
TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

# DIPLOMARBEIT

## Modellkopplung thermodynamischer Systeme

Ausgeführt am Institut für

Analysis & Scientific Computing

der Technischen Universität Wien
unter der Anleitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Felix Breitenecker

durch

**Matthias Rößler**

Witzendorferortsstraße 16
A-3100 St. Pölten

Wien, im Juni 2012

**Zusammenfassung**

In dieser Arbeit wird die Kopplung von Modellen, die aus physikalischen Domänen stammen, betrachtet. Ein Hauptaugenmerk liegt dabei auf dem thermodynamischen Verhalten der Modelle. Nach einer Einleitung, bei der unter anderem das Projekt, im Zuge dessen diese Arbeit verfasst wurde, vorgestellt wird, werden die Grundlagen des Modellbildungsansatzes *Physical Modelling* und einige wichtige Erkenntnisse aus der Theorie der Thermodynamik vorgestellt, sowie eine Einführung in die theoretischen Hintergründe der Modellkopplung gegeben. Danach werden zwei spezielle Modelle, ein Modell einer Maschine und ein Raummodell, die gekoppelt werden sollen, und die Versuchsaufbauten, die zur Verifizierung der Modelle dienen, vorgestellt. Es folgt eine Beschreibung der verwendeten Simulatoren und ein Kapitel über die Implementierung der vorgestellten Modelle. Darüber hinaus werden die Simulationsergebnisse präsentiert und es wird ein Ausblick auf mögliche Modellerweiterungen und zukünftige Arbeiten gegeben.

**Abstract**

In this work the coupling of models derived from physical domains is considered. A main focus lies on the thermodynamic behaviour of the models. First an introduction, in which, among other things, the project, the work was written in, is presented. After that the fundamentals of the modeling approach *Physical Modelling* and some important insights from the theory of thermodynamics, as well as an introduction to the theoretical background of model coupling are presented. Thereafter, two special models, a model of a machine and a room model, which are to be coupled, and the test rigs, which serve to verify the models, are presented. This is followed by a description of the used simulators and a chapter on the implementation of the presented models. After that the simulation results are presented and an outlook on possible model extensions and future work is given.

# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Motivation

In der heutigen Zeit ist Energieoptimierung ein sehr wichtiges Thema. Aus diesem Grund wurde das Forschungsprojekt INFO, das später genauer vorgestellt wird, ins Leben gerufen. Da sich das Projekt zum Ziel gesetzt hat, eine Gesamtsimulation einer Maschinenhalle zu erstellen, stellt sich die Frage, wie ein solch großes Modell aufgebaut und simuliert werden kann. Im Zuge dieser Diplomarbeit werden deshalb zwei Teilbereiche des Gesamtmodells, nämlich ein Modell einer Maschine und ein Modell, das das thermodynamische Verhalten eines Raumes beschreibt, herausgegriffen und es wird untersucht, ob die Modelle mit Hilfe des Ansatzes der Single-Solver Simulation miteinander gekoppelt werden können, das heißt ob sie in einem Simulator zu einem Modell zusammengefügt werden können.

Ein weiterer Aspekt dieser Arbeit, der immer wieder interessant ist, ist der Vergleich von Simulatoren. Dazu werden die untersuchten Modelle in zwei verschiedenen Simulatoren, die aber auf dem gleichen Modellbildungsansatz (Physical Modelling mit Modelica) beruhen, implementiert um zu untersuchen, welcher der beiden Simulatoren mit der gestellten Aufgabe besser zurechtkommt, beziehungsweise wo Unterschiede im Modellaufbau liegen.

## 1.2 Projekt INFO

Das INFO-Projekt [1], an dem mehrere Institute der TU Wien, sowie Industriepartner beteiligt sind, beschäftigt sich mit der Energieoptimierung in Fertigungsbetrieben. Durch die ganzheitliche Modellierung einer Werkzeughalle soll die Energieeffizienz in der Produktion gesteigert werden. Das Projekt wird von der Österreichischen Fördergesellschaft FFG finanziert.
Hierzu werden Simulationsmodelle erstellt, die alle Bereiche einer Werkzeughalle (Maschinen, Prozesse, Gebäudehülle, Gebäudestruktur, Standort,...) abbilden. In diesen Teilmodellen werden in einem ersten Schritt die Schlüsselparameter identifiziert und danach werden die Einzelmodelle zu einem Gesamtmodell gekoppelt. Diese Gesamtsimulation soll dann durch Messungen in bestehenden Fertigungshallen von Industriepartnern validiert werden. In Abbildung 1 wird eine schematische Darstellung dieses Aufbaus gezeigt.

