HIL) bezeichnet. Bei der virtuellen Inbetriebnahme wird die Steuerungssoftware am Simulationsmodell getestet. Wenn dieser Test erfolgreich war wird die Inbetriebnahme durchgeführt, in der die Steuerung an der realen Anlage getestet wird.



Abbildung 52: Vorgehen bei der Entwicklung einer Produktionsanlage unter Verwendung von VIBN

In dieser Aufgabe soll ein Teil einer Anlage zum Sortieren von Zylindern nachgebildet werden. Der betrachtete Anlagenteil dient dem Prüfen der Zylinderhöhe. Mithilfe einer Simulation soll die Steuerung der Anlage entwickelt werden und eine virtuelle Inbetriebnahme durchgeführt werden. Zuletzt wird das Steuerungsprogramm an der realen Anlage getestet.

6.1 Aufbau

Abbildung 53 zeigt den Aufbau der gesamten Anlage. Es handelt sich dabei um eine Station des Anlagenpraktikums der Vorlesung „Grundlagen der modernen Informationstechnik II“. In der Mitte ist der zu bearbeitende Anlagenteil „Prüfen“ zu sehen. Links davon ist die Station „Verteilen“ abgebildet, welche Zylinder an diesen liefert. Rechts ist die Station „Sortieren“ zu sehen, an welche „Prüfen“ bestimmte Zylinder befördert.

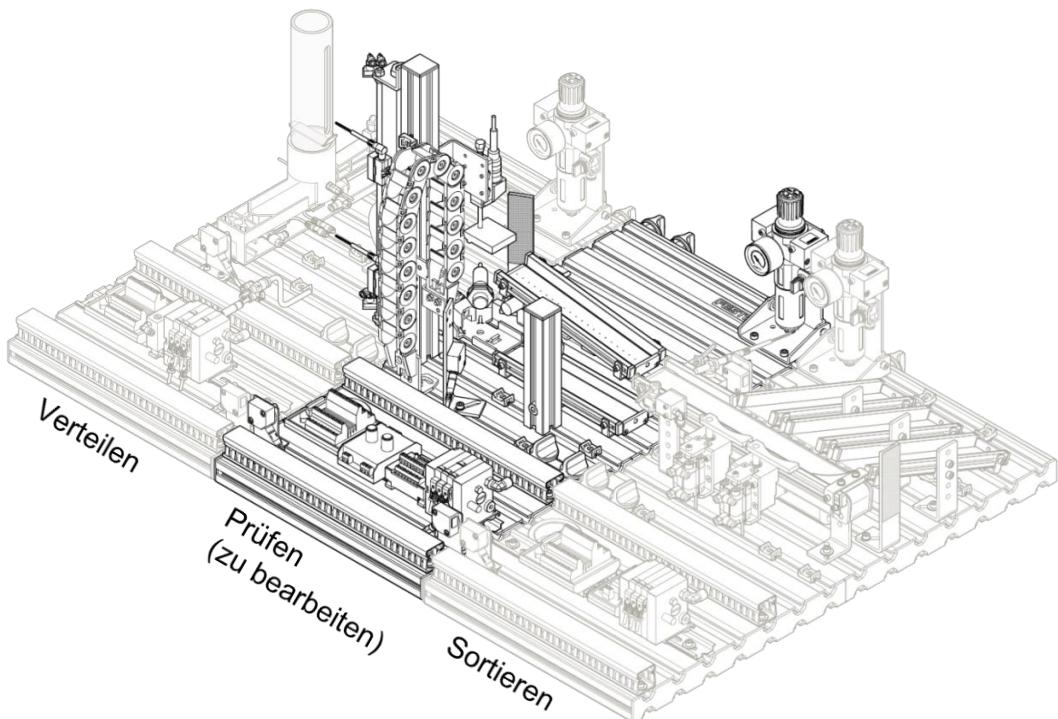


Abbildung 53: Darstellung der gesamten Anlage mit den Stationen Verteilen, Prüfen und Sortieren

6.2 Werkstück Zylinder

Jede der Stationen bearbeitet sogenannte Werkstücke. Dabei handelt es sich um zylindrische Plastikteile, bei denen sich auf einer Seite eine Öffnung befindet, auf die zusätzlich ein Deckel geschraubt werden kann. Diese Werkstücke sind aus verschiedenen Materialien gefertigt (Kunststoff rot, Kunststoff schwarz, Metall silbrig). Die schwarzen Zylinder sind etwas niedriger als die anderen, sodass diese mit Deckel dieselbe Höhe haben wie die andersfarbigen Werkstücke. Tabelle 4 zeigt Farben und Höhen der verschiedenen Zylinder.

Tabelle 4: Abmaße der Zylinder nach Farbe und Aufsatz

Farbe	Höhe	Höhe mit Deckel
Schwarz	22 mm	25 mm
Silber	25 mm	28 mm
Rot	25 mm	28 mm



Abbildung 54: Zylindrische Werkstücke der Sortieranlage: links schwarz, mittig rot, rechts metallisch.

6.3 Funktionsweise der Anlage

Im Folgenden wird die Funktion der Anlage „Prüfen“ beschrieben, wie sie vom Entwicklungsteam ausgelegt wurde. Vom vorherigen Segment werden Zylinder auf die *Zylinderaufnahme* (6) des zu behandelnden Anlagenteils gesetzt. Die Station „Prüfen“, in Abbildung 55: Aufbau der Station Prüfen dargestellt, sortiert Werkstücke ohne Deckel aus. Dafür wird das Werkstück über eine *Hebebühne* (2) angehoben und seine Höhe durch einen Wegtaster gemessen und mit einem Sollwert verglichen. Gleichzeitig wird die Farbe der Werkstücke (schwarz/nicht-schwarz) am *Erkennungsmodul* (1) unterschieden. Abhängig vom Ergebnis wird das Werkstück mithilfe des *Ausschiebers* (5) entweder auf der *Luftkissenrutsche* (3) zur Folgestation geleitet oder von der Hebebühne wieder abgesenkt und auf die *Ausschussrutsche* (4) befördert. Zylinder mit Deckel werden an die Folgestation geleitet, Zylinder ohne Deckel in die *Ausschussrutsche*. Um eine Kollision mit der Folgestation zu vermeiden ist ein Sensor enthalten, der ein Freigabesignal von dieser empfängt (7). Ein Sender (8), der ein Freigabesignal an die vorherige Station schickt, verhindert, dass dessen Kran mit der Hebebühne kollidieren kann.

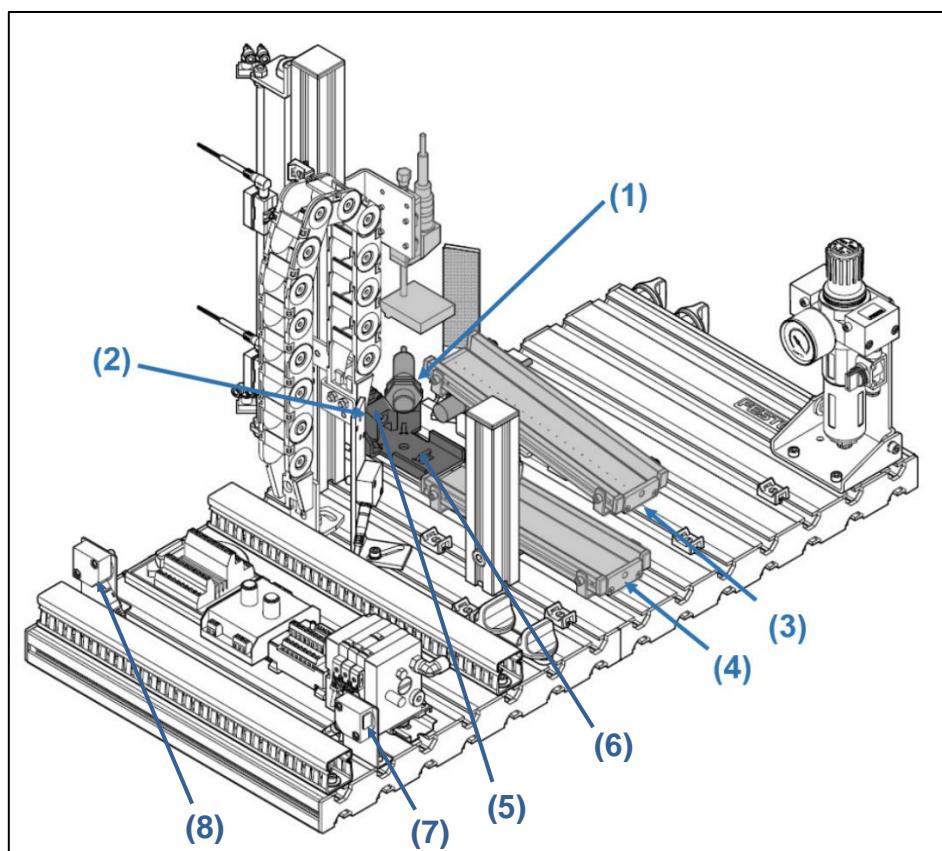


Abbildung 55: Aufbau der Station Prüfen, (1) Erkennungsmodul, (2) Hebebühne, (3) Luftkissenrutsche, (4) Ausschussrutsche, (5) Ausschieber, (6) Zylinderaufnahme

6.4 Vorlage zur Erstellung der Simulation und der Steuerung in Simulink

Zur Bearbeitung dieser Aufgabe erhalten Sie ein Vorlagenmodell, welches in Abbildung 56 zu sehen ist. Diese Vorlage wird im Folgenden erklärt.

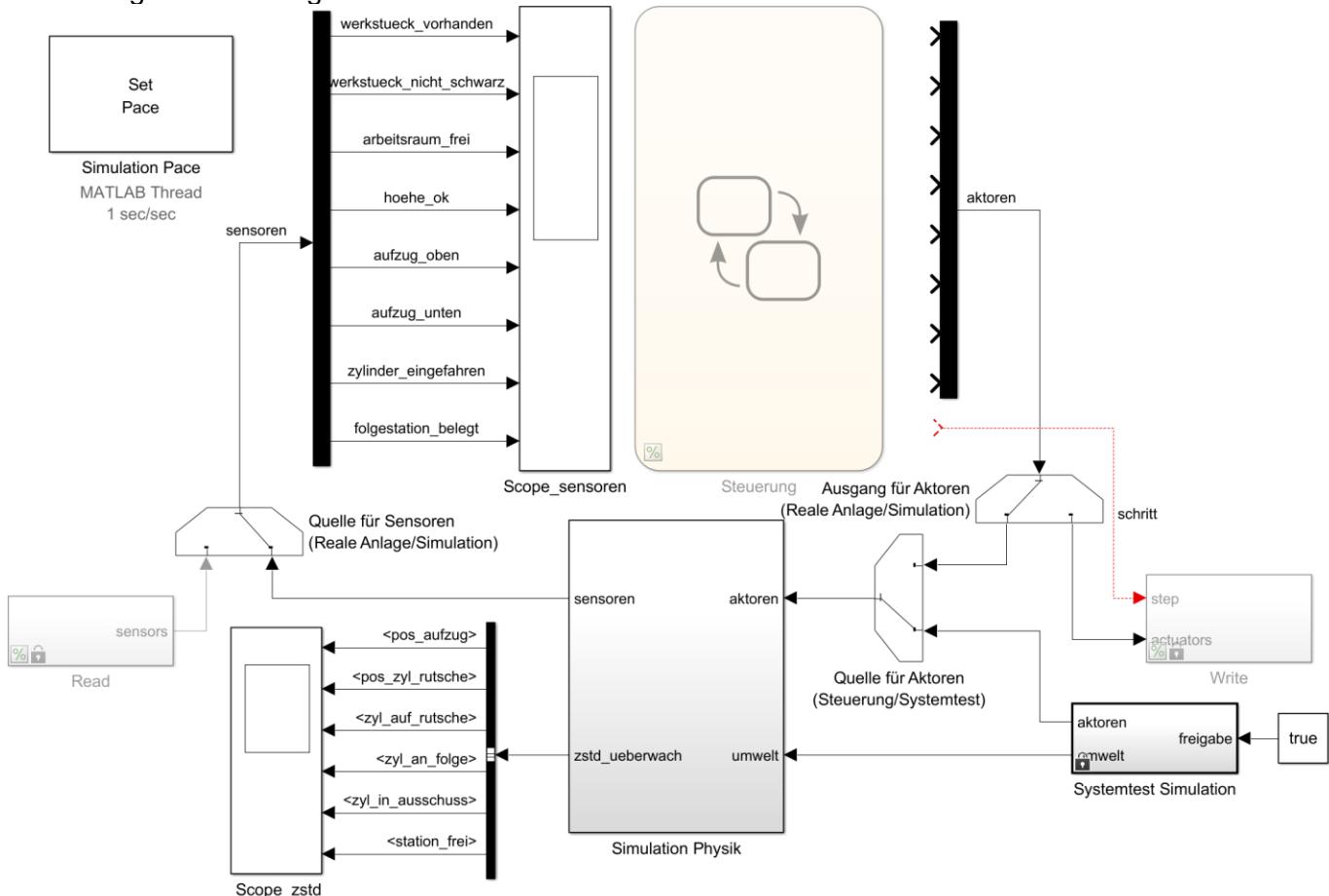


Abbildung 56: Simulink-Vorlage zur Bearbeitung der Aufgabe

Unten mittig ist das Subsystem „Simulation Physik“ zu sehen. In diesem Block werden Sie im ersten Teil der Aufgabe das Verhalten der Anlage nachbilden. Auf die Bedeutung der Eingänge „aktoren“ und „umwelt“ und Ausgänge „sensoren“ und „zstd_ueberwach“ wird in folgenden Abschnitten genauer eingegangen (6.5 und 6.6). Zum Testen der Simulation wird zunächst der „Systemtest Simulation“ benötigt, welcher an den Simulations-Block angeschlossen ist. Mit den Scopes, welche an „sensoren“ und „zstd_ueberwach“ angeschlossen sind, wird das richtige Verhalten der Simulation überprüft. Der Block „Systemtest Simulation“ ist nicht zu bearbeiten und soll nicht verändert werden. Für die Verwendung der Aktorsignale nach Erstellung der Steuerung ist der Schalter „Quelle für Aktoren“ enthalten. Schalten Sie diesen erst nach Abnahme Ihrer Simulation (Aufgabe in Abschnitt 7.3) um.

Im oberen Teil ist ein Stateflow-Diagramm zu sehen, welches in der Aufgabe in Abschnitt 8.2 an die Verbindungen „sensoren“ und „aktoren“ angeschlossen werden soll. Hierzu muss das Scope an „sensoren“ entfernt werden. Im letzten Schritt (Aufgabe in Abschnitt 8.3) soll die Steuerung an der realen Anlage getestet werden. Hierfür sind die Schalter „Quelle für Sensoren“ und „Ausgang für Aktoren“, sowie die Blöcke „Read“ und „Write“ enthalten, welche die Sensorsignale aus der echten Anlage lesen („Read“) und die Aktorsignale an die Anlage senden („Write“). Schalten Sie diese Schalter erst zu „Read“ und „Write“ um, nachdem Ihre Steuerung abgenommen wurde. Die Blöcke „Read“ und „Write“ sind nicht zu bearbeiten und sollen nicht verändert werden.

Links oben ist der Block „Simulation Pace“ zu sehen, welcher die Simulationsgeschwindigkeit festlegt. Nähere Informationen hierzu sind in Abschnitt 7.1 zu finden.

6.5 Ein- und Ausgänge der Anlage

Die Implementierung soll sich in zwei Teile gliedern: die Simulation der **Mechanik** und die **Steuerung** der Anlage. Die Mechanik wird in **Simulink** und die Steuerung in **Stateflow** implementiert.

Die Mechanik besteht aus den Sensoren (Tabelle 5) und Aktoren (Tabelle 6). Die Signale der Sensoren dienen als Input für die Steuerung. Die Aktoren werden nach Auswertung der Sensorsignale von der Steuerung gesetzt. Die Belegung dieser Ein- und Ausgänge ist durch den Anlagenentwurf festgelegt. Zu beachten ist außerdem, dass die Sensoren die Ausgänge der Simulation darstellen und somit als Eingänge für die Steuerung berücksichtigt werden müssen. Die Aktoren hingegen sind die Eingänge der Simulation und die Ausgänge der Steuerung.

Unter den Aktorsignalen der Anlage finden sich **mehrere ungenutzte Signale**. Diese sind für den Anschluss an die **reale Anlage** von Bedeutung. Hier ist durch die Verkabelung eine bestimmte Anordnung der Signale vorgegeben. Das Array der Aktorsignale wird an elektrische Schaltungen geleitet, welche in derselben Reihenfolge angeordnet sind wie die Signale im Array. Damit das richtige Signal mit der richtigen Schaltung verbunden wird, müssen die Positionen 5, 6 und 7 freigelassen werden. In Ihrer Implementierung legen Sie dazu den Wert 0 an diese Ausgänge.

Tabelle 5: Sensoren der Anlage

Signal	Variablenname	Beschreibung
1	sensoren.werkstueck_vorhanden	Auf dem Aufzug wurde ein Werkstück erkannt (liefert nur auf unterer Aufzugposition richtige Daten)
2	sensoren.werkstueck_nicht_schwarz	Werkstück auf dem Aufzug ist nicht schwarz (Sensor bewegt sich mit Aufzug mit)
3	sensoren.arbeitsraum_frei	Der Arbeitsraum ist frei, der Umsetzer der Vorgängerstation ist auf Ausgangsposition (am Magazin)
4	sensoren.hoehe_ok	Die gemessene Höhe liegt im eingestellten Intervall (schwarz mit Deckel / rot ohne Deckel / silbern ohne Deckel gibt jeweils eine 1, sonst 0)
5	sensoren.aufzug_oben	Der Aufzug ist oben
6	sensoren.aufzug_unten	Der Aufzug ist unten
7	sensoren.zylinder_eingefahren	Der Ausschiebezylinder ist zurückgezogen (wirft nicht aus)
8	sensoren.folgestation_belegt	Folgestation ist nicht für Aufnahme von Werkstücken bereit

Tabelle 6: Aktoren der Anlage

Signal	Variablenname	Beschreibung
1	aktoren.aufzug_abwaerts	Aufzug nach unten fahren
2	aktoren.aufzug_aufwaerts	Aufzug nach oben fahren
3	aktoren.zylinder_ausschieben	Ausschiebezylinder ausfahren, Werkstück auf eine der Rutschen schieben (Zylinder bewegt sich zurück, falls Wert nicht gesetzt)
4	aktoren.luft_an	Luftkissen auf der oberen Rutsche anschalten
5	-	Nicht genutzt
6	-	Nicht genutzt
7	-	Nicht genutzt
8	aktoren.station_frei	Freigabesignal an vorhergehende Station geben



6.6 Umwelt und Beobachtung von Zuständen

Neben den Sensor- und Aktorwerten müssen noch weitere Zustände berücksichtigt werden. Die Umwelt und die anderen Anlagenteile haben Auswirkungen auf das Verhalten der Anlage. Durch das Design der Anlage sind diese Parameter vorgegeben und werden von den Sensoren der Anlage ausgelesen. Diese Umwelteinflüsse sind in Tabelle 7 aufgelistet und müssen nicht von Ihnen programmiert werden. Um das Verhalten der simulierten Anlage beurteilen zu können, ist es nötig, bestimmte interne Zustände zu überwachen, welche in Tabelle 8 aufgelistet sind. Die Variablen der Zustandsüberwachung (abgekürzt „zstd_ueberwach“) sind nicht als Sensoren zu sehen, welche von der Steuerung ausgelesen werden, sondern dienen dem Überprüfen der richtigen Funktion. Beispielsweise kann man bei der realen Anlage visuell erkennen, wie viele Zylinder sich in der Ausschussrutsche befinden. In der Simulation ist dies jedoch nicht möglich, daher wird die Variable „zstd_ueberwach.zyl_in_ausschuss“ benötigt.

Tabelle 7: Umwelteinflüsse auf die Anlage

Signal	Variablenname	Beschreibung
1	umwelt.zylinder_platziert	Die vorhergehende Station platziert ein Werkstück in der Zylinderaufnahme
2	umwelt.zylinder_schwarz	Das aktuelle Werkstück ist schwarz
3	umwelt.zylinder_hoehe	Die Höhe des aktuellen Werkstücks in mm
4	umwelt.arbeitsraum_frei	Es befindet sich kein Objekt im Arbeitsraum, z. B. der Kran der vorherigen Station
5	umwelt.zylinder_entfernt_folge	Die Folgestation entfernt einen Zylinder am Ende der Luftkissenrutsche
6	umwelt.folge_belegt	Die Folgestation sendet ein Signal dafür, dass sie belegt ist

Tabelle 8: Zu überwachende Zustände der Anlage

Signal	Variablenname	Beschreibung
1	zstd_ueberwach.pos_aufzug	Position der Hebebühne über unterem Anschlag in mm
2	zstd_ueberwach.pos_zyl_rutsche	Position eines Zylinders auf der Luftkissenrutsche in mm vom Rutschenanfang. (0 falls kein Zylinder vorhanden ist)
3	zstd_ueberwach.zyl_auf_rutsche	Zeigt an, ob sich ein Zylinder auf der Luftkissenrutsche befindet
4	zstd_ueberwach.zyl_an_folge	Nachdem ein Zylinder die Luftkissenrutsche verlassen hat erreicht er die Folgestation. Dieses Signal zeigt an, ob sich ein Zylinder am Anfang der Folgestation befindet.
5	zstd_ueberwach.zyl_in_ausschuss	Zeigt die Anzahl der Zylinder auf der Ausschussrutsche an
6	zstd_ueberwach.station_frei	Zeigt an, ob diese Station frei ist, also ein neues Werkstück aufnehmen kann

7 Simulation der Mechanik

Bei der kontinuierlichen Simulation sind die Zustandsgrößen des Modells (Werte) zu jedem beliebigen Zeitpunkt berechenbar (kontinuierlich in der Zeit). Die Zeitvariable ist eine reelle Zahl, also ebenfalls kontinuierlich.

Die Beziehung zwischen den Zustandsgrößen und die Änderungen der Zustandsgrößen werden bei der kontinuierlichen Simulation meist durch Gleichungssysteme und Differentialgleichungssysteme beschrieben.

Bei der kontinuierlichen Simulation werden die erstellten Gleichungen durch numerische Lösungsverfahren gelöst. Zur numerischen Lösung werden in Simulink verschiedene Solver (das sind Integrationsmethoden) verwendet.

Zudem soll diese Simulation in Echtzeit ablaufen weshalb der Block „Simulation Pace“ verwendet werden muss. Dies ist nötig, da die Steuerung später an die reale Anlage angeschlossen werden soll, deren Verhalten in Echtzeit abläuft. Dieser muss für die Abnahme auf das Verhältnis 1:1 (Simulationszeit : Reale Zeit) eingestellt werden. Für ein schnelleres Durchlaufen der Simulation bei der Erstellung von einzelnen Komponenten kann ein anderes Verhältnis gewählt werden.

7.1 Nützliche Blöcke für die Nachbildung der Anlage in Simulink

Folgende Blöcke können bei der Erstellung der Simulation in Simulink hilfreich sein:

Name	Beschreibung	Symbol
S-R Flip-Flop	Eine steigende Signalflanke an S setzt den Ausgang Q auf 1. Eine steigende Signalflanke an R setzt Q auf 0. !Q gibt den gegenteiligen Wert von Q aus (!Q = 1, falls Q = 0 und !Q = 0, falls Q = 1). Hiermit lässt sich nachbilden, ob ein Objekt sich an einer bestimmten Position befindet.	
Detect Fall Nonpositive	Hiermit kann eine fallende Signalflanke erkannt werden. Wenn das vorherige Eingangssignal positiv war (NOT U/z <= 0) und das aktuelle Eingangssignal kleiner oder gleich null ist (U <= 0) wird 1 ausgegeben. Ansonsten ist der Ausgabewert 0.	
Detect Rise Positive	Hiermit kann eine steigende Signalflanke erkannt werden. Wenn das vorherige Eingangssignal nicht positiv war (NOT U/z > 0) und das aktuelle Eingangssignal größer null ist (U > 0) wird 1 ausgegeben. Ansonsten ist der Ausgabewert 0.	
Transport Delay	Dieser Block ermöglicht die zeitlich verzögerte Weitergabe des Eingangssignals.	
Simulation Pace	Hiermit kann die Simulationsgeschwindigkeit geändert werden. Mit dem Parameter „Simulation pace (sim sec per clock sec)“ kann das Zeitverhältnis zwischen realer und simulierter Zeit geändert werden.	



7.2 Beschreibung der mechanischen Komponenten

Die Anlage „Prüfen“ besteht aus den folgenden sechs mechanischen Komponenten:

- Ausschieber
- Aufzug
- Zylinderaufnahme
- Zylinderprüfung
- Luftkissenrutsche
- Ausschussrutsche

Der Aufbau sowie die Funktionsweise dieser Komponenten werden im Folgenden genauer erläutert.

7.2.1 Der Ausschieber

Aufbau

Der Ausschieber besitzt einen pneumatischen Zylinder, der mithilfe des Signals „aktoren.zylinder_ausschieben“ ausgeschoben wird. Mit ihm verbunden ist das Sensorsignal „sensoren.zylinder_eingefahren“. Zur Nachbildung der Dauer für Einziehen und Ausschieben soll eine Verzögerung von 0,5 s zwischen den Ausschiebe-Befehl und dem Ausgangssignal geschaltet werden. Das Ausgangssignal ist 1, wenn der Zylinder eingefahren ist und 0, wenn er ausgefahren ist. Der Ausschieber ist auf Abbildung 57 zusammen mit der Hebebühne zu sehen.

Ein- und Ausgangsgrößen

Eingänge:	
aktoren.zylinder_ausschieben	Wenn das Signal 1 ist muss der Zylinder ausgeschoben werden, wenn es 0 ist muss der Zylinder eingezogen werden.
Ausgänge:	
sensoren.zylinder_eingefahren	Ist 1, wenn der Zylinder eingezogen ist, 0 wenn er ausgeschoben ist.

Physikalische Randbedingungen

- Verzögerung für das Ein- und Ausfahren des Zylinders: **0,5 s**.

Tipps:

- Der Block **Transport Delay** verzögert ein Signal um eine bestimmte Anzahl an Sekunden.
- Der **Eingang** für eine simulierte Komponente ist ein **Aktor-Signal** oder eine **Umweltvariable**.
- Der **Ausgang** einer Komponente ist ein **Sensor-Signal** oder eine Variable der **Zustandsüberwachung**. (Abschnitte 6.5 und 6.6).



7.2.2 Die Hebebühne

Aufbau

Diese Komponente dient dazu Werkstücke zur Höhenprüfung zum Modul Zylinderprüfung anzuheben und ist auf Abbildung 57 zu sehen. Mithilfe der Aktorensignale „aktoren.aufzug_aufwaerts“ und „aktoren.aufzug_abwaerts“ wird die Fahrtrichtung des Aufzugs gesteuert. Achtung: **Nur wenn genau einer der beiden Eingänge 1 ist darf der Aufzug fahren.** Die höchstmögliche Position ist 125 mm. Die untere Grenze der Hebebühne liegt bei 0 mm.

Die Geschwindigkeit beim Heben und Senken beträgt jeweils 30 mm/s. Zum Anheben existiert ein Hubzylinder, dessen Endlagen abgefragt werden können. Sobald die Hebebühne sich höher als 124 mm befindet wird das Sensorsignal „sensoren.aufzug_oben“ ausgelöst. Wenn sich die Hebebühne unter einer Höhe von 1 mm befindet wird das Sensorsignal „sensoren.aufzug_unten“ ausgelöst. Zur Überwachung der Hebebühne soll zusätzlich die Position „zstd_ueberwach.pos_aufzug“ ausgegeben werden.

Ein- und Ausgangsgrößen

Eingänge:	
aktoren.aufzug_aufwaerts	Bewegt die Hebebühne nach oben
aktoren.aufzug_abwaerts	Bewegt die Hebebühne nach unten
Ausgänge:	
sensoren.aufzug_unten	Zeigt die untere Endlage der Hebebühne an
sensoren.aufzug_oben	Zeigt die obere Endlage der Hebebühne an
zstd_ueberwach.pos_aufzug	Zeigt die Position der Hebebühne an

Physikalische Randbedingungen

- Die **Grenzen** der Hebebühne liegen bei **0 mm und 125 mm**.
- Zu **Anfang** ist befindet sich die Hebebühne bei der Höhe **0 mm**.
- **Geschwindigkeit** zum Heben und Senken: **30 mm/s**.
- Der Sensor für die untere Position löst bei einer Höhe **unter 1 mm** aus.
- Der Sensor für die obere Position löst bei einer Höhe **über 124 mm** aus.

Tipps:

- Verwenden Sie einen **begrenzten Integrator**, um die Höhenänderung der Hebebühne zu simulieren
- Verwenden Sie die Blöcke Signalgenerator, Display und Scope, um die Funktion zu prüfen. Überlegen Sie sich passende Eingangssignale, um Ihre Hebebühne zu testen.

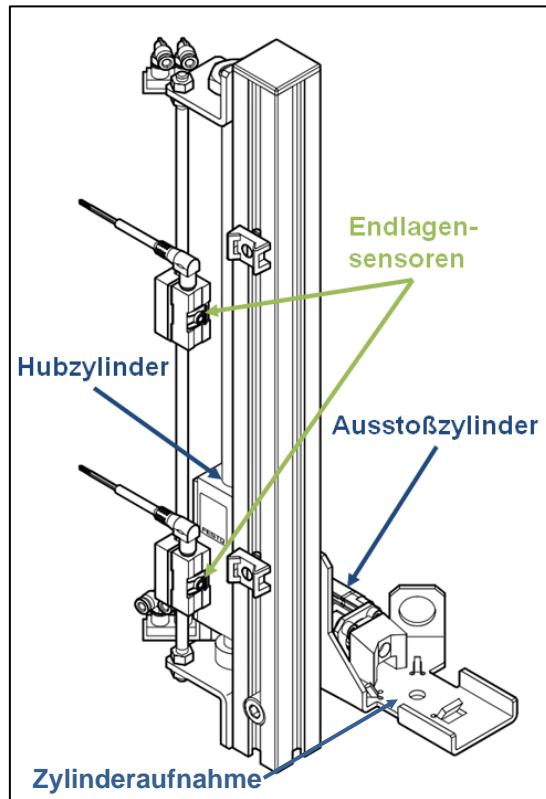


Abbildung 57: Hebebühne mit Sensoren, Zylinderaufnahme und Ausstoßzylinder



7.2.3 Die Zylinderaufnahme

Aufbau

Mit dieser Komponente soll festgestellt werden, ob sich ein Werkstück auf der Hebebühne befindet. Sie ist auf Abbildung 57 zu sehen. Das Ergebnis dieser Überprüfung wird an das Sensorsignal „sensoren.werkstueck_vorhanden“ übertragen, welcher den kapazitiven Näherungssensor (Abbildung 58) nachbildet.

Ein Zylinder kann in die Aufnahme platziert werden, wenn der Ausschieber eingefahren ist („sensoren.zylinder_eingefahren“) und sich die Hebebühne in unterer Endlage befindet („sensoren.aufzug_unten“). Erst wenn die genannten Bedingungen zutreffen, kann ein Zylinder durch die vorherige Station positioniert werden („umwelt.zylinder_platziert“). Um sicherzustellen, dass sich die Hebebühne unten befindet und der Ausschieber eingezogen ist bevor ein Zylinder platziert wird, soll die positive Signalflanke von „umwelt.zylinder_platziert“ verwendet werden. Sobald der Ausschieber von eingezogen auf nicht eingezogen wechselt, wird ein Werkstück ausgeschoben und somit der Wert von „sensoren.werkstueck_vorhanden“ auf 0 gesetzt. Eine negative Signalflanke von „sensoren.zylinder_eingefahren“ bewirkt, dass das Werkstück ausgeschoben und somit aus der Aufnahme entfernt wird.

Ein- und Ausgangsgrößen

Eingänge:	
umwelt.zylinder_platziert	Die vorhergehende Station platziert einen Zylinder in die Aufnahme. Die steigende Taktflanke dieses Signals soll verwendet werden.
sensoren.zylinder_eingefahren	Der Ausschieber ist eingefahren.
sensoren.aufzug_unten	Zeigt an, ob sich die Hebebühne in unterer Endlage befindet.
Ausgänge:	
sensoren.werkstueck_vorhanden	Gibt an, ob sich ein Werkstück in der Aufnahme befindet.

Physikalische Randbedingungen

- Ein Zylinder kann in die Aufnahme platziert werden, wenn der Ausschieber eingefahren ist („sensoren.zylinder_eingefahren“) und die Hebebühne sich in unterer Endlage befindet („sensoren.aufzug_unten“).
- Ein Zylinder wird entfernt, wenn der Ausschieber ausgeschoben wird (negative Signalflanke von „sensoren.zylinder_eingefahren“).

Tipps:

- Verwenden sie ein S-R Flip-Flop, um das Vorhandensein eines Zylinders nachzubilden.
- Eine positive bzw. negative Signalflanke kann mit den Blöcken Detect Rise Positive bzw. Detect Fall Nonpositive erkannt werden.
- Verwenden Sie die Blöcke Signalgenerator und Display, um die Funktion zu prüfen. Überlegen Sie sich passende Eingangssignale, um die Zylinderaufnahme zu testen.

7.2.4 Die Zylinderprüfung

Hiermit sollen die Höhe und die Farbe eines Zylinders in der Aufnahme geprüft werden. Diese Aufgabe übernehmen in der Realität der optische Sensor und der Wegtaster mit Komparator, zu sehen auf Abbildung 58. Nur wenn sich ein Zylinder in der Zylinderaufnahme befindet („sensoren.werkstueck_vorhanden“), kann die Farbe geprüft werden, welche von „umwelt.zylinder_schwarz“ bereitgestellt wird. Solange kein Zylinder vorhanden ist, ist der Wert von „sensoren.werkstueck_nicht_schwarz“ 1, da das Umgebungslicht so hell ist, dass dieser Sensor den Wert „nicht schwarz“ annimmt. Es ist zu beachten, dass dieser Input für den Sensorwert „sensoren.werkstueck_nicht_schwarz“ negiert werden muss. Die Zylinderhöhe kann nur geprüft werden, wenn sich ein Werkstück in der Aufnahme befindet und die Hebebühne sich in oberer Endlage („sensoren.aufzug_oben“) befindet. Wenn diese Bedingungen erfüllt sind und die Zylinderhöhe zwischen 24 mm und 26 mm liegt, gibt „sensoren.hoehe_ok“ eine 1 aus, ansonsten 0.

Ein- und Ausgangsgrößen

Eingänge:	
umwelt.zylinder_schwarz	Zeigt an, ob das aktuelle Werkstück schwarz ist.
umwelt.zylinder_hoehe	Zeigt die Höhe des aktuellen Werkstücks in mm an.
sensoren.werkstueck_vorhanden	Zeigt an, ob ein Werkstück in der Aufnahme liegt.
sensoren.aufzug_oben	Zeigt an, ob die Hebebühne in oberer Endlage ist.
Ausgänge:	
sensoren.werkstueck_nicht_schwarz	Zeigt an, ob das Werkstück nicht schwarz ist, ist 1 falls kein Werkstück vorhanden ist.
sensoren.hoehe_ok	Ist 1 falls die Zylinderhöhe im Bereich [24, 26] mm liegt.

Physikalische Randbedingungen

- Die **Zylinderhöhe** muss zwischen einschließlich **24 mm und 26 mm** liegen, damit das Signal „sensoren.hoehe_ok“ positiv wird.
- Das Umgebungslicht verursacht ein positives Signal auf „sensoren.werkstueck_nicht_schwarz“.
- Die Prüfung der Höhe ist nur dann richtig, wenn sich die Hebebühne in **oberer Endlage** befindet.

Tipps:

- Beachten Sie, dass „sensoren.hoehe_ok“ nicht zwangsweise angibt, ob die gewünschte Werkstückhöhe vorliegt, sondern lediglich, dass sie zwischen 24 mm und 26 mm liegt (vgl. Tabelle 4).
- Der Block Interval Test prüft, ob sich ein Signal in einem bestimmten Bereich befindet.
- Verwenden Sie die Blöcke Manual Switch und Display oder Signal Generator und Scope, um die Funktion zu prüfen. Überlegen Sie sich passende Eingangssignale, um die Zylinderprüfung zu testen.

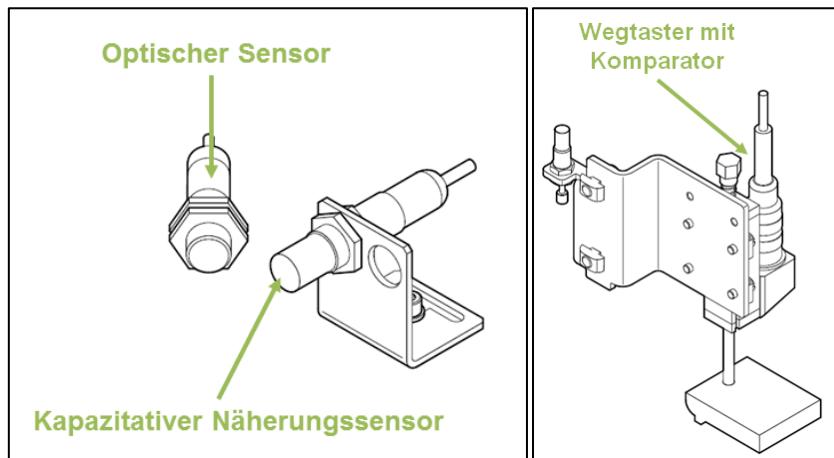


Abbildung 58: Sensoren der Zylinderprüfung (Farberkennung, Näherungssensor, Wegtaster)



7.2.5 Die Ausschussrutsche

Zylinder, die keinen Deckel haben, werden auf die Ausschussrutsche befördert. Damit ein Zylinder auf diese Rutsche bewegt werden kann, muss sich die Hebebühne in unterer Endlage befinden („sensoren.aufzug_unten“). Die Übergabe eines Zylinders von der Zylinderaufnahme auf die Rutsche soll über das Signal „sensoren.werkstueck_vorhanden“ detektiert werden. Bei einer fallenden Signalflanke von „sensoren.werkstueck_vorhanden“, während sich die Hebebühne unten befindet, soll ein Zähler inkrementiert werden, welcher die Anzahl der Zylinder auf der Ausschussrutsche angibt („zstd_ueberwach.zyl_in_ausschuss“). Verwenden Sie für den Zähler ein getriggertes Subsystem, welches einen Memory-Block zum Speichern der Zylinderanzahl enthält.