Neben der Gesamtsimulation, die zur Energieoptimierung in geplanten Fertigungshallen verwendet werden könnte, sind natürlich auch die einzelnen

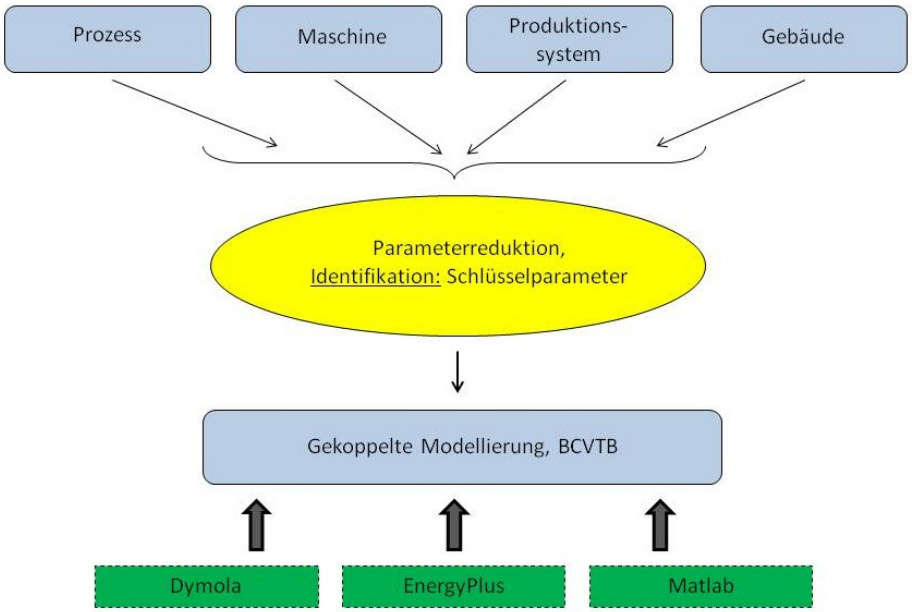



Abbildung 1: Schematische Darstellung des INFO-Projekts

Teilmodelle ein wichtiges Resultat des Projekts, da diese für weitere Entwicklungen zur Verfügung stehen werden.
Die einzelnen Simulationen werden hauptsächlich in DYMOLA (Tool zur Simulation physikalischer Systeme), EnergyPlus (Tool zur energetischen Gebäudesimulation) und Matlab gemacht, die Kopplung der Teilmodelle erfolgt in BCVTB (*Building Controls Virtual Test Bed* – Co-Simulationsumgebung ausgelegt auf Gebäudesimulation).

## 1.3 Ziele

Diese Arbeit beschäftigt sich hauptsächlich mit der Kopplung verschiedener Modellteile in einem Simulator. Diese Art der Kopplung soll eingehend untersucht werden und es sollen Lösungsvorschläge für auftretende Probleme erarbeitet werden. Weiters soll die Sinnhaftigkeit einer solchen Kopplung in einem großen Gesamtmodell untersucht werden, also ob die Änderungen, die bei den Teilmodellen notwendig werden, sinnvoll sind oder ob eine andere Art der Kopplung für eine Gesamtsimulation besser wäre.

Weiters sollen verschiedene Simulatoren hinsichtlich ihrer Eignung zur Simulation größerer Systeme miteinander verglichen werden. Neben den Er-

gebnissen für das gekoppelte Gesamtsystem können auch die Ergebnisse der Teilmodelle miteinander verglichen werden, bei denen sowohl auf modellbildnerischer Ebene als auch für die verwendeten Solver und somit den Simulator einige Herausforderungen gegeben sind.

Ein weiteres Ergebnis dieser Arbeit soll die Untersuchung der erstellten Teilmodelle darstellen. Durch die Verwendung zweier Simulatoren und durch Versuchsaufbauten kann eine Validierung erfolgen und zusätzlich noch zwischen Problemen der Modellbildung und Problemen der Simulation, also numerischen Effekten, unterschieden werden.

# 2 Grundlagen

## 2.1 Physical Modelling

Bei der klassischen Modellbildung physikalischer Systeme wird zuerst eine mathematische Beschreibung des Systems (Zustandsraum) mit Hilfe von bekannten Zusammenhängen zwischen den Zustandsgrößen (Kirchhoffsche Regeln, Newtonsche Axiome, ...) generiert. Dies führt zu einem System von gewöhnlichen Differentialgleichungen (*ODEs*) oder differential-algebraischen Gleichungen (*DAEs*). In einem Simulator wird dann ein Modell erzeugt, zum Beispiel baut man in Simulink ein Blockschaltbild auf, das mathematische Operationen und die Zusammenhänge zwischen den Zustandsvariablen enthält. Im Simulator wird dann wieder ein expliziter oder impliziter Zustandsraum generiert, welcher dann einem ODE-Solver übergeben wird, der eine numerische Lösung zurückgibt (*Top-Down-Approach*).
Abbildung 2 zeigt den Weg von einem elektrischen Schaltkreis, in diesem Fall ein RL-Glied, zum Simulink-Modell. Gleichung (1) zeigt die Differentialgleichung, die das System beschreibt.

$$L \cdot \frac{\mathrm{d}i}{\mathrm{d}t} = U - R \cdot i \tag{1}$$

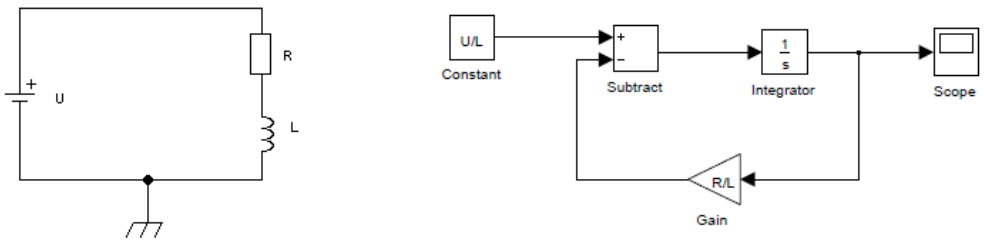



Abbildung 2: Ersatzschaltbild und Simulink-Modell eines einfachen Schaltkreises

Da die Spannung durch die Quelle vorgegeben ist, ist die einzige Veränderliche in dem System der Strom, der durch den Schaltkreis fließt. Wie man sieht, kann man aus dem Simulink-Modell direkt Zusammenhänge zwischen den Systemvariablen herauslesen und das zu grunde liegende Gleichungssystem kann direkt bestimmt werden.

Ein großer Nachteil dieses Ansatzes ist, dass die Modelle nicht direkt erweiterbar sind. Würde man einen Kondensator in den Schaltkreis einbauen, würde sich das Gleichungssystem komplett verändern und das Simulink-Modell müsste komplett neu aufgebaut werden. Bei komplexeren Modellen kommt noch hinzu, dass das Simulink-Modell sehr schnell groß und unübersichtlich wird.
Im Gegensatz dazu wird beim Physical Modelling nicht das Systemverhalten, sondern das Verhalten der einzelnen Komponenten in dem System, die dann mit Hilfe verschiedener Regeln verknüpft werden, durch Gleichungen beschrieben (*Bottom-Up-Approach*). Durch die lokale Beschreibung können Modelle leicht erweitert werden und es gibt auch immer einen Zusammenhang zwischen dem Aufbau des Modells und des zu grunde liegenden, physikalischen Systems.

### 2.1.1 Grundbegriffe

Beim Aufbau eines Modells werden Komponenten miteinander verbunden. Diese Verbindungen beschreiben Energieflüsse zwischen den Komponenten, die durch zwei Typen von Variablen beschrieben werden, die klarerweise von der physikalischen Domäne abhängen, in der das Modell aufgebaut ist. Diese Variablen werden als *Effort*- und *Flow*-Variablen oder *Across*- und *Through*-Variablen bezeichnet. Tabelle 1 zeigt eine Zusammenstellung dieser Variablen in verschiedenen physikalischen Domänen.

| Domäne | Effort-Variable ($e(t)$) | Flow-Variable ($f(t)$) |
|---|---|---|
| Mechanik - transl. | Geschwindigkeit [m/s] | Kraft [N] |
| Mechanik - rot. | Winkelgeschw. [rad/s] | Drehmoment [Nm] |
| Elektrotechnik | Spannung [V] | Strom [A] |
| Thermodynamik | Temperatur [K] | Wärmestrom [J/s] |
| Hydraulik | Druck [N/m$^2$] | Volumenstrom [m$^3$/s] |

Tabelle 1: Effort und Flow für verschiedene Domänen

Für allgemeine Beschreibungen wird die Effort-Variable $e(t)$ und die Flow-Variable $f(t)$ genannt. Für jede Verbindung gelten zwei fundamentale Gleichungen:

$$e_i(t) = e_j(t) \qquad \forall i, j \tag{2}$$

$$\sum_i f_i(t) = 0 \tag{3}$$

In der Elektrotechnik könnten (2) und (3) als die Kirchhoffschen Regeln gedeutet werden.
Wie bereits erwähnt, werden durch die Verbindungen Komponenten miteinander verknüpft. Diese Komponenten haben zwei bestimmende Charakteristika:

- *Ports* oder *Konnektoren*, die die Anzahl der möglichen Verbindungen und die physikalischen Domänen bestimmen.
- *Gleichungen*, die das lokale Verhalten der Komponente bestimmen und einen Zusammenhang zwischen Effort- und Flow-Variable herstellen.

Das heißt, dass die Ports die Endstücke der Verbindungen bilden und somit nur Ports aus der gleichen Domäne sinnvoll miteinander verbunden werden können. Weiters werden die Komponenten lokal durch die ihnen zugrunde liegenden Gleichungen beschrieben. Abbildung 3 zeigt einen Widerstand, der durch die Gleichungen (4)-(7) beschrieben wird.

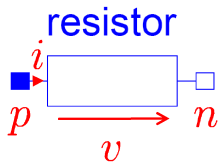



Abbildung 3: Graphische Repräsentation eines Widerstandes in Modelica

$$v = p.v - n.v \tag{4}$$

$$0 = p.i + n.i \tag{5}$$

$$i = p.i \tag{6}$$

$$R \cdot i = v \tag{7}$$

In den Gleichungen sieht man, dass sowohl interne Variablen in der Komponente, als auch Variablen in den beiden Ports $p$ und $n$ definiert sind. Die Variablen $v$ und $i$ sind durch das Ohmsche Gesetz miteinander verknüpft.
In den Gleichungen (4)-(6) sind die Verknüpfungen zwischen den internen Variablen, $v$ und $i$, mit den Variablen an den Ports, $p.v$, $n.v$, $p.i$ und $n.i$, zu sehen. Diese Form der Verknüpfung ist typisch für den *Modelica*-Standard, einem Sprachstandard für Physical Modelling. Andere nennenswerte Sprachstandards für Physical Modelling sind die *Simscape*-Language, eine Eigenentwicklung von MathWorks Inc., und *VHDL-AMS*.

```
model Resistor "Ideal linear electrical resistor"
  extends Modelica.Electrical.Analog.Interfaces.OnePort;
  parameter SI.Resistance R(start=1000) "Resistance";
equation
  R*i = v;
end Resistor;

partial model Modelica.Electrical.Analog.Interfaces.OnePort
  SI.Voltage v;
  SI.Current i;
  PositivePin p;
  NegativePin n;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end OnePort;
```