Ein- und Ausgangsgrößen

Eingänge:	
sensoren.werkstueck_vorhanden	Gibt an, ob ein Werkstück in der Aufnahme liegt
sensoren.aufzug_unten	Gibt an, ob die Hebebühne sich in unterer Endlage befindet
Ausgänge:	
zstd_ueberwach.zyl_in_ausschuss	Gibt die Anzahl der Zylinder auf der Ausschussrutsche an

Physikalische Randbedingungen

- Nur wenn sich die Hebebühne unten befindet, kann ein Zylinder durch fallende Flanke von „sensoren.zylinder_vorhanden“ auf die Ausschussrutsche geschoben werden.

Tipps:

- Verwenden Sie die Blöcke Manual Switch und Display oder Signal Generator und Scope, um die Funktion zu prüfen. Überlegen Sie sich passende Eingangssignale, um die Ausschussrutsche zu testen.

7.2.6 Die Luftkissenrutsche

Zylinder mit Deckel werden auf die Luftkissenrutsche (Abbildung 59) geschoben und zur Folgestation befördert. Eine Luftkissenrutsche besteht aus einem Hohlkörper, in den Luft eingeblasen wird (1). Diese Luft tritt an kleinen Öffnungen wieder aus und hebt dadurch das darauf liegende Objekt an (2). Die Neigung α der Rutsche gegenüber dem Horizont bewirkt, dass sich die Gravitationskraft F_g in eine parallele Komponente zur Rutsche $F_{g,x}$ (x -Richtung) und eine senkrechte Komponente zur Rutsche $F_{g,y}$ (y -Richtung) aufteilt wird. Während die y -Komponente durch die ausströmende Luft ausgeglichen wird, wirkt x -Komponente beschleunigend auf den Zylinder.

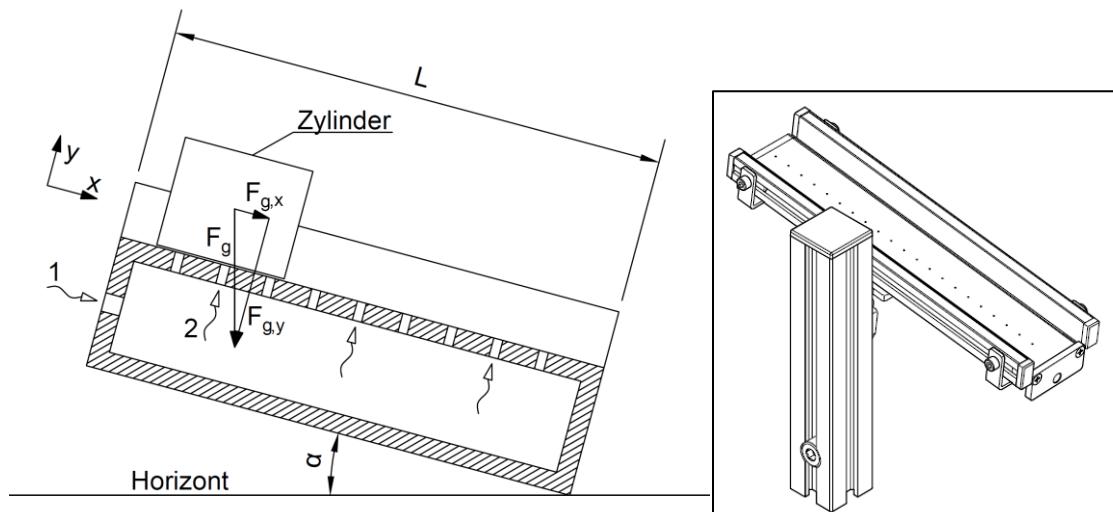


Abbildung 59: Funktionsprinzip und isometrische Ansicht der Luftkissenrutsche

Zur Detektion eines neuen Werkstücks auf der Rutsche soll geprüft werden, ob die Hebebühne sich in oberer Endlage befindet („sensoren.aufzug_oben“) und ob das Signal „sensoren.zylinder_vorhanden“ eine fallende Taktflanke aufweist. Falls diese Bedingungen erfüllt sind, muss das Signal „zstd_ueberwach.zyl_auf_rutsche“ auf 1 gesetzt werden. Wenn sich ein Zylinder auf der Rutsche befindet und die Luftzufuhr eingeschalten ist („aktoren.luft_an“), wird er durch Gravitation beschleunigt. Gleichzeitig wirkt Reibung entgegen der Bewegungsrichtung, welche proportional zur Geschwindigkeit ist. Somit kann die Position auf der Rutsche durch folgende Differentialgleichung dargestellt werden.

$$\ddot{x} = g \sin \alpha - k \dot{x},$$

wobei $g = 9810 \frac{\text{mm}}{\text{s}^2}$ beträgt und $k = 1.5 \text{ s}^{-1}$ angenommen wird. Der Neigungswinkel der Rutsche beträgt $\alpha = 2^\circ$. Die Position des Zylinders soll an „zstd_ueberwach.pos_zyl_rutsche“ übergeben werden. Sobald der Zylinder am Ende der Rutsche angelangt ist ($x > 229 \text{ mm}$) wird „zstd_ueberwach.zyl_auf_rutsche“ auf 0 und die Variable „zstd_ueberwach.zyl_an_folge“ auf 1 gesetzt. Wenn sich kein Zylinder auf der Rutsche befindet, sollen die Integratoren zurückgesetzt werden, sodass der nächste Zylinder, der auf die Rutsche gesetzt wird, bei Position $x = 0 \text{ mm}$ beginnt. Achtung: Das Signal zum Zurücksetzen der Integratoren muss durch einen Memory-Block oder einem Delay-Block mit einer Verzögerung von einem Schritt verzögert werden, da sonst fehlerhaftes Verhalten auftritt.

Physikalische Randbedingungen

- Die Länge der Rutsche beträgt $L = 230 \text{ mm}$.
- Die Neigung der Rutsche beträgt $\alpha = 2^\circ$.
- Die Erdbeschleunigung beträgt $g = 9810 \frac{\text{mm}}{\text{s}^2}$.
- Der Reibungsfaktor beträgt $k = 1.5 \text{ s}^{-1}$.
- Die Position des Zylinders liegt im Bereich $0 \text{ mm} \leq x \leq 230 \text{ mm}$.



Ein- und Ausgangsgrößen

Eingänge:	
sensoren.werkstueck_vorhanden	Zeigt an, ob ein Werkstück in der Aufnahme liegt.
sensoren.aufzug_oben	Zeigt an, ob die Hebebühne in oberer Endlage ist.
aktoren.luft_an	Aktiviert die Luftkissenrutsche.
umwelt.zylinder_entfernt_folge	Eine Steigende Signalflanke entfernt ein Werkstück vom Anfang der Folgestation.
Ausgänge:	
zstd_ueberwach.pos_zyl_rutsche	Zeigt den Abstand eines Werkstücks zum Rutschenanfang in mm an.
zstd_ueberwach.zyl_auf_rutsche	Zeigt an, ob sich ein Zylinder auf der Rutsche befindet.
zstd_ueberwach.zyl_an_folge	Zeigt an, ob sich ein Werkstück am Anfang der Folgestation befindet.

Tipps:

- Verwenden Sie S-R Flip-Flops um die Zustände „zstd_ueberwach.zyl_auf_rutsche“ und „zstd_ueberwach.zyl_an_folge“ nachzubilden.
- Verwenden Sie einen unbegrenzten Integrator mit Rücksetzfunktion für die Berechnung der Geschwindigkeit und einen begrenzten Integrator mit Rücksetzfunktion für die Berechnung der Position.
- Verwenden Sie die Blöcke Manual Switch und Display oder Signal Generator und Scope, um die Funktion zu prüfen. Überlegen Sie sich passende Eingangssignale, um die Luftkissenrutsche zu testen.



7.2.7 Zusammenbau der Komponenten

Zum Zusammensetzen der Komponenten können Sie sich an den Variablennamen orientieren. Beispielsweise benötigen mehrere Subsysteme das Signal „sensoren.werkstueck_vorhanden“. Dieses Ausgangssignal aus der Zylinderaufnahme ist mit jedem Eingangssignal das „sensoren.werkstueck_vorhanden“ benötigt zu verbinden. Genauso ist mit allen weiteren Signalen zu verfahren.

Nachdem alle Subsysteme miteinander und mit den Ein- und Ausgangsvariablen (Sensoren, Aktoren, Umwelt und Zustandsüberwachung) verbunden sind, sollten noch die Signale „aktoren.station_frei“, „zstd_ueberwach.station_frei“, „umwelt.folge_belegt“, „sensoren.folgestation_belegt“, „umwelt.arbeitsraum_frei“ und „sensoren.arbeitsraum_frei“ ohne Verbindung sein.

Einige dieser Signale werden von der Anlage nicht modifiziert, sondern direkt mit der Steuerung verbunden, also durchgeschliffen. „aktoren.station_frei“ soll zu „zstd_ueberwach.station_frei“ durchgeschliffen werden und „umwelt.folge_belegt“ zu „sensoren.folgestation_belegt“. Damit das Signal „sensoren.arbeitsraum_frei“ positiv ist, müssen „umwelt.arbeitsraum_frei“ und „sensoren.aufzug_unten“ wahr sein.

Der Eingangsport „freigabe“ des Blocks „Systemtest Simulation“ ist in der Vorlage mit der Konstanten true belegt. Belassen Sie dies so wie es ist für den Test des Simulationsmodells.

Tipps:

- Verbinden Sie alle gleichnamigen Signale aus Subsystemen Aktoren, Sensoren, Umwelt und Zustandsüberwachung miteinander.
- Verbinden Sie „aktoren.station_frei“ direkt mit „zstd_ueberwach.station_frei“ und „umwelt.folge_belegt“ direkt mit „sensoren.folgestation_belegt“.
- „umwelt.arbeitsraum_frei“ und „sensoren.aufzug_unten“ müssen beide den Wert 1 haben, um „sensoren.arbeitsraum_frei“ auf 1 zu schalten.
- Sie können die Blöcke „Goto“ und „From“ verwenden, um die Übersichtlichkeit in Systemen mit vielen Verbindungslinien zu verbessern. Anstatt der Linien werden dann Marken mit dem Namen des Signals angezeigt.



7.3 Aufgabe: Erstellung des Simulationsmodells der Anlage Prüfen

- Verwenden Sie die für diese Aufgabe bereitgestellte **Vorlage**. Die Bearbeitung dieser Aufgabe erfolgt nur im **Subsystem „Simulation Physik“**, welches in der Vorlage zu finden ist.
- Erstellen Sie jeweils ein Subsystem für die Teilsysteme **Ausschieber, Hebebühne, Zylinderaufnahme, Zylinderprüfung, Ausschussrutsche und Luftkissenrutsche** in dem vorgegebenen **Simulink-Modell** auf Basis der vorangegangenen Beschreibung.
- Überlegen Sie sich eine Möglichkeit **die einzelnen Komponenten** zu testen. Lassen Sie jede Einzelkomponente abnehmen bevor Sie die Komponenten zusammenfügen.
- Das vollständige Modell wird mit dem Block „Systemtest Simulation“ überprüft.

Eine Aufgabe gilt erst dann als gelöst, wenn Sie die Funktionsfähigkeit ihres Programmes NACHWEISEN und ERKLÄREN können. Benennen Sie hierzu alle relevanten Signale und Blöcke. Betrachten Sie die Ein- und Ausgangssignale in EINEM Scope, in dem alle Signale benannt sind. Überlegen Sie sich sinnvolle Testsignale, um die Funktionalität zu verifizieren.

Tipps:

- Einige Bestandteile der Komponenten sind sehr ähnlich aufgebaut. Sie können diese **Teile wiederverwenden**.
- Wichtiger Hinweis: Bei der Modellierung sollte immer der Simulationszweck im Auge behalten werden. Hier geht es ausschließlich darum, **der Steuerung die gleichen Eingangssignale und Reaktion auf Ausgangssignale darzustellen wie in der realen Maschine**. Darum müssen viele Details nicht mitsimuliert **werden**, z. B. die genaue Position des Ausschiebers.
- Zur Abnahme wird das Subsystem „Testsystem Simulation“ an Ihre Simulation angebunden. Sie können Ihre Simulation bereits vor der Abnahme mit diesem Block testen.
- Zum **Konvertieren zwischen Datentypen** (z. B. boolean in double) dient der Block „Data Type Conversion“.



8 Steuerung der Anlage

Die Steuerung soll in einem Stateflow Diagramm erstellt werden, welches zunächst die Sensorsignale der Simulation als Eingabe erhält und Aktorsignale an die Simulation sendet.

8.1 Beschreibung der Steuerung

Zu Anfang befindet sich der Aufzug an der unteren Position, der Ausstoßzylinder ist eingezogen, es ist dort kein Werkstück vorhanden und die Luftrutsche ist ausgeschaltet. Zudem wird kein Freigabesignal gesendet („aktoren.station_frei“). Wenn der Ausgangszustand hergestellt ist, beginnt die Anlage mit folgender Routine:

In diesem Zustand (kein Werkstück vorhanden) wird der vorhergehenden Station das Freigabesignal gegeben, welche daraufhin ein Werkstück liefert. Sobald ein Werkstück erkannt wird, wird das Freigabesignal zurückgenommen. Wenn der Arbeitsraum frei ist, wird nach kurzer Wartezeit (ca. 0,5 Sekunden) das Werkstück mit Hilfe der Hebebühne angehoben und durch das resultierende Andrücken an das Messmodul auf seine Bauhöhe überprüft. Vor der Höhenmessung muss kurz gewartet werden, um ein Nachschwingen der Feder zu vermeiden. Gleichzeitig kann mit dem optischen Sensor festgestellt werden, ob das Bauteil schwarz ist oder nicht.

Werkstücke mit Deckel werden auf die Luftkissenrutsche ausgestoßen, sobald das Freigabesignal der Folgestation vorliegt. Hierzu wird der der Ausstoßzylinder verwendet und die Luftkissenrutsche eingeschalten. Nach einer Wartezeit von ca. 2 Sekunden wird die Luftzufuhr ausgestellt.

Handelt es sich dagegen um ein Werkstück ohne Deckel, wird das Werkstück erst auf der unteren Hebebühnenposition auf die Ausschussrutsche ausgestoßen. Schwarze Werkstücke mit Deckel liefern eine positive Höhenprüfung und eine negative Prüfung der Helligkeit. Andersfarbige Werkstücke liefern eine negative Höhenprüfung und eine positive Helligkeitsprüfung. Somit ist für Werkstücke mit Deckel die Operation „sensoren.hoehe_ok“ != „sensoren.werkstueck_nicht_schwarz“ immer wahr und für Werkstücke ohne Deckel immer falsch. Dies ist in Tabelle 9 zusammengefasst. Nach dem Ausstoßen wird die Anlage wieder in ihren Ausgangszustand gebracht und der Ablauf beginnt von vorne.

Hinweise:

- Bei den Messungen sollte sich das Werkstück eine kurze Zeit (z. B. eine Sekunde) in Ruhe bei dem Sensor befinden, bevor der Wert abgefragt wird.
- Es gibt keinen Sensor mit dem geprüft werden kann, ob der Ausstoßzylinder vollständig ausgefahren ist. Verwenden sie stattdessen eine geeignete Kontrollstruktur.
- Vor dem Hebevorgang des Aufzuges muss immer auf einen freien Arbeitsraum geachtet werden!
- Bei Vorgängen deren Endposition nicht durch Endlagensensoren überprüft werden können, müssen Sie zur Annahme des Erreichens der Endlage Zeitmessungen durchführen. Gehen Sie zunächst von ausreichend großen Werten aus (z. B. fünf Sekunden), bevor Sie die Zeiten später optimieren.
- Ziehen Sie folgende Tabelle zu Rate, um Werkstücke mit Deckel mit den gegebenen Sensoren zu identifizieren:

Tabelle 9: Unterscheidung der Sensorwerte bei der Zylinderprüfung

		Mit Deckel		Ohne Deckel	
		hoehe_ok	hoehe_ok!=werkstueck_nicht_schwarz	hoehe_ok	hoehe_ok!=werkstueck_nicht_schwarz
		werkstueck_nicht_schwarz		werkstueck_nicht_schwarz	
schwarz	1			0	
	0		1	0	
rot oder metal-lisch	0			1	
	1		1	1	

8.2 Aufgabe: Implementierung der Steuerung in Stateflow und virtuelle Inbetriebnahme der Station

Die Sensorsignale sind in Vektoren boolscher Variablen zusammengefasst, da dies von Spezifikation der Signalübertragung vorgegeben ist.

Diese Vektoren müssen mit dem Block Demux in einzelne Signale aufgeteilt werden und an die Eingänge des Steuerungs-Blocks angeschlossen werden. Beachten Sie hierbei die Signalbelegung aus Abschnitt 6.5. Der erste Ausgang des Demuxers entspricht Signal 1 aus der Tabelle der Sensoren usw. In Abbildung 60 ist ein Beispiel für den Demuxer dargestellt.

Die Aktoren der Maschine werden ebenfalls als Vektoren deklariert. Diese müssen aus den Ausgabesignalen des Steuerungs-Blocks mit dem Block Mux zusammengefasst werden, bevor sie an die Simulation übergeben werden.

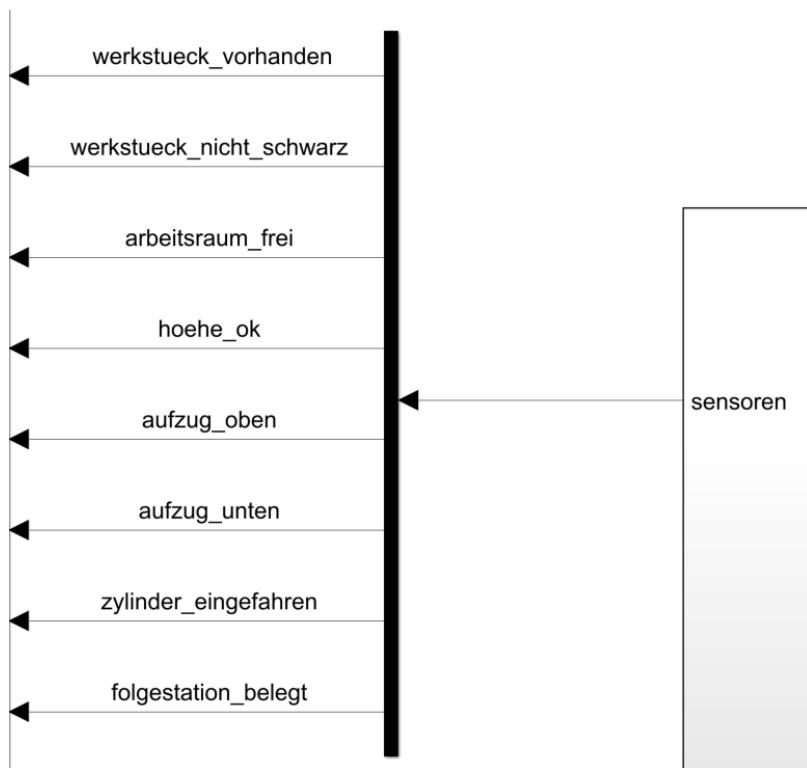


Abbildung 60: Beispiel zur Verwendung des Simulink-Blocks Demux

Implementieren Sie die Steuerung in Stateflow in Form eines Zustandsautomaten. Die Steuerung soll anschließend mit Hilfe der erstellten Simulation auf Richtigkeit geprüft und gegebenenfalls angepasst werden. Ziel ist es, die Simulation korrekt zu automatisieren, um später den erstellten Ablauf direkt auf der spezifizierten Anlage in Betrieb nehmen zu können.

- Zur Implementierung der Steuerung erzeugen Sie zunächst ein Stateflow-Chart. Fügen Sie in diesem für jedes Sensorsignal ein Input Data-Item und für jedes Aktorsignal ein Output Data-Item hinzu. Beachten Sie hierbei die Reihenfolge der Signale aus Abschnitten 6.5 und 6.6.
- Falls Sie das Diagramm aus der Vorlage verwenden, beachten Sie, dass dieses auskommentiert ist. Um dies rückgängig zu machen, wählen Sie Rechtsklick -> „uncomment“ auf diesen Block oder verwenden Sie die Tastenkombination „Strg+Shift+X“.
- Verbinden Sie die Ein- und Ausgänge des Stateflow-Charts mit den demuxten Variablen aus Simulink.
- Fügen Sie in Ihrem Steuerungsdiagramm für jeden Zustand in der Beschreibung der Steuerung (Abschnitt 8.1) einen State-Block in das Diagramm.



- Überlegen Sie sich welche Bedingungen erfüllt sein müssen, um von einem Zustand zum nächsten überzugehen.
- Setzen Sie in jedem State die einzustellenden Aktoren auf den richtigen Wert.
- Fügen Sie eine zusätzliche Schrittvariable „schritt“ als Ausgang zu Ihrem Steuerungs-Chart hinzu und überwachen Sie diese von außen mit einem Scope oder Display. Die Variable soll den aktuell ausgeführten Zustand angeben und hat den Datentyp Integer.
- Die unbenutzten Aktorsignale (siehe Abschnitt 6.5) sollen mit der Konstanten 0 belegt werden.
- Zum Testen der Steuerung stellen Sie bitte den Schalter „Quelle für Aktoren“ um (Doppelklick), so dass die Aktorsignale der Steuerung die Simulation der Anlage erreichen.
- Entfernen Sie den Konstanten Wert true vom Eingang „freigabe“ des Blocks „Systemtest Simulation“ und verbinden Sie diesen mit dem Signal „aktoren_station_frei“ der Steuerung. Dies ist notwendig, damit der Systemtest nur dann einen Zylinder platziert, wenn dies von der Steuerung erlaubt wird.

Bitte beachten Sie bei der Umsetzung folgende Hinweise:

Eine Aufgabe gilt erst dann als gelöst, wenn Sie die Funktionsfähigkeit ihres Programmes NACHWEISEN und ERKLÄREN können. Benennen Sie hierzu alle relevanten Signale und Blöcke. Betrachten Sie die Ein- und Ausgangssignale in EINEM Scope, in dem alle Signale benannt sind. Überlegen Sie sich sinnvolle Testsignale, um die Funktionalität zu verifizieren.

Hinweise:

- Die Schrittvariable soll den aktuell ausgeführten Zustand angeben. Diese Variable müssen Sie manuell setzen.
- Es muss darauf geachtet werden, dass Sensorsignale in der Steuerung nur gelesen werden können.
- Es empfiehlt sich das System in regelmäßigen Abständen auf Funktionalität der Teifunktionen zu testen.
- Zum Testen der Steuerung und für die finale Abgabe müssen Sie alle Ausgänge, sowie die Schrittvariable, in einem Scope visualisieren.

Verwenden Sie den Block „Clock“, um die aktuelle Ausführungsduer zu ermitteln. Benutzen Sie diese Information zur Implementierung von definierten Warteintervallen.

8.3 Aufgabe: Inbetriebnahme der Station (nicht für Online Praktikum)

Der erstellte Ablauf soll nun direkt zur Steuerung der echten Anlage verwendet werden. Bitte bearbeiten Sie die folgenden Punkte und melden sich dann bei einem Tutor:

- Stellen Sie im Block „Simulation pace“ sicher, dass die Steuerung in Echtzeit ausgeführt wird.
- Stellen Sie durch Doppelklick auf die Schalter „Quelle für Sensoren“ und „Ausgang für Akteuren“ der Vorlage eine Verbindung zwischen realer Anlage und Steuerung her. Sie überbrücken somit die Simulation der Anlage und ersetzen diese durch die echte Anlage.
- Die Blöcke „Read“ und „Write“ sind auskommentiert, um Fehler bei der Erstellung der Simulation und der Steuerung zu vermeiden, solange keine Verbindung zur realen Anlage besteht. Diese müssen nun wieder aktiviert werden. Drücken Sie dazu die Tastenkombination „Strg + Shift + X“ oder wählen Sie im Rechtsklick-Menü der Blöcke die Option „uncomment“.
- Speichern Sie das Modell auf Ihrem Netzwerklaufwerk.
- Melden Sie sich bei einem Tutor und wechseln Sie auf dessen Anweisung hin den PC.

Sebald Sie sich am PC für die Inbetriebnahme der Anlage befinden, führen Sie folgendes durch:

- Melden Sie sich mit Ihrer Kennung an.
- Öffnen Sie das gespeicherte Modell vom Netzwerklaufwerk.
- Starten Sie auf dem Desktop das Programm „Communicator PLC - Festo“. Dieses stellt die Kommunikation zwischen Matlab/Simulink und der Anlage her.
- Legen Sie folgende Zylinder in dieser Reihenfolge in das Magazin der ersten Station ein:
1. Schwarz mit Deckel, 2. Rot mit Deckel, 3. Rot ohne Deckel
- Starten Sie die Ausführung der Steuerung in Simulink.
- Informieren Sie einen Tutor. Dieser startet nun die Steuerungen der anderen Stationen.

Die erste Station liefert nun die Zylinder an Ihre Station. Ihre Anlagensteuerung sollte nun die Zylinder korrekt erkennen und auf Höhe/Farbe prüfen. Auf Basis dieser Eigenschaften wird das Teil dann entweder aussortiert oder an die Folgestation geliefert, welche diese sortiert.

Die Aufgabe gilt als erfolgreich abgeschlossen, wenn die Zylinder korrekt durch die Station „Prüfen“ ge laufen sind.

Die Inbetriebnahme der Station kann in dem Online Praktikum nicht durchgeführt werden und kann übersprungen werden!



Praktikum Simulationstechnik



Versuch 3



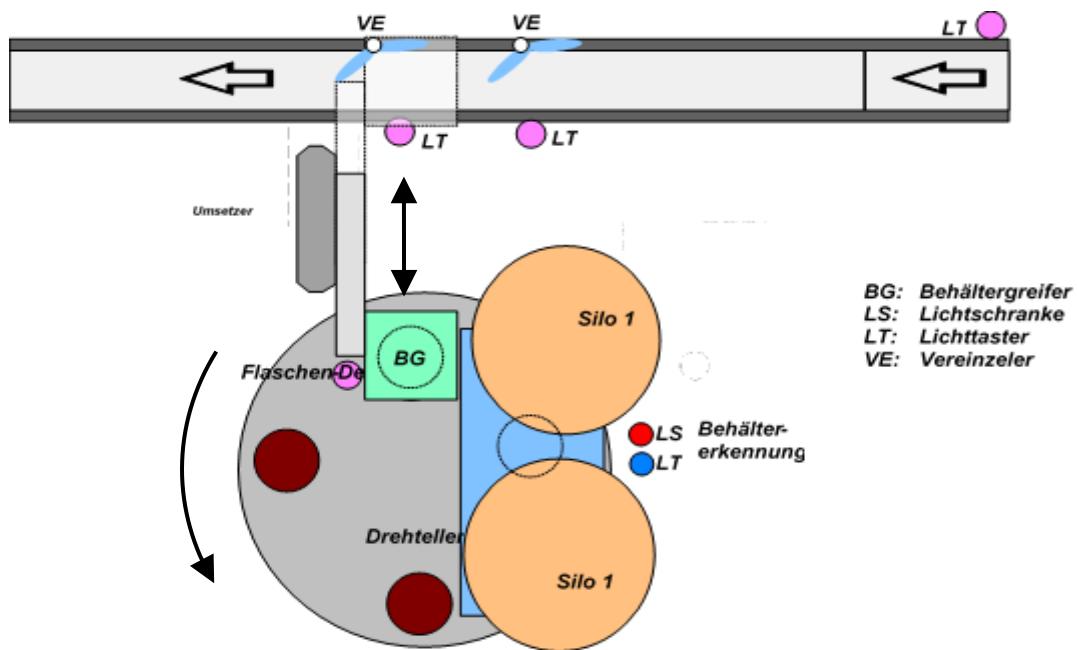
INHALTSVERZEICHNIS

9 GESAMTANLAGE	85
9.1 AUFBAU	85
9.2 FUNKTIONSWEISE	85
9.3 BAHNKOORDINATEN	86
9.4 GROBSTRUKTUR DER GESAMTANLAGE	87
10 MODELLIEREN DER MECHANIKKOMPONENTEN DER ANLAGE	88
10.1 AUFGABE	88
10.2 DIE TEILSYSTEME	89
10.2.1 ZUFUHRBAND	89
10.2.2 ARBEITSBAND	90
10.2.3 VEREINZELER 1	91
10.2.4 UMSETZER & GREIFER	92
10.2.5 DREHTELLER	93
10.2.6 VEREINZELER 2	93
10.2.7 SILOS UND ABFÜLLSTATION	94
11 SIMULATION DES PROZESSGUTS	95
11.1 FLASCHEN_DATA	95
11.2 AUFGABEN	98
11.2.1 FLASCHENANLAGENGESCHWINDIGKEITEN	98
11.2.2 ANLAGENPOSITIONSBESTIMMUNG	99
11.2.3 FLASCHENGESCHWINDIGKEITSBESTIMMUNG	99
11.2.4 FLASCHENABSTANDSÜBERWACHUNG	100
11.2.5 INTEGRATION DER EINZELNEN KOMPONENTEN	101

9 Gesamtanlage

9.1 Aufbau

Die folgende Abbildung zeigt den schematischen Aufbau der Flaschenabfüllanlage.



9.2 Funktionsweise

Der Funktionsweise der Anlage kann durch die folgenden Schritte beschrieben werden. Ein Video finden Sie im Vorlagenverzeichnis.

- Die Flaschen kommen vom Zufuhrband auf das Arbeitsband. Dort werden sie am ersten Vereinzeler aufgehalten.
- Wenn die Steuerung es erlaubt, darf eine Flasche den ersten Vereinzeler passieren. Danach wird sie vom zweiten Vereinzeler aufgehalten.
- Der Umsetzer greift die Flasche und hebt sie auf den Drehteller. Dieser dreht sich so lange **gegen den Uhrzeigersinn**, bis die Flasche unter der Siloöffnung steht. Dann öffnen sich die Silos und die Flasche wird gefüllt. Nach dem Schließen der Silos wird die Flasche wieder zum Umsetzer transportiert. Der Umsetzer greift die Flasche und transportiert diese zurück auf das Arbeitsband. Danach öffnet der zweite Vereinzeler und die Flasche läuft auf dem Abfuhrband weiter.



9.3 Bahnkoordinaten

Um die Anlage aussagekräftig simulieren zu können muss die Position jeder Flasche bekannt sein. Hierzu werden sogenannte *Bahnkoordinaten* definiert. Diese Bahnkoordinaten beschreiben die Strecke, die die Flaschen bereits in der Anlage zurückgelegt haben.

Jede Flasche durchläuft folgende Stationen. Dabei sind die Bahnkoordinaten wie folgt festgelegt:

Tabelle 10: Anlagenpositionen, Bahnkoordinaten und zugehörige Längen der Anlage

Station	Anlagenposition	Bahnkoordinaten	Länge in Bahnkoordinaten
Noch nicht vorhanden	1	0	0
Zufuhrband	2]0;101[100
Arbeitsband	3	[101;201[100
Vereinzeler1	4	[201;205[4
Arbeitsband	5	[205;305[100
Umsetzer	6	[305;405[100
Drehteller	7	[405;505[100
Umsetzer	8	[505;605[100
Vereinzeler2	9	[605;610[5
Abfuhrband	10	[610; ∞ [∞

9.4 Grobstruktur der Gesamtanlage

Die Gesamtsimulation der Flaschenabfüllanlage besteht aus dem **mechanischen Teil** (Nachbildung der Mechanik, Sensorik und Aktorik), dem **Flaschenmodul** (Nachbildung der Flaschen in der Anlage) und dem **Steuerungsteil**. Der mechanische Teil und das Flaschenmodul werden in Simulink modelliert und der Steuerungsteil in Stateflow. Abbildung 62 zeigt den schematischen Aufbau der Gesamtanlage

- Der Steuerungsteil empfängt von der **mechanischen Komponente** (Sensoren) den Status der Anlage und schaltet abhängig davon die **Aktoren (Motoren)** der mechanischen Komponente ein bzw. aus.
- Die mechanische Komponente übergibt dem Flaschenmodul die **Geschwindigkeiten** der einzelnen Anlagenteile (Bänder, Drehteller etc.), also die Information, welcher Teil der Anlage sich zum aktuellen Zeitpunkt mit welcher Geschwindigkeit bewegt.
- Das Flaschenmodul, das die Position der einzelnen Flaschen in der Anlage verwaltet, bestimmt aus diesen Geschwindigkeiten die aktuellen Positionen der einzelnen Flaschen und gibt diese wieder an die mechanische Komponente zurück.

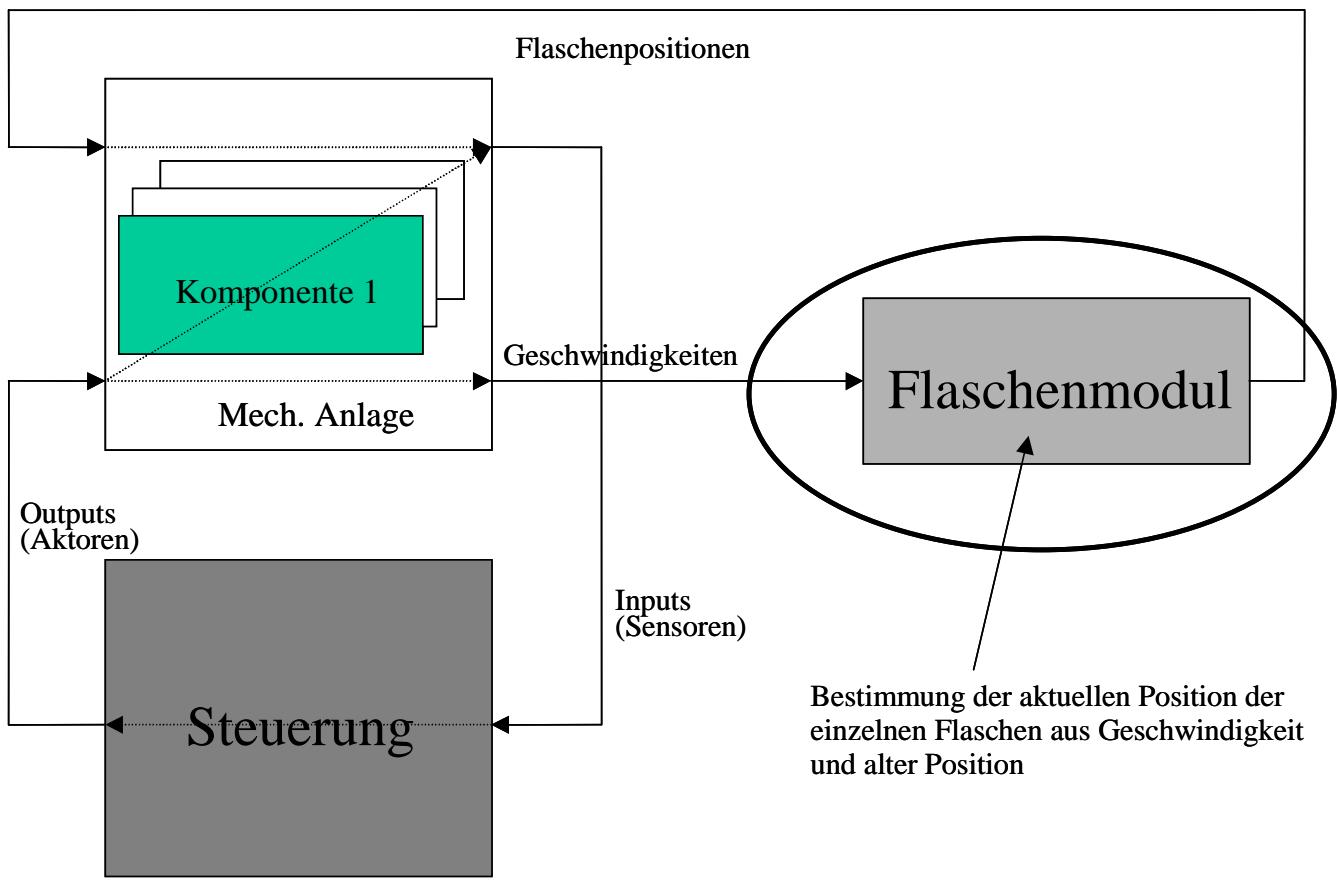


Abbildung 62: Grobstruktur der Gesamtanlage



10 Modellieren der Mechanikkomponenten der Anlage

In diesem Versuch wird die Flaschenabfüllanlage modelliert. Der Nachweis der Funktionsfähigkeit der einzelnen Komponenten und die Beachtung der Hinweise sind die Grundlagen für die Bearbeitung der Aufgabe.

10.1 Aufgabe

Modellieren Sie alle Teilsysteme aus dem nachfolgenden Kapitel 10.

Wichtig: Komponenten werden nur abgenommen, wenn die Funktionsweise mit geeigneten Eingangs- und Ausgangssignalen verifiziert wurde. Verwenden Sie dazu beispielsweise den Simulink-Block *Signal Builder* als Eingangssignal und Simulink-Block *Scope* zur Auswertung

Eine Aufgabe gilt erst dann als gelöst, wenn Sie die Funktionsfähigkeit ihres Programmes NACHWEISEN und ERKLÄREN können. Benennen Sie hierzu alle relevanten Signale und Blöcke. Betrachten Sie die Ein- und Ausgangssignale in EINEM Scope, in dem alle Signale benannt sind. Überlegen Sie sich sinnvolle Testsignale, um die Funktionalität zu verifizieren.

Tipp:

Überlegen Sie sich im Voraus, welche Teile ihrer Modelle Sie wieder verwenden könnten, d.h. ob es Aspekte gibt, die in mehreren Komponenten modelliert werden müssen. Entwickeln Sie für diese Aspekte eigene Subsysteme, und verwenden Sie Teile ihrer Modelle wieder.