Abbildung 4: Elektrischer Widerstand Modelica

```
entity resistor is
port (
terminal p1, p2 : electrical);
end entity resistor;

architecture ideal of resistor is
constant R : real := 10.0e3;
quantity v across i through p1 to p2;
begin -- architecture ideal
i == v / R;
end architecture ideal;
```

Abbildung 5: Elektrischer Widerstand VHDL-AMS

```
component resistor < foundation.electrical.branch
parameters
    R = { 1, 'Ohm' };   % Resistance
  end
  function setup
    if R <= 0
        pm_error('simscape:GreaterThanZero','Resistance')
    end
  end
  equations
    v == R*i;
  end
end

component(Hidden=true) branch
  nodes
    p = foundation.electrical.electrical;
    n = foundation.electrical.electrical;
  end
  variables
    i = { 0, 'A' };
    v = { 0, 'V' };
  end
  function setup
    across( v, p.v, n.v );
    through( i, p.i, n.i );
  end
end
```

Abbildung 6: Elektrischer Widerstand Simscape

In den Abbildungen 4, 5 und 6 sind die Modelle eines elektrischen Widerstandes in den drei Standards zu sehen. Allen gemeinsam ist, neben dem Physical Modelling Ansatz und der Verwendung des Ohmschen Gesetzes, auch die Nutzung von Konzepten aus der objektorientierten Programmierung, wie Klassen und Vererbung.
Ein auffälliger Unterschied ist, dass bei Modelica die Verknüpfungen zwischen den Variablen in der Komponente und an den Ports direkt in den Gleichungen definiert werden, während bei VHDL-AMS und Simscape die Verknüpfung über die vordefinierten Funktionen `through` und `across` erfolgt.

Die vorgestellten Sprach-Standards sind keine Simulatoren, das heißt sie definieren nur, wie das Modell aufgebaut wird, nicht aber, wie das entstehende System gelöst wird. Die Simscape-Language ist eine Toolbox, die in Simulink eingebettet ist und die dort bereitgestellten Solver benutzt. Für VHDL-AMS und Modelica gibt es mehrere Simulatoren, die den definierten Standard verstehen. Ein VHDL-AMS-Simulator ist zum Beispiel *Portunus*, entwickelt von der *CEDRAT Group*. Modelica-Simulatoren sind unter anderem *Dymola*, *MapleSim* oder *OpenModelica*. Da in dieser Arbeit zwei Modelica-Simulatoren betrachtet werden, wird im Folgenden der Modelica-Standard genauer vorgestellt, wobei nur die für das Physical Modelling relevanten Aspekte berücksichtigt werden, eine genauere Beschreibung der Grundbegriffe des Modelica-Standards sind in [2] zu finden.

### 2.1.2 Modelica

1996 begannen die Arbeiten an einer standardisierten Sprache zur Modellierung dynamischer Systeme initiiert durch Hilding Elmqvist. 1999 wurde Modelica 1.3 als erste Version, die auch in Anwendungen verwendet wurde, veröffentlicht. Seither wird die Verfeinerung des Modelica-Standards unter der Aufsicht der Modelica Association weitergeführt. Die letzte Version ist Modelica 3.2, veröffentlicht im März 2010 [3].

| | |
|---|---|
| `class` | Allgemeine Klasse ohne Einschränkungen |
| `model` | Wird meistens zum Modellieren verwendet, die einzige Einschränkung ist, dass `model` nicht in `connections` verwendet werden darf |
| `connector` | Enthält die Definition einer Schnittstelle zwischen einer Klasse und ihrer Umgebung, es sind keine Gleichungen erlaubt |
| `type` | Durch sehr kurze Syntax werden neue Typen von Feldern deklariert, es können nur bestehende Typen erweitert werden und es dürfen keine Gleichungen verwendet werden |
| `block` | Kausalität ist für jede Variable vordefiniert (Inputs bzw. Outputs), darf nicht in `connections` verwendet werden |
| `package` | Wird verwendet um Modelica-Code zu verwalten, es dürfen nur Klassen-Definitionen und Parameter vorkommen, keine Variablen |

Tabelle 2: Klassen in Modelica

Die Grundbausteine von Modelica sind Klassen. Eine Klasse kann beliebig

viele Objekte (oder auch Instanzen) beinhalten. Die wichtigsten Elemente einer Klasse sind Deklarationen von Feldern (*field-declarations*) und Gleichungen. In Feldern werden die Daten der Instanzen der Klasse gespeichert, während die Gleichungen das Verhalten der Klasse bestimmen. Neben der `class`-Klasse, die alle Funktionen bietet, gibt es noch weitere Arten von Klassen, die einigen Restriktionen unterworfen sind, diese sollen vor allem die Lesbarkeit des Codes erhöhen. Die Unterscheidung erfolgt durch verschiedene Schlüsselworte am Beginn der Klasse, eine Liste der wichtigsten Arten ist in Tabelle 2 zu finden.