10.2 Die Teilsysteme

Die Anlage besteht aus folgenden Teilsystemen:

- Zuführband
- Arbeitsband
- Vereinzeler1
- Umsetzer
- Drehteller
- Abfüllstation
- Vereinzeler2

10.2.1 Zuführband

Das Zuführband ist immer in Betrieb, wenn die Anlage an ist und bringt die Flaschen zur Anlage.

Aufbau

Das Band besitzt einen Elektromotor, der das Band direkt antreibt. Am Band ist ein Sensor angebracht (Lichttaster), der die vorbeilaufenden Flaschen detektiert.

Ein- und Ausgangsgrößen

Eingänge:	
o_band_zufuhr	Schaltet ein Relais, das den Elektromotor mit Strom versorgt
Flaschenpositionen	Vektor mit 25 Elementen der alle Flaschenpositionen (in Bahnkoordinaten) verwaltet
Ausgänge:	
i_zufuhr_LT	Gibt eine logische 1, wenn sich eine Flasche vor dem Lichttaster, d.h. in den Bahnkoordinaten]0;4[befindet.
v_band_zufuhr	Gibt die aktuelle Bandgeschwindigkeit in Bahnkoordinatenweg / Sekunde aus

Physikalische Randbedingungen

- Da das Band von Beginn an laufen soll, sind alle **Anlaufvorgänge zu vernachlässigen**. Geschwindigkeitsänderungen durch Last werden ebenfalls vernachlässigt.
- Vom **Anfang bis zum Ende** des Bandes benötigt eine Flasche **10s**.

Tipps

- i_{zufuhr_LT} ist ein Skalar
- Zur Berechnung von Skalarwerten aus Vektoren existieren verschiedene Möglichkeiten
- Logische Komparatoren lassen sich auch auf Vektoren anwenden:

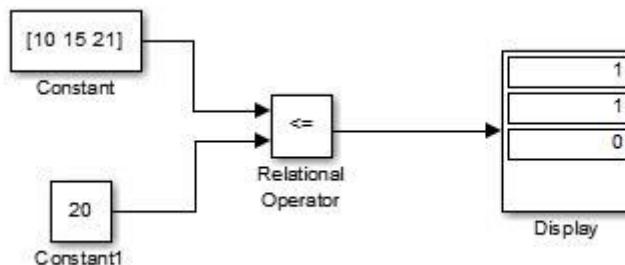


Abbildung 63: Arbeitsweise von logischen Komparatoren

- Logische Signale („TRUE“ = 1) lassen sich zusammenzählen.
- Verwenden Sie die Tabelle 10 auf Seite 86 mit den Bahnkoordinaten.

10.2.2 Arbeitsband

Das Arbeitsband ist wie das Zufuhrband aufgebaut. Es wird nur vom Vereinzeler1 unterbrochen.

Ein- und Ausgangsgrößen

Eingänge:	
o_band_arbeit	Schaltet ein Relais, das den Elektromotor mit Strom versorgt
Ausgänge:	
v_band_arbeit	Gibt die aktuelle Bandgeschwindigkeit in Bahnkoordinatenweg / Sekunde aus

Physikalische Randbedingungen

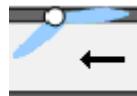
- Anlaufvorgänge können vernachlässigt werden!
- Eine Flasche benötigt vom **Beginn des Arbeitsbandes bis zum Vereinzeler 1** $10/3\text{ s}$

10.2.3 Vereinzeler 1

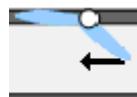
Der Vereinzeler wird pneumatisch geöffnet und durch **eine Feder** in die Ausgangsstellung zurückgebracht.

Die Zustände Geschlossen/Offen des Vereinzeler beziehen sich auf das „Schiebelement“:

Geschlossen / Flasche aufhaltend



Offen / eine Flasche durchlassend



Die Pfeile stellen hierbei die Bewegungsrichtung des Förderbandes dar.

Hinweise

- Der Vereinzeler lässt bei jeden Öffnungs-/Schließvorgang jeweils nur eine Flasche durch.
- Skizzieren Sie sich zunächst das **Kräftegleichgewicht** am Vereinzeler und setzen Sie dieses dann in einem Simulink-Modell um.
- Eine fehlerhafte Ansteuerung (z.B. Aktor nur für 0,1 s eingeschaltet) soll abgebildet werden können.

Ein- und Ausgangsgrößen

Eingänge:	
o_ve1	Öffnet das Pneumatikventil
Flaschenpositionen	Vektor von 25 Elementen der alle Flaschenpositionen verwaltet
Ausgänge:	
i_ve1_LT	Ist 1, wenn sich Flasche in den Bahnkoordinaten]199,5;202,5[befindet
i_ve1_geschlossen	Ist 1, wenn Vereinzeler 1 sich in Ausgangsstellung (geschlossen) befindet, sonst 0
i_ve1_offen	Ist 1, wenn Vereinzeler 1 vollständig geöffnet ist, sonst 0
v_ve1	Gibt die aktuelle Öffnungsgeschwindigkeit in Bahnkoordinatenweg / Sekunde aus

Physikalische Randbedingungen

- Der Vereinzeler besitzt jeweils einen mechanischen Endanschlag in beide Richtungen, über den nicht gefahren werden kann, egal wie lange Druck anliegt. Beachten Sie darum, dass er sich nicht bewegt, wenn er am Anschlag steht → **der Vereinzeler kann eine positive oder negative Geschwindigkeit haben oder stehen ($v=0$)!**
- Anlaufvorgänge können auch hier vernachlässigt werden.
- Für eine Bewegung** von einem zum anderen Anschlag benötigt er **0,25s**.
- Die Ausgangsstellung ist geschlossen.**
- i_ve1_LT ergibt sich aus der Sensorposition, die von Tabelle 10 etwas abweichen kann



10.2.4 Umsetzer & Greifer

Der Umsetzer wird ebenfalls pneumatisch angetrieben. Er verfügt über einen Greifer zum Greifen der Flaschen. Diesen Greifer kann der Umsetzer auf einer vorgegebenen Bahn bewegen.

Ein- und Ausgangsgrößen

Eingänge:	
o_umsetz_band	Öffnet das Ventil, das den Umsetzer in Richtung Band bewegt
o_umsetz_teller	Öffnet das Ventil, das den Umsetzer in Richtung Drehteller bewegt
o_greifer_auf	Öffnet das Ventil, das den Greifer öffnet
o_greifer_zu	Öffnet das Ventil, das den Greifer schließt
Ausgänge:	
i_umsetz_band	Ist 1, wenn Umsetzer am Band, sonst 0
i_umsetz_teller	Ist 1, wenn Umsetzer am Teller, sonst 0
i_umsetz_greifer_auf	Ist 1, wenn Greifer vollständig geöffnet
i_umsetz_greifer_zu	Ist 1, wenn Greifer vollständig geschlossen
v_umsetzer	Gibt die aktuelle Umsetzergeschwindigkeit (nicht Greifer) in Bahnkoordinatenweg / Sekunde aus. Der Umsetzer verfügt über eine positive Geschwindigkeit, wenn er sich vom Teller zum Band bewegt.

Physikalische Randbedingungen

- Auch hier können wieder alle Anlaufvorgänge vernachlässigt werden.
- Beachten sie auch hier, dass jeweils der Greifer und die Bahn Endanschläge besitzen!
- Das vollständige **Öffnen oder Schließen** des Greifers benötigt **0,25s**.
- Die zurückzulegende **Strecke des Umsetzers ist 100**.
- **Von einem Ende** der Bahn **zum anderen** benötigt der **Umsetzer 1,25s**.
- Zu Beginn steht der **Umsetzer am Teller** und der **Greifer ist geschlossen**.
- Der Greifer kann **nicht gleichzeitig offen und geschlossen** sein



10.2.5 Drehsteller

Wurde bereits am ersten Tag im Versuch 4.3.3 (Seite 36) modelliert!

Fügen Sie außerdem den am ersten Tag in Versuch 5.2.4 (Seite 56) modellierten Winkelsensor hinzu und passen Sie die Ein- und Ausgänge entsprechend der Tabelle an.

Ein- und Ausgangsgrößen

Eingänge:	
o_teller	Schaltsignal für den Drehtellermotor
Flaschenpositionen	Vektor von 25 Elementen der alle Flaschenpositionen verwaltet
Ausgänge:	
i_teller_ausger	Ist 1, wenn der Teller ausgerichtet ist (siehe Kapitel 5.2.4 auf Seite 58).
i_teller_behaelter	Ist 1, wenn sich die Flasche bei den Bahnkoordinaten]404,8;405,2[bzw.]504,8;505,2[befindet.
v_teller	Geschwindigkeit des Tellers in Bahnkoordinaten / Sekunde

Hinweis

- i_teller_behaelter ergibt sich aus der Sensorposition, die von Tabelle 10 etwas abweichen kann

10.2.6 Vereinzeler 2

Die Funktionsweise des Vereinzeler 2 ist identisch zu der vom Vereinzeler 1.

Ein- und Ausgangsgrößen

Eingänge:	
o_ve2	Öffnet das Ventil
Flaschenpositionen	Vektor von 25 Elementen der alle Flaschenpositionen verwaltet
Ausgänge:	
i_ve2_LT	Ist 1, wenn sich die Flasche bei den Bahnkoordinaten]304,8;305,2[bzw.]604,8;605,2[befindet.
i_ve2_geschlossen	Ist 1, wenn Vereinzeler2 sich in Ausgangsstellung befindet, sonst 0
i_ve2_offen	Ist 1, wenn Vereinzeler2 vollständig geöffnet ist, sonst 0
v_ve2	Gibt die aktuelle Öffnungsgeschwindigkeit in Bahnkoordinatenweg / Sekunde aus

Physikalische Randbedingungen

Es gelten dieselben Randbedingungen wie für den Vereinzeler 1.



10.2.7 Silos und Abfüllstation

Die Abfüllstation besteht aus zwei Silos und zwei Ventilen, die das Granulat bei eingeschalteter Spannung hindurch lassen. Das Granulat fließt anschließend zusammen und in die Flaschen.

Zur Flaschenerkennung gibt es eine Lichtschranke und einen Lichttaster. Das Signal der oberen Lichtschranke tritt ungehindert durch das Glas hindurch, aber nicht durch den Abfüllstoff. So kann detektiert werden, ob die Flasche schon voll ist.

Ein- und Ausgangsgrößen

Eingänge:	
o_silo1	Öffnet das Ventil des 1. Silos
o_silo2	Öffnet das Ventil des 2. Silos
Silo1_gefüllt	Bei einer rising edge wurde das Silo1 vollständig gefüllt
Silo2_gefüllt	Bei einer rising edge wurde das Silo2 vollständig gefüllt
Flaschenpositionen	Vektor von 25 mit allen Flaschenpositionen
Ausgänge:	
i_silo1_leer	Ist 1, wenn das Silo1 leer ist, sonst 0
i_silo2_leer	Ist 1, wenn das Silo2 leer ist, sonst 0
i_BEKenn_LT_unten	Ist 1, wenn Flasche in den Bahnkoordinaten]477,5;482,5[, sonst 0
i_BEKenn_LS_oben	Ist 0, wenn Flasche in den Bahnkoordinaten]477,5;482,5[und die Flasche voll ist, sonst 1

Physikalische Randbedingungen

- Wenn das Ventil geöffnet wird, fällt das Granulat sofort in die Flasche (keine zeitliche Verzögerung).
- Das Granulat fließt mit **einem konstanten Volumenstrom** durch das Ventil (Volumenstrom ist zeitlich konstant).
- Das **Volumen einer Flasche** beträgt **10**.
- Der **Volumenstrom** durch ein Ventil beträgt **1**.
- Das **Volumen der Silos** beträgt lediglich **10**. (Das ist in der Tat sehr wenig, so kann aber ohne langes Warten getestet werden, ob das Nachfüllen der Silos funktioniert).

Tipps

- Ermitteln Sie die **Durchflussmenge für jedes Silo einzeln** (Füllgradermittlung der Silos) und addieren sie diese zur Durchflussmenge in die Flasche (Füllgradermittlung der Flasche).
- Überlegen Sie sich eine geeignete Möglichkeit für **jede leere Flasche die Füllgradermittlung neu zu beginnen**.
- Die Silos können nur abfüllen, solange sie nicht leer sind. Sie können jedoch noch weiter abfüllen, wenn die Flasche voll ist (Die Flasche läuft dann über, dies zu verhindern ist allerdings Aufgabe der Steuerung und an dieser Stelle nicht vorzusehen).

11 Simulation des Prozessguts

In diesem Versuch ist die Komponente Flaschenmodul der Flaschenabfüllanlage zu modellieren. **Beachten Sie auch die Hinweise in der Aufgabenstellung auf Seite 98!**

11.1 Flaschen_data

Die Komponente Flaschen_data ist der Hauptblock des Flaschenmoduls. In Flaschen_data werden die Positionen der einzelnen Flaschen der Gesamtanlage gespeichert und die aktuellen Positionen der Flaschen berechnet. Die einzelnen Bahnkoordinaten finden Sie in Tabelle 10 unter 1.3 Bahnkoordinaten

Eingänge:

Als Eingang werden dem Modul Flaschen_data die Geschwindigkeiten der Anlagenteile (Vektor mit 10 Elementen für die einzelnen Anlagenteile → Bahnkoordinaten) und die Information, ob der Greifer des Umsetzers geschlossen ist, übergeben. Zusätzlich hat Flaschen_data einen Eingang, über den „neue Flaschen“, in Form von „steigenden Flanken“, zu der Simulation hinzugefügt werden können.

Ausgang:

Der Ausgang von Flaschen_data ist ein Vektor, der die aktuellen Positionen der einzelnen Flaschen in der Anlage beinhaltet (Vektor mit 25 Elementen für max. 25 Flaschen in Bahnkoordinaten).

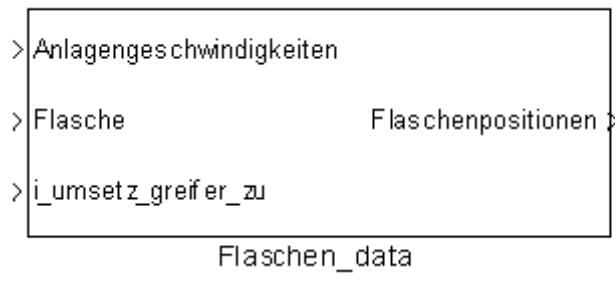


Abbildung 64: Flaschen_data

Flaschen_data besteht aus den zwei Hauptkomponenten Flaschengeschwindigkeiten und Flaschenstarter (Flaschengenerator) (siehe Abbildung 65).

Das Modul Flaschengeschwindigkeiten bekommt als Eingänge die Geschwindigkeiten der Anlagenteile (Anlagengeschwindigkeiten: Vektor mit 10 Elementen für die einzelnen Anlagenteile) und die Information, ob der Greifer des Umsetzers geschlossen ist (*i_umsetz_greifer_zu*), übergeben.

Außerdem werden dem Modul Flaschengeschwindigkeiten die zuletzt aktuellen Flaschenpositionen (Vektor mit 25 Elementen für die max. 25 Flaschen) übergeben. Aus diesen 3 Eingängen berechnet das Modul Flaschengeschwindigkeiten nun die aktuellen Flaschengeschwindigkeiten (Vektor mit 25 Elementen).

Das Ziel des Moduls Flaschenstarter ist, in den Vektor mit den aktuellen Flaschenpositionen neu hinzugekommene Flaschen zu integrieren, d.h. wenn beim Eingang Flasche (siehe Abbildung 65) eine steigende Flanke anliegt (0→1), dann wird dem Vektor Flaschenposition eine neue Flasche hinzugefügt (d.h. das jeweilige Element im Vektor wird von 0 auf 1 gesetzt, und somit angezeigt, dass sich eine neue Flasche auf dem Zuführband befindet; siehe Versuch 3, Seite 86, Tabelle 10 (1.3 Bahnkoordinaten)).

Der Ausgang von Flaschenstarter ist der Flaschenpositionsvektor mit der bereits integrierten neuen Flasche.

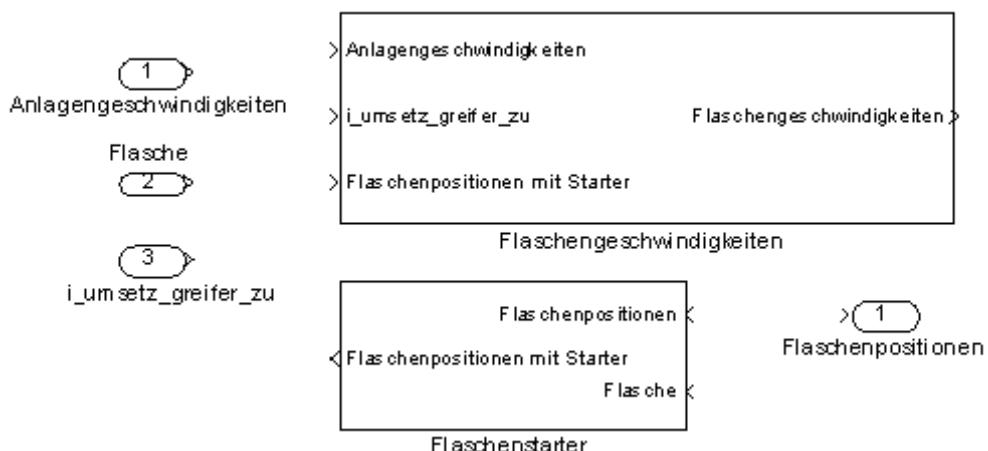


Abbildung 65: Hauptbestandteile von Flaschen_data

Wichtig: Ihre Aufgabe in diesem Praktikumsversuch ist es, das Modul Flaschengeschwindigkeiten zu modellieren, und mit dem bereits vorgegebenen Modul Flaschenstarter die Komponenten Flaschen_data zu vervollständigen.

Der Nachweis der Funktionsfähigkeit ist auch hier zu beachten!

Die Aufgabenstellung befindet sich am Ende des Kapitels zusammen mit Hinweisen und Tipps für die Implementierung.

Im Folgenden wird das Modul Flaschengeschwindigkeiten gegliedert und spezifiziert.

Aufbau von Flaschengeschwindigkeiten

Das Modul Flaschengeschwindigkeiten berechnet aus den alten Flaschenpositionen und den Anlagengeschwindigkeiten die aktuellen Flaschengeschwindigkeiten. Hierbei wird jeder Flasche abhängig von ihrer Position eine Anlagengeschwindigkeit als Flaschengeschwindigkeit zugeordnet.

Um aus den Anlagengeschwindigkeiten und den alten Flaschenpositionen die neuen Flaschengeschwindigkeiten bestimmen zu können, sind folgende Teilschritte notwendig:

1. Flaschenanlagengeschwindigkeit:

Die von der mechanischen Komponente (siehe Abbildung 64) erhaltenen Anlagengeschwindigkeiten können für die Anlagenteile Vereinzelner und Umsetzer negativ sein, da sich diese Komponenten in 2 Richtungen bewegen können. Die Flaschen auf dem Band bewegen sich jedoch bezüglich der eingeführten Bahnkoordinaten **nur vorwärts**.