Am Anfang einer Klasse müssen die verwendeten Felder deklariert werden. Die Syntax der Deklaration sieht im Allgemeinen folgendermaßen aus:

$$<\textit{Präfix}> <\textit{Typ}> <\textit{Name}>$$

Präfixe sind Schlüsselwörter, die das Verhalten des Feldes bestimmen.

- `constant`: Eine Konstante ändert nie ihren Wert, sobald er festgelegt wurde (zum Beispiel mathematische oder Natur-Konstanten).
- `parameter`: Ein Parameter kann zur Simulationszeit seinen Wert nicht ändern, kann aber zum Beispiel bei der Initialisierung immer wieder neu festgelegt werden.
- `discrete`: Diskrete Variablen ändern ihren Wert zu diskreten Zeitpunkten, bleiben dazwischen aber konstant.
- `input` / `output`: Input- bzw. Output-Variablen können verwendet werden, wenn die Kausalität im Vorhinein klar ist (bei `block`-Klassen müssen alle Variablen entweder als Input oder Output deklariert werden).
- `flow`: Flow-Variablen werden in `connector`-Klassen verwendet und bestimmen die aufgestellten Gleichungen in `connections`, Variablen ohne `flow`-Präfix werden als Effort-Variablen gedeutet (siehe Gleichungen (2) und (3)).

Es gibt noch weitere Präfixe, die unter anderem Zugriffsrechte auf die Felder definieren, auf diese soll hier aber nicht näher eingegangen werden. Hat ein Feld kein Präfix, wird es als Variable gedeutet und kann über die Zeit kontinuierlich seinen Wert verändern.
Zusätzlich zu den Präfixen für Felder können auch Klassen Präfixe zugeordnet werden, ein sehr häufig auftretendes ist `partial`, das definiert, dass eine Klasse unvollständig ist und nur als Layout für andere Klassen verwendet werden kann. Dass eine Klasse die `partial`-Klasse erweitert, wird durch das Schlüsselwort `extends` gekennzeichnet.

Der Typ eines Feldes bestimmt, in welchem Datenformat es gespeichert wird, neben den vordefinierten Typen

- `Integer`
- `Real`
- `Boolean`
- `String`

kann auch jede andere Klasse als Typ eines Feldes dienen.

Der Name des Feldes dient der Zuordnung, über ihn kann auf das Feld wieder zugegriffen werden. Nach dem Namen können den Variablen und Parametern noch verschiedene Werte zugeordnet werden, wie zum Beispiel Start- und Default-Werte, Einheiten oder ein Definitionsbereich.
Zur Illustration der Syntax wird das Modell des elektrischen Schaltkreises aus Abbildung 2 betrachtet. Dazu wird Gleichung (1), mit der Variable $i$ und den Parameter $R$, $L$ und $U$, benutzt. Abbildung 7 zeigt, wie das Modell in einer einzelnen Klasse aufgebaut ist.

```
model RLGlied
  Real i(start=0,unit="A");
  parameter Real R(unit="Ohm")=10;
  parameter Real L(unit="H")=1;
  parameter Real U(unit="V")=1;
equation
  L*der(i)=U-R*i;
end RLGlied;
```

Abbildung 7: Modell eines RL-Glieds in Modelica

Der Modelica-Sprachstandard ist gleichungsbasiert, das heißt, dass in der Equation-Section wirklich Gleichungen stehen können und es keine Zuordnungen sein müssen, es also egal ist, ob die Gleichung in der Form $x = y$ oder $x - y = 0$ gegeben ist. Das ist eine wichtige Voraussetzung für den Ansatz des Physical Modellings, da im Vorhinein nicht immer klar ist, welche Variable aus welcher Gleichung bestimmt wird. Sollte die Kausalität im Vorhinein klar sein, kann statt der Gleichung auch eine Zuordnung durch $x := y$ definiert werden, dadurch kann zum Beispiel die Rechenzeit verkürzt werden, da die Zuordnung nicht direkt in das Gleichungssystem aufgenommen wird.
In den Gleichungen können alle Variablen und Parameter vorkommen, die

vorher definiert wurden, zusätzlich kann noch die Simulationszeit (`time`) verwendet werden. Neben den mathematischen Operationen `+,-,*,/,^` dürfen auch Funktionen wie `sin`, `cos` oder `exp` vorkommen. Auch mathematische und Natur-Konstanten können benutzt werden, diese sind in der Modelica-Standard-Library hinterlegt und es kann durch `Modelica.Constants.x` auf sie zugegriffen werden. In diesem Zusammenhang sei auch erwähnt, dass auch SI-Einheiten in der Modelica-Standard-Library hinterlegt sind. Diese sind als Typen hinterlegt und können mit Hilfe von `Modelica.SIunits.x` benutzt werden. Zur Illustration zeigt Abbildung 8 das Modell des RL-Schaltkreises aus Abbildung 7, der in diesem Fall mit Wechselspannung betrieben wird.

```
model RLGliedWechsel
  Modelica.SIunits.Current i(start=0);
  Modelica.SIunits.Voltage U;
  parameter Modelica.SIunits.Resistance R=10;
  parameter Modelica.SIunits.Inductance L=1;
  parameter Modelica.SIunits.Frequency f=10;
  parameter Modelica.SIunits.Voltage A=10;
  constant Real pi=Modelica.Constants.pi;
equation
  U:=A*sin(2*pi*f*time);
  L*der(i)=U-R*i;
end RLGliedWechsel;
```