Das Modul Flaschenanlagengeschwindigkeit setzt deshalb die Anlagengeschwindigkeiten in die zugehörigen (positiven) Geschwindigkeiten um, mit denen sich die Flaschen auf dem jeweiligen Anlagensegment (Bahnkoordinaten) bewegen. Hierfür werden negative Geschwindigkeiten, je nach Komponente, kompensiert oder zu positiven Geschwindigkeiten gemacht.

Zusätzlich wird bei der Geschwindigkeit einer Flasche im Umsetzer berücksichtigt, ob der Greifer geschlossen ist oder nicht (siehe Abbildung 65). Außerdem, muss berücksichtigt werden, dass der Greifer in der Simulation zweimal vorkommt (Anlageteil sechs und acht), in der Realität aber nur einmal.

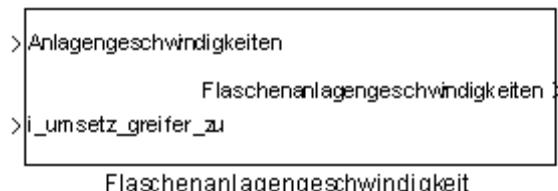


Abbildung 66: Modul Flaschenanlagengeschwindigkeit

2. Anlagenpositionsbestimmung:

Um den einzelnen Flaschen eine Geschwindigkeit zuweisen zu können, ist zu bestimmen, in welchem Segment der Anlage sich die Flaschen befinden. Dem Modul Anlagenpositionsbestimmung werden dazu die Flaschenpositionen in Bahnkoordinaten (Vektor) übergeben.

Nach Zuordnung in welchem Anlagenteil sich die einzelnen Flaschen befinden, wird ein Vektor mit Anlagenpositionen der einzelnen Flaschen zurückgegeben (siehe Versuch 3, Seite 86, Tabelle 10 (1.3 Bahnkoordinaten)).



Abbildung 67: Modul Anlagenpositionsbestimmung

3. Flaschengeschwindigkeitsbestimmung:

Mittels der Flaschenanlagengeschwindigkeiten (Die Geschwindigkeiten, mit der eine Flasche im entsprechenden Anlagenteil bewegt wird, z. B. Geschwindigkeit eines Bandes) und den Anlagenpositionen der einzelnen Flaschen kann im Modul Flaschengeschwindigkeitsbestimmung die Flaschengeschwindigkeit für jede Flasche bestimmt werden.



Flaschengeschwindigkeitsbestimmung

Abbildung 68: Modul Flaschengeschwindigkeitsbestimmung

4. Flaschenabstandsüberwachung:

Durch das Modul Flaschenabstandsüberwachung wird verhindert, dass zwei Flaschen kollidieren. Dieses Modul sorgt dafür, dass die Flaschen stets einen Mindestabstand (**in Bahnkoordinaten ≥ 6**) zueinander haben.

Der Eingang des Moduls sind die Flaschenpositionen (Vektor mit Position der Flaschen in Bahnkoordinaten).

Der Ausgang des Moduls ist ein Vektor (mit 25 Elementen; BOOL), der für jede Flasche angibt, ob sich die Flasche bewegen darf oder nicht.



Flaschenabstandsüberwachung

Abbildung 69: Modul Flaschenabstandsüberwachung



11.2 Aufgaben

Ihre Aufgabe ist es im Folgenden, die einzelnen Teilkomponenten des Moduls Flaschengeschwindigkeiten zu modellieren und zusammenzusetzen. Anschließend werden Sie die Komponente Flaschen_data vervollständigen.

Wichtig: Verwenden Sie die Vorlage, die Ihnen auf dem Verlagenlaufwerk (V:\) Moodle zur Verfügung gestellt wird.

Eine Aufgabe gilt erst dann als gelöst, wenn Sie die Funktionsfähigkeit ihres Programmes NACHWEISEN und ERKLÄREN können. Benennen Sie hierzu alle relevanten Signale und Blöcke. Betrachten Sie die Ein- und Ausgangssignale in EINEM Scope, in dem alle Signale benannt sind. Überlegen Sie sich sinnvolle Testsignale, um die Funktionalität zu verifizieren.

11.2.1 Flaschenanlagengeschwindigkeiten

Das Modul filtert für die Anlage nicht zulässige Geschwindigkeiten und gibt die tatsächliche Geschwindigkeit aus.

Eingänge:	
Anlagengeschwindigkeiten	Vektor der 10 ungefilterten Anlagengeschwindigkeiten
i_umsetz_greifer_zu	Booleanwert, der angibt, ob der Greifer des Umsetzers geschlossen ist
Ausgänge:	
Flaschenanlagengeschwindigkeiten	Vektor mit 10 Elementen, der die korrigierten Geschwindigkeiten in den jeweiligen Anlagenabschnitten enthält

Tipps und Hinweise:

- Flaschen werden im Vereinzeler nur vorwärts transportiert
- Der Umsetzer kann nur Flaschen transportieren, wenn der Greifer geschlossen ist
- Die Geschwindigkeit des Umsetzers in Richtung Teller wird von der mech. Komponente als negativer Wert, und vom Teller in Richtung Fließband als positiver Wert übergeben.
- Der Umsetzer ist physikalisch nur einmal, in der Simulation aber mehrfach vorhanden. Trotzdem kann logischerweise immer nur maximal eine dieser Instanzen eine Flasche transportieren.
- $0,5 * (|u| \pm u)$ verwenden. Das Symbol \pm zeigt an, dass die Formel mit entweder + oder - implementiert werden kann. Durch ausprobieren von positiven bzw. negativen Eingangssignalen u wird die Auswirkung deutlich. Alternativ kann ein Totzonenglied verwendet werden.
- Multiplikation mit Bool-Werten mit anschließender Konvertierung möglich (TRUE = 1, FALSE = 0)
- Die Geschwindigkeit des Abfuhrbands entspricht der Geschwindigkeit des Arbeitsbands



11.2.2 Anlagenpositionsbestimmung

Das Modul weist einer bestimmten Flaschenposition einen Anlagenteil zu (siehe Seite 91).

Eingänge:	
Flaschenposition	Vektor mit 25 Elementen für die Bahnkoordinaten der Flaschen
Ausgänge:	
Anlagenpositionen	Vektor mit 25 Elementen, der den Flaschen den jeweiligen Anlagenabschnitt zuordnet

11.2.3 Flaschengeschwindigkeitsbestimmung

Eingänge:	
Flaschenanlagengeschwindigkeiten	Vektor mit 10 Elementen für die Flaschenpositionen
Anlagenpositionen	Vektor mit 25 Elementen, der den Flaschen den jeweiligen Anlagenabschnitt zuordnet
Ausgänge:	
Flaschengeschwindigkeiten	Vektor mit 25 Elementen, der die aktuellen Geschwindigkeiten der Flaschen enthält

Bestimmen Sie nun aus den Flaschenanlagengeschwindigkeiten (0...10) und den Flaschenpositionen die Flaschengeschwindigkeiten.

Tipps und Hinweise:

- Verwenden Sie den Selector-Block (vgl. Seite 27). Der Selector weist einem Vektor mit Indizes die zugehörigen Werte aus einem anderen Vektor zu. Hierfür muss ein Eingang als "Index Vector (port)" definiert sein. Beachten Sie zusätzlich die „Input Port size“.

Es folgt ein vereinfachtes Beispiel mit Vektoren kleinerer Größe:

$$\text{Flaschenanlagengeschwindigkeiten} = \begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix}$$

$$\text{Anlagenpositionen} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 2 \end{bmatrix}$$

$$\Rightarrow \text{Flaschengeschwindigkeiten} = \begin{bmatrix} 10 \\ 20 \\ 30 \\ 20 \end{bmatrix}$$

Mit dem Selector kann man somit eine Reihe Indizes (in ein Array) mit dem Array verknüpfen.

```
for(int i=0;i<numIndices;i++)  
    ausgang[i] = array[index[i]];
```



11.2.4 Flaschenabstandsüberwachung

Eingänge:	
Flaschenposition	Vektor mit 25 Elementen für die Bahnkoordinaten der Flaschen
Ausgänge:	
Bewegung möglich	Vektor mit 25 Booleanwerten

Abschließend ist ein Modul zu entwickeln, das den Abstand zwischen den Flaschen überwacht und gegebenenfalls die Geschwindigkeit der nachfolgenden Flasche auf *null* setzt. Dazu wird ein Block benötigt, der als Eingangsgröße die Flaschenpositionen (in Bahnkoordinaten) und als Ausganggröße einen boolschen Vektor der Größe 25 aufweist. Dieser Vektor hinterlegt für jede Flasche, ob sie sich bewegen darf, oder ihre Bewegung aufgrund geometrischer Unverträglichkeit (eine andere Flasche befindet sich bereits an der Zielstelle) nicht möglich ist.

Tipps und Hinweise:

- Verwenden Sie den Simulink Block *MATLAB Fcn* (dessen Name kürzt "MATLAB function" ab), um die MATLAB-Funktion *circshift* anzuwenden und somit den Abstand zu berechnen.
- Das Eingangssignal des Blocks *MATLAB Fcn* wird mit *u* bezeichnet.

Anwendungsbeispiel *circshift*:

$$\text{circshift} \left(\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}, 1 \right) = \begin{pmatrix} u_3 \\ u_1 \\ u_2 \end{pmatrix}$$

- **Der Abstand zwischen den Flaschenmittelpunkten muss mindestens sechs (Flaschenradius ist drei) betragen!**
- Flaschenabstand = *circshift(Flaschenpositionen,1)* – Flaschenpositionen
- Erlauben Sie der ersten Flasche mittels des Blocks *Assignment* sich immer zu bewegen, da sie keine Vorgängerin hat und sie über die Funktion *circshift* mit der letzten Flasche verglichen wird.

11.2.5 Integration der einzelnen Komponenten

Integrieren Sie die Subsysteme zu dem Modul Flaschengeschwindigkeiten (Abbildung 70).

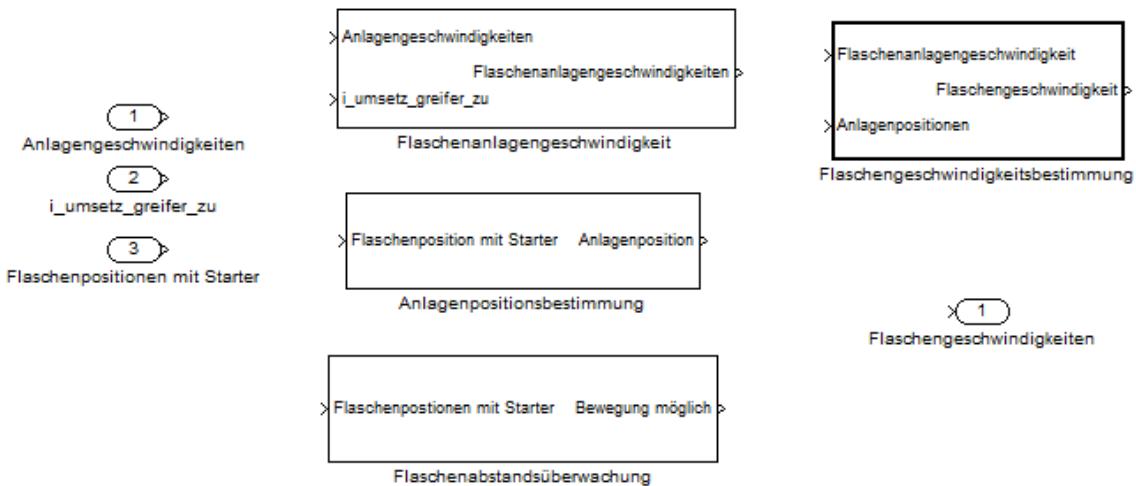


Abbildung 70: Inhalte von Flaschengeschwindigkeiten

Vervollständigen Sie nun die Komponente Flaschen_data gemäß Abbildung 65.



Praktikum Simulationstechnik



Versuch 4



Inhaltsverzeichnis

12 STEUERUNG DER ANLAGE	105
12.1 DAS PRINZIP DER STEUERUNG.....	105
12.2 BESCHREIBUNG DER STEUERUNG	106
12.3 SENSORSIGNAL: DEKLARATION DER VARIABLEN IM DATA DICTIONARY	107
12.4 AKTORSIGNAL: DEKLARATION DER VARIABLEN IM DATA DICTIONARY	108
12.5 AUFGABE: IMPLEMENTIERUNG DER STEUERUNG	109



12 Steuerung der Anlage

**Das Modell aus dem vorhergehenden Versuch stellt alle Sensorsignale der Anlage zur Verfügung und liegt als Vorlage im Vorlagenverzeichnis.
Verwenden Sie für die Steuerung nicht ihr eigenes Modell!**

Die Sensorsignale zeigen, ob sich in bestimmten Positionen der Anlage eine Flasche befindet. Zusätzlich wurden die Bewegungsgleichungen der Komponenten auch in Simulink implementiert. Die Bewegung einer Komponente hängt davon ab, ob der entsprechende Aktor ein- bzw. ausgeschaltet ist.

In diesem Versuch wird eine **Steuerung für die Abfüllstation in Stateflow implementiert**.

Die Aufgabe der Steuerung ist, die Aktoren (Motoren, Ventile, usw.) der Komponenten, abhängig von den Sensorsignalen, ein- bzw. auszuschalten, damit die Flaschen möglichst schnell mit dem Granulat aus den beiden Silos abgefüllt und zu den nächsten Stationen weitergeleitet werden. Die Aktoren werden durch Variablen angesprochen und müssen von der Steuerung entsprechend gesetzt bzw. zurückgesetzt werden.

12.1 Das Prinzip der Steuerung

Zur Implementierung der Steuerung müssen Sie beachten, dass **Sensorsignale nur abgefragt** und **Aktorsignale nur gesetzt** werden können. Man kann nicht abfragen, ob der Motor des Drehellers eingeschaltet ist, da kein Sensor zu diesem Zweck in der Anlage zur Verfügung steht.

Beispiel für Sensoren und Aktoren:

Im folgenden Beispiel wird das Vorgehen beim Abfragen eines Sensors bzw. beim Setzen eines Aktors erläutert.

Es gibt einen Sensor, der erkennt, ob sich der Umsetzer am Teller befindet und es gibt einen anderen Sensor, der erkennt, ob sich der Umsetzer am Band befindet. Es ist klar, dass beide Sensoren nie gleichzeitig 1 liefern dürfen. Wenn beide 0 liefern, bedeutet es, dass der Umsetzer gerade unterwegs ist. Es gibt zwei Pneumatikzylinder zum Antreiben des Umsetzers. Ein Zylinder bringt den Umsetzer vom Teller zum Band und der andere Zylinder bringt den Umsetzer vom Band zum Teller. Beide Motoren dürfen nicht gleichzeitig eingeschaltet werden und beide müssen zu null gesetzt werden, wenn der Umsetzer angehalten werden soll.

Die folgenden Variablen sprechen diese zwei Paare von Sensoren und Aktoren an:

Sensoren	Aktoren
i_umsetz_band	o_umsetz_band
i_umsetz_teller	o_umsetz_teller



12.2 Beschreibung der Steuerung

Die Komponenten der Anlage sind:

- Zufuhrband
- Arbeitsband
- Vereinzeler1
- Umsetzer mit Behältergreifer
- Drehteller
- Silo 1 und Silo 2
- Vereinzeler 2

Wichtige Hinweise zum Ablauf der Abfüllanlage:

- Sobald der Lichttaster am Anfang des Zufuhrbandes eine Flasche erkennt, führen die beiden Bänder die Flasche (es können auch mehrere sein) dem ersten Vereinzeler zu.
- Die Flaschen warten vor dem ersten Vereinzeler darauf, dass sie durchgelassen werden. Der Vereinzeler lässt eine Flasche nur durch, wenn im Bereich des Drehtellers noch Platz frei ist. Damit ist sichergestellt, dass stets Platz für eine befüllte Flasche vorhanden ist, die mit Hilfe des Umsetzers vom Drehteller auf das Band zurückgestellt wird.
- Der zweite Vereinzeler entlässt die befüllte, zum Band zurückgestellte Flasche in Richtung einer weiteren Station. Ein Näherungssensor am Drehteller erkennt, ob sich auf der Umsetzerposition auf dem Drehteller eine Flasche befindet oder ob der Umsetzer ihn mit einer neuen Flasche bestücken kann. Erst dann lässt der erste Vereinzeler eine leere Flasche an den Umsetzer weiter.
- Damit der Drehteller in der richtigen Position angehalten werden kann, ist ein Sensor angebracht, der die vier Positionen des Drehtellers erkennt. An jeder der vier Positionen des Drehtellers gibt es einen Platz für eine Flasche.
- Sobald der Sensor eine Flasche unter dem Silo detektiert, wird diese befüllt. Dafür werden beide Silos geöffnet, bis ein Sensorsignal andeutet, dass die Flasche voll ist. Danach **muss 2 Sekunden gewartet werden, da das Granulat noch etwas nachrieselt.**
- Während des Befüllens kann der Greifer eventuell eine schon abgefüllte Flasche wieder zurück auf das Band und gleich danach eine leere Flasche auf den Drehteller stellen. Erst danach dreht der Drehteller in die nächste Position weiter.



12.3 Sensorsignale: Deklaration der Variablen im Data Dictionary

Die Sensorsignale werden in Vektoren boolscher Variablen zusammengefasst, damit die Übersichtlichkeit des Modells eingehalten werden kann. Diese Vektoren müssen im **Data Dictionary** als **Data mit Scope „Input from Simulink“, Type „boolean“ und Size, die Anzahl der Komponenten**, deklariert werden (siehe auch Abbildung 71). Weiterhin kann der **First Index im Tab „Description“ auf 1 gesetzt werden**.

Komponenten	Sensorsignale (Simulink-Variablen)	Variablen (werden in Stateflow angesprochen)	Vektoren (Variablen im Data Dictionary)
Vereinzeler 1 & Vereinzeler 2	i_ve1_offen	i_ver einzeler(1)	i_ver einzeler
	i_ve1_geschlossen	i_ver einzeler(2)	
	i_ve2_offen	i_ver einzeler(3)	
	i_ve2_geschlossen	i_ver einzeler(4)	
	i_ve1_LT	i_ver einzeler(5)	
	i_ve2_LT	i_ver einzeler(6)	
Förderband 1 & Förderband 2	i_zufuhr_LT	i_band	i_band
Umsetzer & Greifer	i_umsetz_band	i_umsetzer(1)	i_umsetzer
	i_umsetz_teller	i_umsetzer(2)	
	i_umsetz_greifer_auf	i_umsetzer(3)	
	i_umsetz_greifer_zu	i_umsetzer(4)	
Drehsteller	i_teller_ausger	i_teller(1)	i_teller
	i_teller_behaelter	i_teller(2)	
Silo 1 & Silo 2	i_BEKenn_LS_oben	i_silo(1)	i_silo
	i_BEKenn_LT_unten	i_silo(2)	
	i_silo1_leer	i_silo(3)	
	i_silo2_leer	i_silo(4)	

Tabelle 11: Deklaration der Variablen in Stateflow: Sensorsignale

Die Abbildung 71 fasst die Variablen zusammen, die im Data Dictionary als „Input from Simulink“ deklariert werden müssen.