Abbildung 8: RL-Glied mit Wechselspannung in Modelica

Wie bereits erwähnt, ist der Grundgedanke des Physical Modellings, dem auch Modelica folgt, die lokale Beschreibung der Komponenten, um die Wiederverwendbarkeit dieser zu ermöglichen.
Im Folgenden soll der Aufbau einer Library in Modelica exemplarisch an Hand der Komponenten des Schaltkreises aus Abbildung 8 gezeigt werden.
Als erster Schritt müssen die Schnittstellen zwischen den einzelnen Komponenten definiert werden, in Modelica wird dies durch die bereits erwähnten Konnektoren realisiert. Abbildung 9 zeigt den Aufbau eines Konnektors aus der Elektrotechnik, wie er auch in der Modelica-Standard-Library verwendet wird.

Als nächstes werden die einzelnen Komponenten als Modelle aufgebaut, später kann sich das Modell des Schaltkreises dann als Modell mit mehreren Submodellen, die die Komponenten enthalten, interpretiert werden. Da sehr viele Komponenten (nicht nur in der Elektrotechnik) zwei Konnektoren ha-

```
connector Pin
  Modelica.SIunits.Voltage v;
  flow Modelica.SIunits.Current i;
end Pin;
```

Abbildung 9: Konnektor in Modelica

ben, gibt es in der Modelica-Standard-Library die Klasse `OnePort`, die ein unvollständiges Modell einer elektrischen Komponente mit zwei Pins darstellt (Abbildung 10).

```
partial model OnePort
  Modelica.SIunits.Voltage v;
  Modelica.SIunits.Current i;
  Pin p;
  Pin n;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end OnePort;
```

Abbildung 10: Oneport-Klasse in Modelica

Die Modelle des Widerstandes, der Spule und der Spannungsquelle sind eine Erweiterung dieser Oneport-Klasse. Zur Illustration zeigt Abbildung 11 den Source-Code der Spannungsquelle, der Aufbau der beiden anderen Komponenten ist analog.

Die letzte Komponente, die noch benötigt wird, ist das Referenzelement `Ground`. Dieses Element hat nur einen Konnektor `p` und die bestimmende Gleichung ist `p.v=0`, das heißt, dass das Potential im Referenzelement auf 0 gesetzt wird.
Bevor das Modell des Schwingkreises mit den hier definierten Komponenten aufgebaut wird, sei noch erwähnt, dass es sich nicht um die Komponenten aus der Modelica-Standard-Library handelt. Diese Unterscheiden sich teilweise durch ihre Syntax, wobei die verwendeten Gleichungen im Endeffekt natürlich die gleichen bleiben. Ein nennenswerter Unterschied ist, dass in der Modelica-Standard-Library zwei Arten von Pins gibt, den `PositivePin` und den `NegativePin`, der Grund dafür liegt nicht in den Gleichungen der

```
model VoltageSource
  extends OnePort;
  parameter Modelica.SIunits.Voltage A;
  parameter Modelica.SIunits.Frequency f;
  constant Real pi=Modelica.Constants.pi;
equation
  v=A*sin(2*pi*f*time);
end VoltageSource;
```

Abbildung 11: Wechselspannungsquelle in Modelica

Pins, sondern in der graphischen Repräsentation der Pins. Da Modelica auch darauf ausgelegt ist, dass der Simulator auch einen graphischen Editor bereitstellt, ist es sinnvoll die Flussrichtung des Stroms visuell zu kennzeichnen, dies geschieht eben über die beiden Pins (vgl. Abbildung 3).

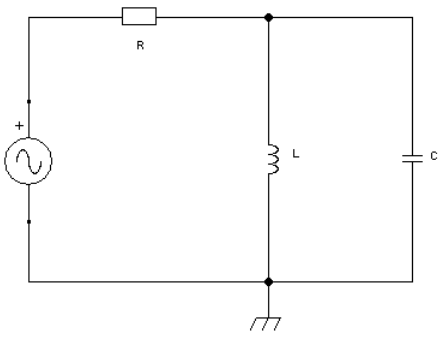
Beim Aufbau des Gesamtmodells kann auf alle zuvor definierten Komponenten zurückgegriffen werden. Die Ports der einzelnen Komponenten werden mit Hilfe der Funktion `connect` verbunden, die im Gleichungssystem dann genau die Gleichungen (2) und (3) aufstellt. Abbildung 12 zeigt den Aufbau des Gesamtmodells.

```
model RLGliedKomp
  VoltageSource U(A=10,f=10);
  Resistor R(R=10);
  Inductor L(L=1);
  Ground G;
equation
  connect(U.p, R.p);
  connect(R.n, L.p);
  connect(L.n, G.p);
  connect(L.n, U.n);
end RLGliedKomp;
```