	Name	Scope	Port	Data Type	Size
[...]	i_band	Input	1	boolean	
[...]	i_silo	Input	2	boolean	5
[...]	i_teller	Input	3	boolean	2
[...]	i_umsetzer	Input	4	boolean	4
[...]	i_ver einzeler	Input	5	boolean	6

Abbildung 71: Variablen im Data Dictionary (Eingänge der Steuerung)

12.4 Aktorsignale: Deklaration der Variablen im Data Dictionary

Die Akteure der Anlage werden ebenfalls als Vektoren deklariert. Setzt man den Wert eines Akteurs auf 1, wird er eingeschaltet. Zum Ausschalten der Akteure werden die entsprechenden Variablen auf null zurückgesetzt. Tabelle 12 zeigt die Zuordnung von den Aktorsignalen zu den Stateflow Variablen. Die Vektoren müssen im Data Dictionary als Data mit Scope „Output to Simulink“, Type „double“ und Size, die Anzahl der Komponenten, deklariert werden

Komponenten	Aktorsignale (Simulink-Variablen)	Variablen (werden in Stateflow angesprochen)	Vektoren (Variablen im Data Dictionary)
Vereinzeler 1 & Vereinzeler 2	o_ve1	o_ver einzeler(1)	o_ver einzeler
	o_ve2	o_ver einzeler(2)	
Förderband 1 & Förderband 2	o_band_zufuhr	o_band(1)	o_band
	o_band_arbeit	o_band(2)	
Umsetzer & Greifer	o_umsetz_band	o_umsetzer(1)	o_umsetzer
	o_umsetz_teller	o_umsetzer(2)	
	o_umsetz_greifer_auf	o_umsetzer(3)	
	o_umsetz_greifer_zu	o_umsetzer(4)	
Drehsteller	o_teller	o_teller	o_teller
Silo 1 & Silo 2	o_silo1	o_silo(1)	o_silo
	o_silo2	o_silo(2)	

Tabelle 12: Deklaration der Variablen in Stateflow: Aktorsignale

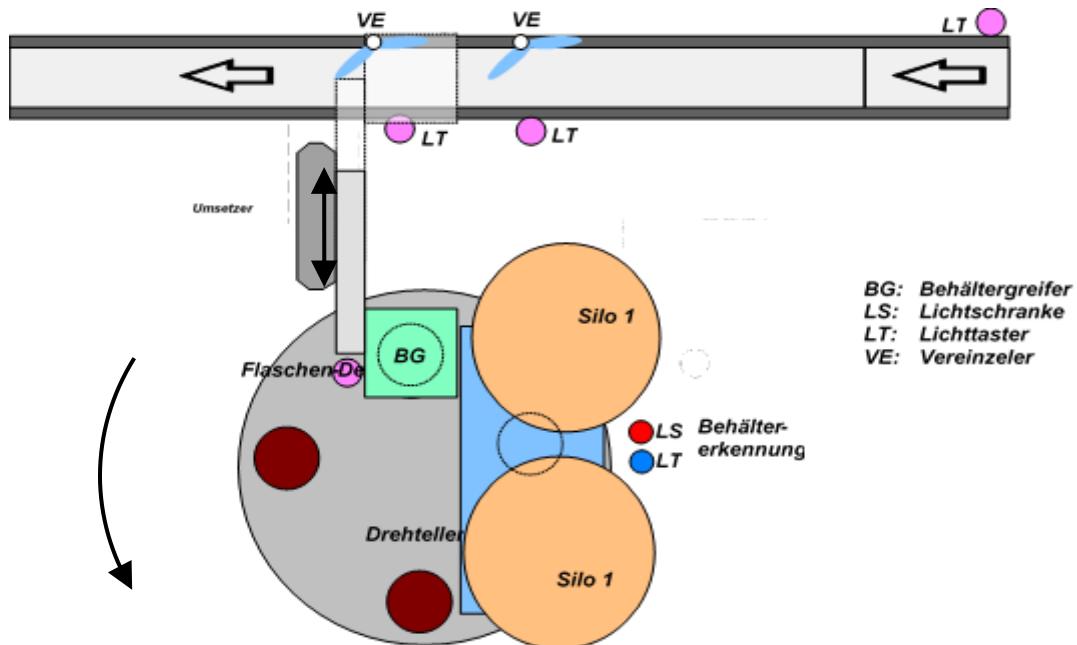


Abbildung 72: Schematische Darstellung der Anlage

12.5 Aufgabe: Implementierung der Steuerung

Erstellen Sie ein schriftliches Konzept und zeigen Sie es einem Tutor bevor Sie mit der Implementierung beginnen!

Überlegen Sie sich hierfür die Reihenfolge der Stationen welche eine Flasche auf ihrem Weg durch die Anlage passiert. (z.B. Flasche befüllen, Flasche am VE1 durchlassen, etc.)

Weiterhin sollten Sie entscheiden, ob Sie eine **zentrale Intelligenz** nutzen möchten, ob eine Komponente die Kontrolle über die anderen Komponenten übernimmt und deren Aktionen koordiniert oder ob Sie eine **verteilte Intelligenz (dezentral)** verwenden.

Es wird dringend empfohlen eine **zentrale Steuerung mit Intelligenz** zu implementieren, da dies Übersichtlichkeit und eine einfache Fehlersuche ermöglicht. Ein möglicher Aufbau der Steuerung ist in Abbildung 73 dargestellt. Die Erstellung solcher Paralleler Subcharts funktioniert wie folgt:

- Erzeugen Sie zunächst ein Stateflow-Chart. In diesem erstellen Sie einen State „Steuerung“. In diesem werden wiederum die States Intelligenz, Umsetzer, Baender, Silos, VE1_2 und Teller erzeugt.
- Diese States wandeln sie anschließend in **Subcharts** um. Dazu klicken Sie mit der rechten Maustaste auf den Zustand und unter der Option „Group & Subchart“ wählen Sie „subcharted“.
- Im State Steuerung sollen nun die Steuerungen der einzelnen **Komponenten parallel ablaufen** und **unabhängig voneinander implementiert** werden. Dazu klicken Sie mit der rechten Maustaste in den Zustand „Steuerung“ und unter der Option „Decomposition“ wählen Sie „(AND) Parallel“.

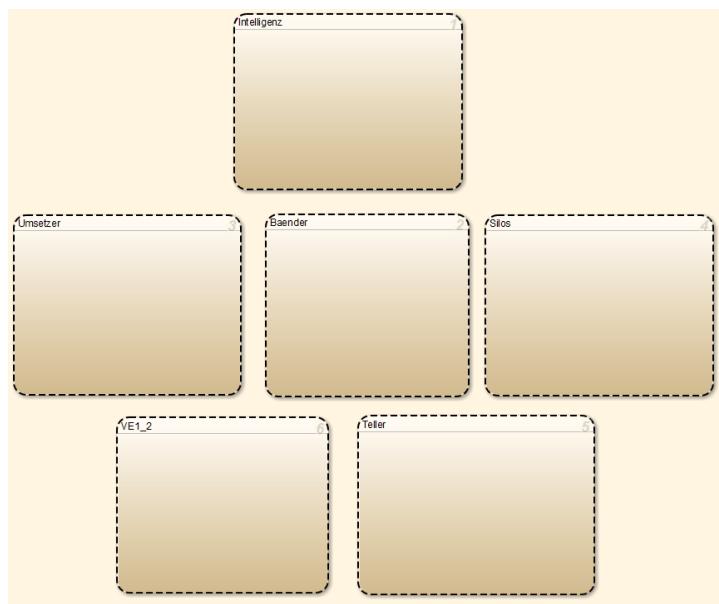


Abbildung 73: Beispielhafte Steuerung in Stateflow

Wichtige Hinweise:

- Es befinden sich **am Anfang keine Flaschen** auf der Anlage.
- Das **Zuführband** muss von Beginn an **eingeschaltet** sein und kann durchgehend laufen.
- Das **Arbeitsband** wird eingeschaltet, sobald **eine Flasche am Lichtaster 1 erkannt** wird. Ab diesem Zeitpunkt kann das Arbeitsband durchgehend laufen.
- Das **Arbeitsband muss nicht abgeschaltet** werden, bevor der **Umsetzer eine Flasche greift**.

- Der Umsetzer steht zu Beginn am Teller und der Greifer ist geschlossen.
- Leere Silos dürfen nicht geöffnet werden.
- Wenn der Umsetzer angehalten werden soll, müssen beide Aktoren auf null gesetzt werden.
- Bevor der Vereinzeler 1 öffnet muss der Umsetzer bereits am Band warten. Andernfalls wird die Flasche am Vereinzeler 2 umgeworfen.
- Das Ende des Produktionszyklus soll nicht implementiert werden (es sind auch keine Notaussituationen zu berücksichtigen).
- Der Vereinzeler 2 darf nicht schalten, sofern sich keine Flasche vor ihm befindet.
- Der Greifer des Umsetzers darf sich ohne Flasche nicht schließen.
- Beachten Sie die Nachrieselzeit.
- Es dürfen niemals beide Pneumatikventile des Umsetzers oder beide Pneumatikventile des Greifers gleichzeitig geöffnet werden.
- Der Greifer muss geöffnet sein, wenn er auf eine Flasche wartet.
- Die Vorlage verwendet C als Action Language, nicht MATLAB. Vektorzugriff also mit [], nicht (). Die Array-Indizierung beginnt jedoch weiterhin bei 1, nicht bei 0.
- In der Vorlage sind bereits Scopes vorhanden, die alle Zustände der Anlage aufzeichnen. **Nutzen Sie diese Informationen zur Fehlersuche.**

Folgende Abbildung zeigt eine mögliche Lösung der Steuerung. Jeder Graph repräsentiert hierbei eine Flasche. Dargestellt ist der Verlauf der Bahnkoordinaten über der Zeit.

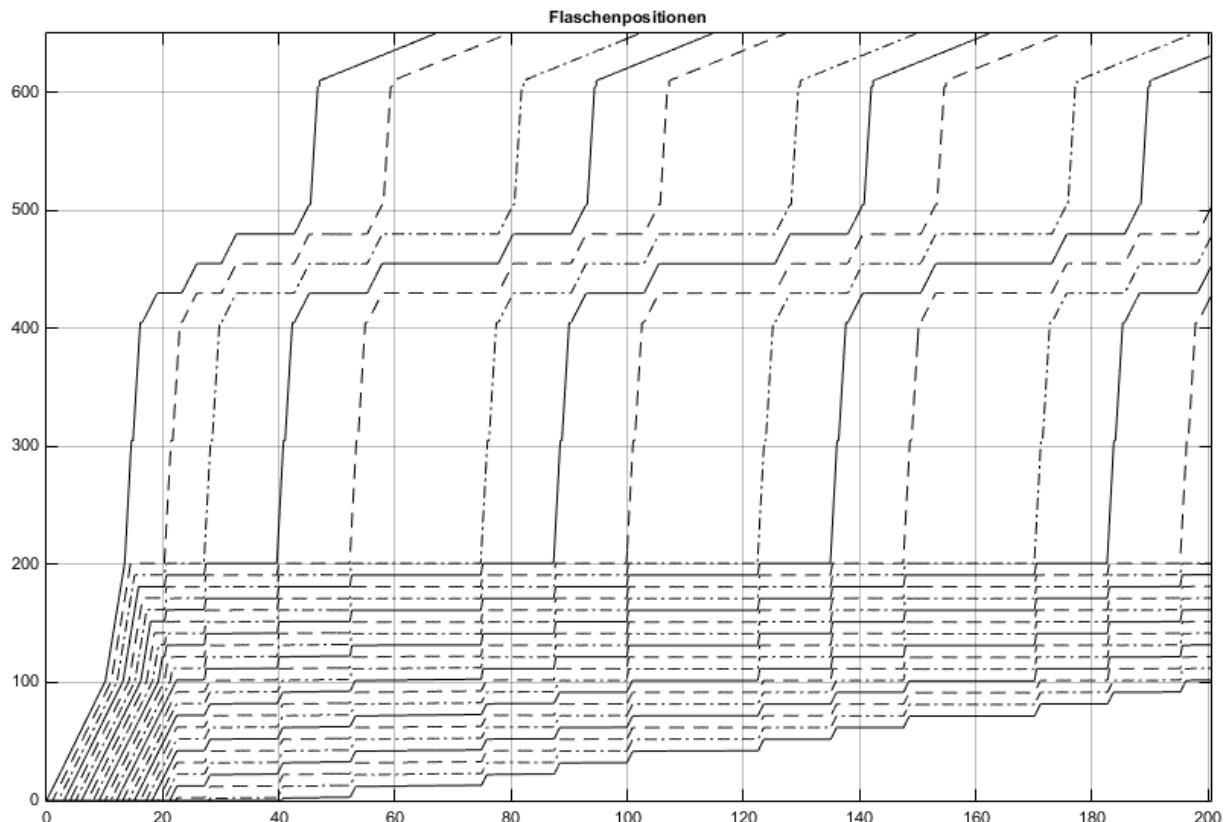


Abbildung 74: Mögliche Lösung von Versuch 4



Praktikum Simulationstechnik



Versuch 5



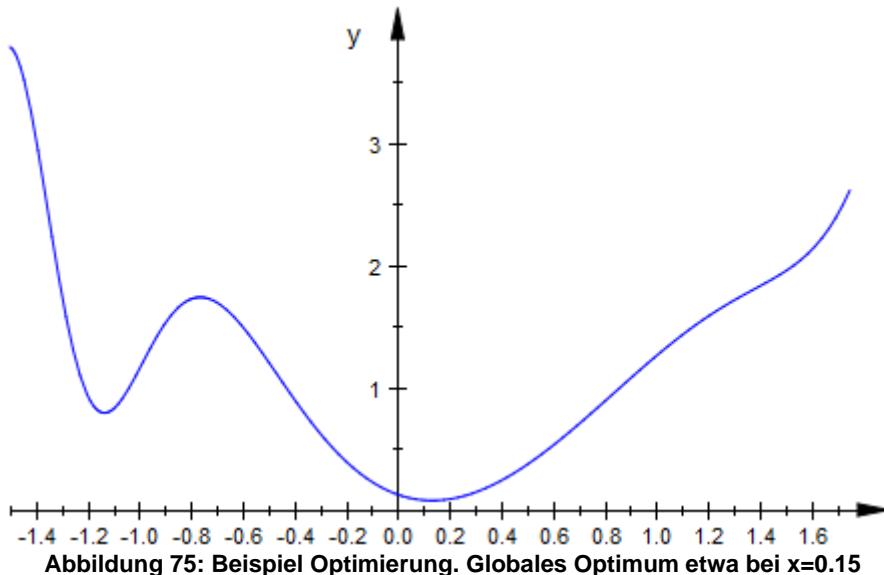
Inhaltsverzeichnis

13 GRUNDLAGEN OPTIMIERUNG	113
--	------------

13 Grundlagen Optimierung

Eine Optimierung hat das Ziel die **ideale Konfiguration** hinsichtlich einer ausgewählten Eigenschaft (z.B. Energieverbrauch, Gewicht, Geschwindigkeit) zu finden.

Der Begriff „Optimierung“ beschreibt ein sehr weites Feld mit einer Vielzahl verschiedener Ansätze und Theorien, deshalb werden an dieser Stelle nur die absoluten Grundlagen vermittelt. Bei Interesse sei auf Fachliteratur oder entsprechende Vorlesungen an dieser Fakultät verwiesen.



Prinzipielle Vorgehensweise

Zu Beginn einer Optimierung muss immer **eine zu optimierende Zielgröße oder Zielfunktion** gefunden werden. Weiterhin müssen etwaige **Einschränkungen** beachtet werden. Soll beispielsweise eine Brücke auf möglichst wenig Materialeinsatz optimiert werden muss immer auch Rücksicht auf die Tragfähigkeit genommen werden.

Im nächsten Schritt werden die **kritischen Parameter** bestimmt. Dies sind alle Parameter, die:

- Einfluss auf die Zielgröße haben und
- sich ohne weiteres ändern lassen.

Ein Beispiel zu b) ist die Breite einer Brücke. Eine schmale Brücke wird zwar weniger Gewicht haben, soll die Brücke allerdings Platz zwei Fahrspuren haben kann die Breite nicht beliebig reduziert werden.

Anschließend wird innerhalb der ausgewählten **Intervalle der Parameter** nach einem Optimum gesucht. In den meisten Fällen sind die Parameter nicht unabhängig voneinander. Das heißt die Parameter können nicht der Reihe nach optimiert werden, sondern müssen **gemeinsam variiert** werden.

Die theoretisch einfachste Möglichkeit ist es **alle möglichen Kombinationen durchzuprobieren**. Dies ist auch die Methode der Wahl für dieses Praktikum.

Anmerkung: Bei größeren Optimierungsproblemen mit vielen Parametern und langen Rechenzeiten für das Modell stößt diese Vorgehensweise durch die sehr große Anzahl möglicher Kombinationen schnell an ihre Grenzen. In diesen Fällen werden in der Praxis andere, mathematisch fortgeschrittenere Strategien verwendet (z.B. Latin-Hypercube-Design oder Gradienten-basierte Ansätze), welche teilweise auch bereits in MATLAB implementiert sind. Hierauf wird an dieser Stelle jedoch nicht weiter eingegangen.

14 Modellierung und Optimierung eines inversen Pendels

Die letzte Aufgabe des Praktikums besteht darin, ein System, bestehend aus einem inversen Pendel und dem dazugehörigen Regler, von Grund auf zu modellieren und anschließend zu optimieren.

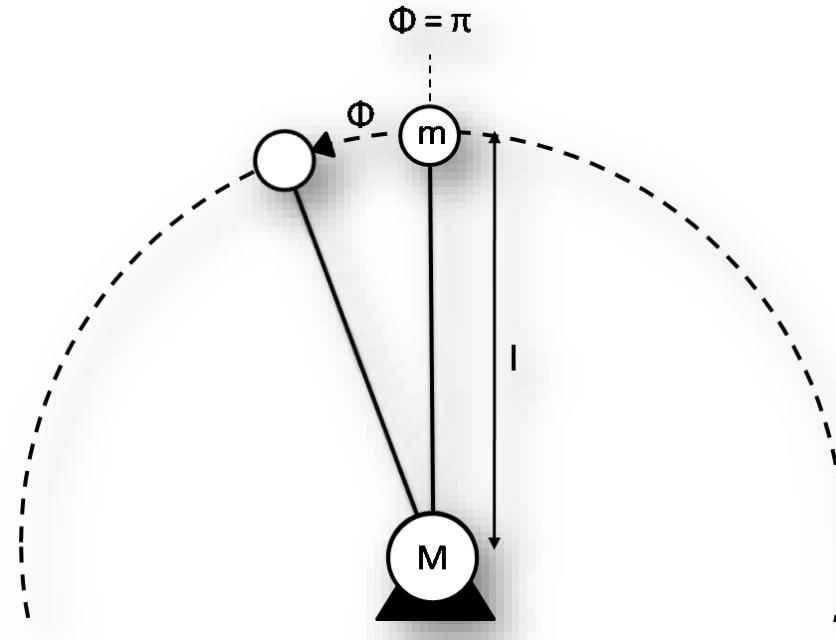


Abbildung 76: Inverses Pendel

Abbildung 76 zeigt den schematischen Aufbau des inversen Pendels. Dieses besteht aus einer Masse m , die über einen Stab (Länge l) in einem Gelenk (Reibungskoeffizient k_r) drehbar gelagert ist. An diesem Gelenk greift ein Elektromotor an, der das Moment M zum Ausbalancieren des Pendels aufbringt. Die Position der Masse m wird über den Winkel ϕ definiert, dessen Gleichgewichtslage sich bei $\phi = \pi$ befindet.