Abbildung 12: Modell des RL-Glieds mit lokalen Komponenten

Die beiden Modelle aus Abbildung 8 und 12 beschreiben das gleiche System, der große Vorteil des zweiten Modells liegt darin, dass es sehr leicht erweitert werden kann. Bei der Erweiterung zu einem RLC-Schwingkreises, wie er in Abbildung 13 gezeigt wird, müssen in dem gleichungsbasierten Mo-

dell die System-Gleichungen wieder händisch hergeleitet werden, während bei dem komponentenbasierten Modell die Gleichungen vom Simulator zusammengesammelt werden und deshalb nur eine neue Komponente und ein paar Verbindungen eingefügt werden müssen, wobei das Modell des Kondensators wieder als Erweiterung der `OnePort`-Klasse aufgebaut werden kann.




```
model RLCGlied
  extends RLGliedKomp;
  Capacitor C(C=1);
equation
  connect(R.n, C.p);
  connect(C.n, G.p);
end RLCGlied;
```

Abbildung 13: Ersatzschaltbild und Modelica-Modell eines RLC-Glieds

Zum Abschluss dieses Kapitels sollen noch kurz Annotationen erwähnt werden. Annotationen sind eigentlich kein Teil des Modelica-Standards, da sie keinen Einfluss auf den Aufbau des Modells haben, bilden aber in gewisser Weise eine Erweiterung. Wie bereits erwähnt ist Modelica darauf ausgelegt, dass es in einem Simulator auch einen graphischen Editor gibt, um den Aufbau der Modelle leicht zu machen und übersichtlich zu halten. Annotationen sind eine Möglichkeit, die von Modelica bereitgestellt wird, um die graphische Repräsentation in der Komponente zu speichern. Ein weiterer Verwendungszweck ist die Dokumentation, die auch in der Annotation hinterlegt ist.
Annotationen werden meist vom Editor automatisch generiert, ein großer Vorteil ist, dass dadurch Komponenten und Modelle von einem Simulator in einen anderen portiert werden können und dabei auch die graphische Repräsentation und die Dokumentation mitgegeben wird.

Der Modelica-Standard liefert eine Modellbeschreibungssprache die in einer Vielzahl von Simulatoren verwendet wird, was er nicht definiert ist, wie aus den gegebenen Modellen Gleichungssysteme erzeugt werden, die von einem ODE-Solver gelöst werden können. Das muss von dem jeweiligen Simulator bereitgestellt werden. Durch den Anspruch der Allgemeinheit, den Modelica hat, treten bei der Auflösung der Gleichungssysteme einige Probleme auf (z.B. Index-Reduktion), diese werden in den Kapiteln über die einzelnen Simulatoren, die in dieser Arbeit verwendet werden, genauer betrachtet.

## 2.2 Thermodynamik

Thermodynamik beschäftigt sich mit der Möglichkeit durch Umverteilen von Energie in ihre verschiedenen Erscheinungsformen Arbeit zu verrichten. Es gibt zwei Betrachtungsweisen:

- **Statistische Thermodynamik:** Auf mikroskopischer Ebene werden durch Theorien der klassischen Mechanik und der Quantenmechanik Wechselwirkungen zwischen den einzelnen Teilchen (Moleküle, Atome) beschrieben und so auf das Verhalten des Gesamtsystems geschlossen
- **Klassische Thermodynamik:** Auf makroskopischer Ebene werden durch fundamentale Hauptsätze und Bilanzgleichungen zwischen Teilsystemen direkt Zustandsgleichungen für das Gesamtsystem aufgestellt

Da der Zugang des Physical Modellings und der klassischen Thermodynamik sehr ähnlich sind, wird in dieser Arbeit hauptsächlich auf die Theorie der klassischen Thermodynamik Bezug genommen. Es werden zuerst die Grundlagen und klassische Resultate vorgestellt [4] und dann der Bezug zum Physical Modelling und der Modelica Standard Library hergestellt.

### 2.2.1 Systeme

Das zu untersuchende Objekt, das von seiner Umgebung abgegrenzt ist, wird als System, die Hülle des Objekts wird als Systemgrenze bezeichnet. Es werden vier Arten von Systemen unterschieden:

- **Offenes System:** Die Systemgrenze ist durchlässig gegenüber Materie und Wärme
- **Geschlossenes System:** Die Systemgrenze ist undurchlässig gegenüber Materie, lässt aber Wärme durch
- **Adiabates System:** Die Systemgrenze ist undurchlässig gegenüber Wärme, kann aber Materie durchlassen
- **Abgeschlossenes System:** Die Systemgrenze ist undurchlässig sowohl gegenüber Materie als auch Wärme

Ein Gesamtsystem kann in mehrere Teilsysteme aufgeteilt sein, die miteinander verbunden sind.

### 2.2.2 Zustandsgrößen

Um den thermodynamischen Zustand eines Systems zu beschreiben genügen drei Zustandsgrößen, diese werden auch als *thermische Zustandsgrößen* bezeichnet. Außerdem werden noch andere Zustandsgrößen von ihnen abgeleitet, die bei der mathematischen Beschreibung der Zustandsgleichungen und der Hauptsätze sinnvoll sind.