Das System wird durch folgende Bewegungsgleichung beschrieben, wobei $u(t)$ den Einfluss des Elektromotors darstellt:

$$\ddot{\phi}(t) = -\frac{k_r}{m}\dot{\phi}(t) - \frac{g}{l} \cdot \sin(\phi(t)) + u(t)$$

Als Regler soll ein üblicher PID-Regler verwendet werden, der durch die folgende Formel beschrieben wird:

$$y(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau) d\tau + K_d \cdot \frac{d}{dt} \cdot e(t)$$

$y(t)$ bezeichnet die Regelgröße (in diesem Fall die Ansteuerung für den Motor)

$e(t)$ bezeichnet die Regelabweichung (in diesem Fall die Differenz zur Gleichgewichtslage)

Als **freie Parameter** zur Optimierung stehen die jeweiligen Verstärkungsfaktoren K_i , K_p und K_d zur Verfügung.

14.1 Modellierung des Pendels

Modellieren Sie das Pendel nach der dafür angegebenen Differentialgleichung unter Verwendung der folgenden Parameter:

$$k_r = 0.02 \frac{kg \cdot m^2}{s}$$

$$m = 0.2 \text{ kg}$$

$$l = 0.5 \text{ m}$$

$$g = 9.81 \frac{N}{kg}$$

Hinweise: Testen Sie das Systemverhalten, durch eine Startposition $\phi \neq \pi$ und $u(t) = 0$. Das Pendel muss sich bei $\phi = 0$, oder $\phi = 2\pi$ einschwingen. Abbildung 77 zeigt ein mögliches Ergebnis. Dargestellt ist ein Plot des Winkels ϕ .

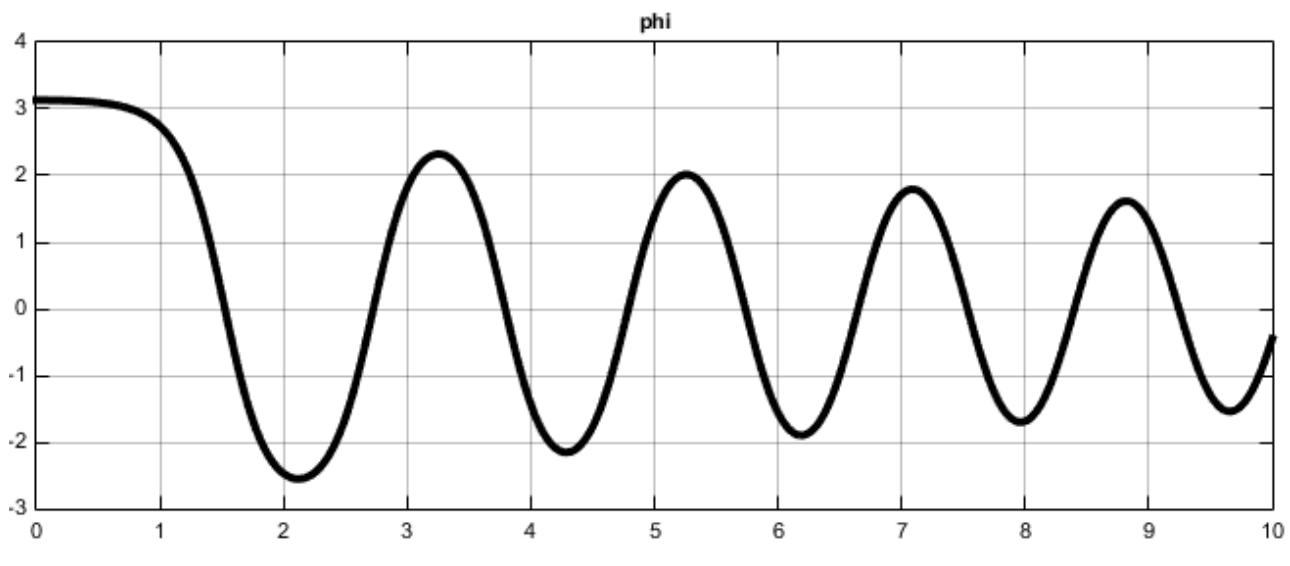


Abbildung 77: Pendel ohne Regelung (Startwert $\phi = \pi - 0.01$)

In den folgenden Schritten soll Ihr Pendelmodell um eine Regelung erweitert werden. Lassen Sie jedoch zunächst die korrekte Funktion Ihres Modells von einem Tutor überprüfen.

14.2 Regelung des Pendels

Erweitern Sie Ihr Pendelmodell um die Subsysteme *Motor* und *Regler* und sehen sie die Möglichkeit zur Einbindung einer Störgröße vor.

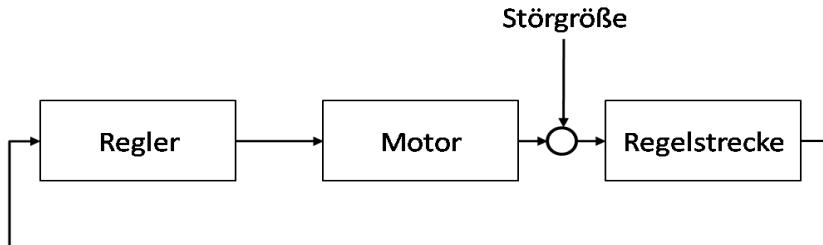


Abbildung 78: Regelkreis für das inverse Pendel

Hinweise:

- Die Motordynamik wird im Modell vereinfacht als **PT1-Verhalten** betrachtet. Die Zeitkonstante T beträgt 0.2s. Der **Verstärkungsfaktor ist eins**.
- Die **Übertragungsfunktion** eines PT1-Gliedes lautet: $G(s) = \frac{1}{Ts+1}$. Benutzen Sie zur Modellierung den Block „**Transfer Fcn**“.
- Modellieren Sie den PID-Regler nach der Formel auf Seite 114. Der Block „**PID-Controller**“ stellt **keinen klassischen PID-Regler** dar und führt daher zu einer Verfälschung des Ergebnisses.
- Verwenden Sie als Störgröße ein einzelnes Rechtecksignal der Amplitude 10 und Länge 0.1s. Dieses soll nach 1s angreifen. Das Störsignal entspricht einem Anstoßen des Pendels.
- Der **Sollzustand** des Systems ist $\Phi = \pi$.
- Die Regelabweichung $e(t)$ ist **Sollwert – Istwert**.
- Testen** Sie ihren Regler für verschiedene Parameter K_i , K_p und K_d **mit und ohne Störgröße**. Ein Beispiel für einen stabilen (jedoch bei Weitem nicht optimalen) Regler ist die Kombination $K_i = 0$, $K_p = 22$, $K_d = 15$. Dieser Regler wurde auch in Abbildung 79 verwendet.

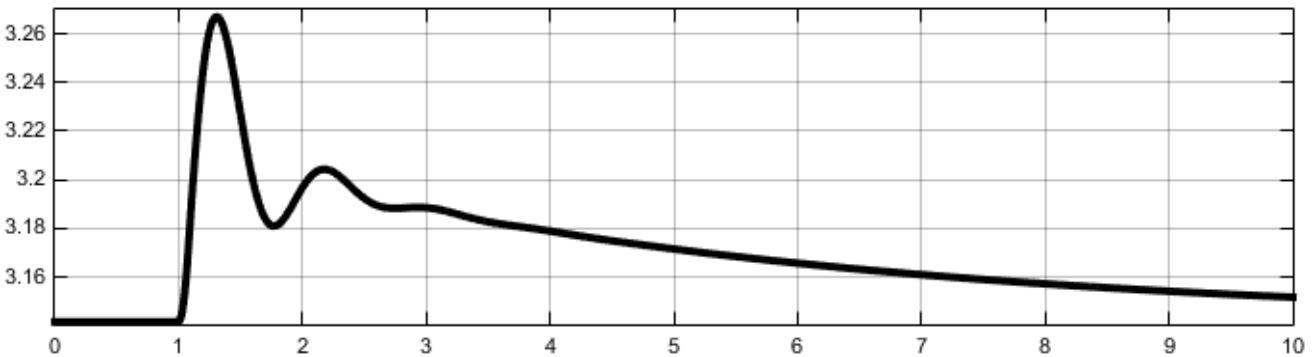


Abbildung 79: Pendel mit Regelung ($K_i = 0$, $K_p = 22$, $K_d = 15$, $\Phi = \pi$, Störung nach $t = 1$ s)



14.3 Optimierung des Pendelreglers

Optimieren Sie den Regler durch gleichzeitige Variation der drei Parameter K_i , K_p und K_d hinsichtlich des folgenden Gütekriteriums:

„Zeitpunkt der Simulation, zu dem die Regelabweichung zum letzten Mal größer als 0,001 war.“

Hinweise:

- Entwerfen Sie ein M-File, welches das Modell für **alle möglichen Parameterkombinationen** simuliert (Befehl: „`sim('Modellname')`“) und am Schluss die optimale Konfiguration und den hierfür erzielten Gütwert ausgibt.
- Ermitteln Sie das **Gütekriterium in Simulink**. Verwenden Sie hierfür den Block „*To Workspace*“, um Variablen aus Simulink im MATLAB-Workspace zu speichern. Stellen Sie dazu in den Optionen „Save format“ auf „Array“ und „Limit data points to last“ auf „1“. Durch diese Einstellungen wird nach Beendigung der Simulation nur der letzte Simulationswert als Skalar an den Workspace übergeben.
- Das Pendel ist zu Beginn in Gleichgewichtslage ($\Phi = \pi$).
- Suchen Sie das Optimum mit $K_i \in [0; 2]$ mit Schrittweite 1, $K_p \in [30; 70]$ und $K_d \in [10; 40]$ jeweils mit minimaler Schrittweite 2.
- Zur Zeitersparnis ist es empfehlenswert die Schrittweite zu Anfang etwas höher zu setzen, bis die einwandfreie Funktionsweise des Modells sichergestellt ist.
- Stellen Sie die Simulationszeit auf **10s** ein.
- Schließen Sie für den eigentlichen Optimierungsvorgang das Simulink-Modell um die Berechnung zu Beschleunigen.
- Beachten Sie unbedingt die vorherigen drei Punkte, **andernfalls dauert die Simulation sehr lange!**

14.4 Grafische Aufbereitung der Pendeloptimierung

Nun soll das Pendel nochmals optimiert werden, der Fokus liegt diesmal allerdings auf einer grafischen Aufbereitung der Optimierung und damit einer besseren Bewertbarkeit der Ergebnisse.

Das Pendel wird von nun an nur noch mit einem PD-Regler stabilisiert. Dies reicht vollkommen aus um das Pendel zumindest in einen gewünschten Korridor um die Ruhelage zurück zu führen (statische Genauigkeit ist nicht gefragt).

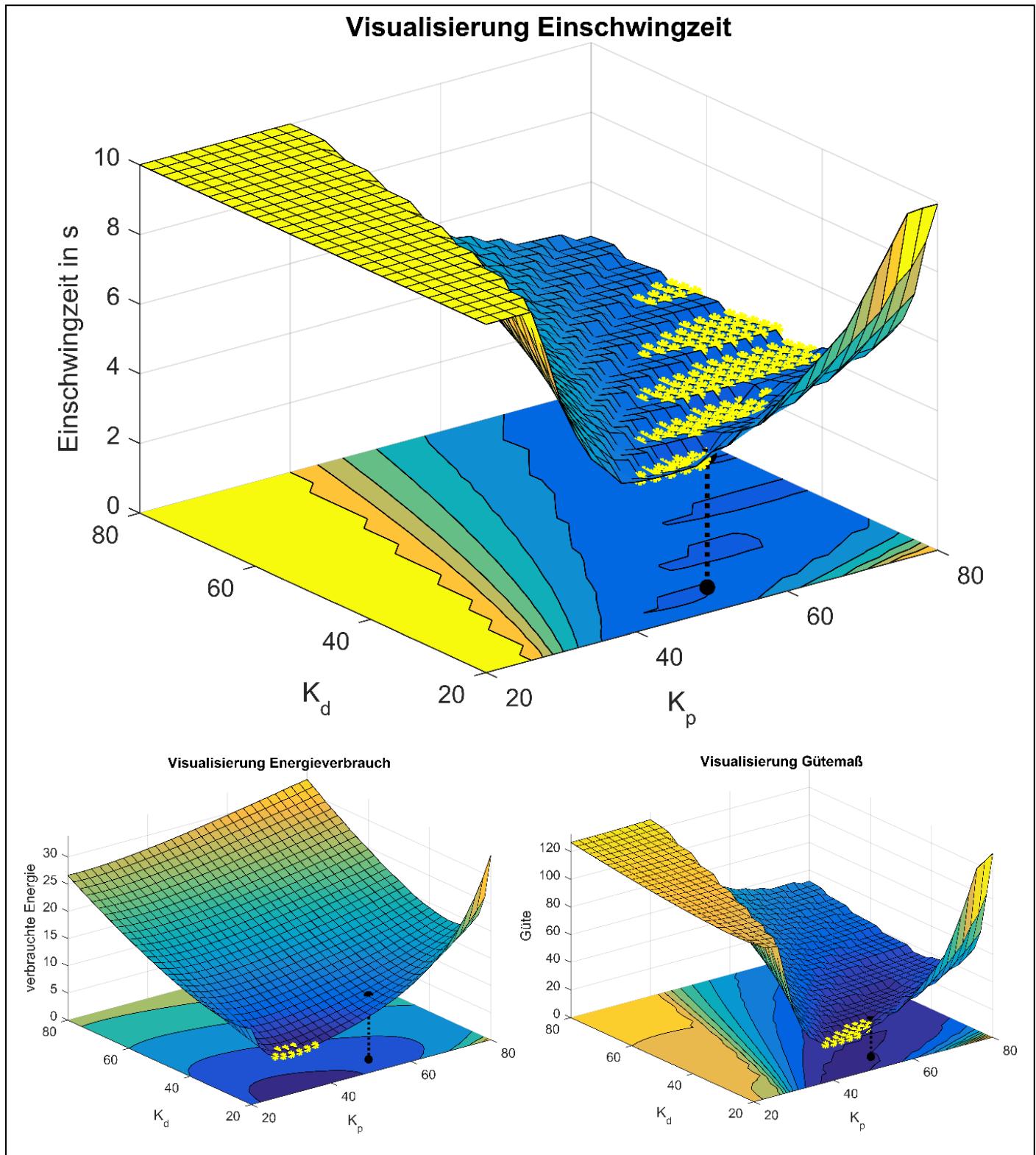
Eine gute Möglichkeit zweidimensionale Optimierungsprobleme darzustellen ist ein 3D-Plot, bei dem die dritte Achse ein Güte-Kriterium oder den zu optimierenden Wert darstellt. 3D-Plots werden in Matlab mit dem Befehl `plot3` erzeugt. Die genaue Syntax dafür entnehmen Sie der Hilfestellung von Matlab.

Der PD Regler liefert relativ gleichbleibende Ergebnisse für die Pendeloptimierung, solange das Verhältnis zwischen P und D Anteil gleich bleibt, es erhöht sich lediglich die Frequenz mit der das Pendel zurück in die Ruhelage schwingt. Dies bedeutet jedoch höhere erforderliche Stellgrößen, weshalb von nun an auch der benötigte Strom um das Pendel zu stabilisieren bewertet werden soll.

Der benötigte Strom wird einfach durch Integration des Motoreingangs gewonnen. Integriert wird nur bis der eingeschwungene Zustand erreicht ist, und der Wert wird zusammen mit der Einschwingzeit an den simout-Block übergeben (mux benutzen!).

Mit dem so gewonnen „Aufwand“, den wir betreiben um das Pendel zu stabilisieren, wird zusätzlich eine Gütfunktion aufgestellt, um eine gewichtete Kombination der Einschwingzeit und des dafür benötigten Aufwandes zu erhalten. Die Gütfunktion, die im Rahmen dieses Praktikums gewählt wird lautet

$$Q = t_{Einschwing}^2 + E_{Verbraucht} .$$



- Optimieren Sie das Pendel mit einem PD Regler im Bereich $K_p \in [20; 80]$ und $K_d \in [20; 80]$ jeweils mit Schrittweite 2.
- Speichern Sie diesmal alle Wertekombinationen von K_p , K_d , mit der zugehörigen Zeit, Energie und Gütekfunktion in einer Matrix (Initialisieren Sie dieses in der richtigen Größe am Anfang Ihres Scripts, um die Optimierung zu beschleunigen).
- Plotten Sie die so gewonnenen Daten mit Hilfe des vorgegebenen Codes in drei Graphen, benennen Sie alle Achsen und betiteln Sie den Plot. Stellen Sie hierbei die Daten als Oberflächenplot und fügen Sie einen projizierten, gefüllten Konturplot hinzu (siehe Hinweise).
- Erweitern Sie diese Plots, indem Sie alle Werte ergänzend plotten, die bis auf 10% an das jeweilige Optimum herankommen. Stellen Sie diese Punkte als grünen Asterischen (*) dar.
- Markieren Sie das durch die Zeitoptimierung gefundene Optimum in allen drei Plots als roten Kreis. Achten Sie hierbei darauf, den Punkt sowohl auf der Oberfläche als auch auf der projizierten Kontur darzustellen. Verbinden Sie die beiden Punkte mit einem Lot auf die Z-Ebene. Ist nur ein Optimum vorhanden?
- Diskutieren Sie das Ergebnis!

Hinweise:

- Um die jeweils besten 10% zu ermitteln ist der Befehl `min()` hilfreich.
 - Sie können eine Matlab-figure erweitern, indem Sie „`hold on`“ / „`hold off`“ verwenden.
 - Achtung: Richtige Indizierung der Matrizen / Vektoren wichtig. Siehe dazu Einführung zu Matlab.
 - Es wird empfohlen eine dreidimensionale Matrix zu erstellen. Diese hat die Dimensionen $n \times m \times 5$, wobei n die Anzahl der Gitterpunkte in x- und m die Anzahl der Gitterpunkte in y-Richtung entspricht. In der dritten Dimension speichern Sie folgende Werte zu jedem Gitterpunkt (Pseudocode):
- ```
M(1,1,:) = [x-Wert, y-Wert, Einschwingzeit, Energieverbrauch, Gütekriterium];
```
- Folgendes Beispiel zeigt Ihnen die Erstellung eines 3D-Oberflächenplots mit projizierter Kontur:

```
% Vektoren definieren
xVec = -3:0.2:3;
yVec = -3:0.2:3;
% Gitternetz erstellen
[X,Y] = meshgrid(xVec,yVec);
% Generiere Z-Daten mit Offset
Z = peaks(X,Y)+6.531; %Beispiel

%Plotten
figure;
surf(X,Y,Z);
hold on;
contourf(X,Y,Z);
stem3(-2.5,-2.5,6.531,'kx');
plot3(-2.5,-2.5,0,'kx');
grid on;
xlabel('x');
ylabel('y');
zlabel('z');
axis('tight');
colormap('parula');
title('Peaks Function');
hold off;
```