**Volumen:** Das Volumen $V$ gibt die räumliche Ausdehnung des Systems an und schließt die im System enthaltene Masse ein. Die Einheit des Volumens ist *Kubikmeter* [$m^3$].
Dividiert man das Volumen durch die Masse, erhält man das spezifische Volumen $v$. Der Kehrwert des spezifischen Volumens ist die Dichte $\rho$

$$v = \frac{V}{m} = \frac{1}{\varrho} \tag{8}$$

**Druck:** Der innere Druck $p$ ist definiert als Quotient von Druckkraft $F$ und der Fläche $A$, auf die sie wirkt. $F$ steht dabei senkrecht auf $A$.
Im Rahmen der statistischen Thermodynamik, genauer gesagt der *kinetischen Gastheorie*, wird der innere Druck eines Gases folgendermaßen gedeutet:

> Die einzelnen Atome des Gases werden als Kugelmassen betrachtet, die im Raum umherfliegen und beim Auftreffen auf die Gefäßwand reflektiert werden, wobei sie eine Kraft ausüben. Die kumulative Kraft aller Gasatome entspricht dann der Druckkraft $F$.

Die Einheit des Drucks ist *Pascal* [Pa].

**Temperatur:** Die Einheit der thermodynamischen Temperatur $T$ ist *Kelvin* [K]. Um eine Skala zur Messung der Temperatur zu erstellen müssen zwei Bezugspunkte festgelegt werden. Der erste Bezugspunkt ist der absolute Nullpunkt, an dem $T = 0\text{K}$ gilt. Der zweite Bezugspunkt ist der Tripelpunkt des Wassers $T_{tr}$. Dadurch wird ein Temperaturunterschied von einem Kelvin definiert:

$$1\text{K} = \frac{T_{tr}}{273,16} \tag{9}$$

Der Zusammenhang zur Temperatur bezüglich der Celsius-Skala $\vartheta$ ist folgendermaßen festgelegt:

$$\vartheta = T - 273,15\ ^\circ\mathrm{C} \tag{10}$$

Im Gegensatz zu Volumen und Druck (extensive Größen) ist die Temperatur eine intensive Größe, das heißt ihr Zahlenwert kommt nicht durch Addition der Werte aus den Teilsystemen zustande.

**Molare Zustandsgrößen:** Neben den thermischen Zustandsgrößen gibt es noch weitere. Die molaren Zustandsgrößen molare Masse $M$ und das molare Volumen $v_m$ sind Größen die sich auf die Stoffmenge von 1 Mol beziehen und können mit Hilfe der Molzahl $n$ berechnet werden:

$$M = \frac{m}{n} \tag{11}$$

$$v_m = \frac{V}{n} \tag{12}$$

Die molaren Größen sind im allgemeinen materialabhängig.

**Innere Energie:** Jedes System besitzt Energie, die Gesamtenergie $E_g$ eines Systems setzt sich aus seiner inneren Energie $U$, kinetischer Energie $E_{kin}$ und potentieller Energie $E_{pot}$ zusammen:

$$E_g = U + E_{kin} + E_{pot} \tag{13}$$

Die Einheit der Energie ist *Joule* [J]. Neben der inneren Energie kann auch die *spezifische innere Energie* $u$ definiert werden:

$$u = \frac{U}{m} \tag{14}$$

**Enthalpie:** Da in den Energiebilanzen offener Systeme häufig die Summe aus $U$ und dem Produkt aus $p$ und $V$ auftritt, fasst man diese Größen zur Enthalpie $H$ zusammen:

$$H = U + p \cdot V \tag{15}$$

Die *spezifische Enthalpie* $h$ ist analog definiert:

$$h = \frac{H}{m} = u + p \cdot v \tag{16}$$

### 2.2.3 Gleichgewichtszustände

Ein Gleichgewicht bezeichnet im Allgemeinen den Zustand eines Systems, der sich ohne äußere Einwirkung nicht ändert. In der Thermodynamik können vier Gleichgewichte definiert werden:

**Chemisches Gleichgewicht:** Ein System befindet sich im chemischen Gleichgewicht, wenn in seinem Inneren und an den Systemgrenzen keine Stoffumwandlungen durch chemische Reaktionen erfolgen.

**Mechanisches Gleichgewicht:** Systeme sind im mechanischen Gleichgewicht, wenn in ihnen der gleich Druck herrscht. Werden zwei Systeme mit unterschiedlichen Drücken miteinander verbunden, findet ein Druckausgleich statt, bis die Systeme im mechanischen Gleichgewicht sind.

**Thermisches Gleichgewicht:** Das thermische Gleichgewicht wird durch den *Nullten Hauptsatz der Thermodynamik* zusammengefasst:

> *Systeme im thermischen Gleichgewicht haben die selbe Temperatur. Kommen zwei Systeme mit unterschiedlicher Temperatur miteinander in Berührung, so fließt solange Wärme vom wärmeren System zum kälteren, bis beide die gleiche Temperatur haben.*

Dies ist die physikalische Grundlage der Temperaturmessung mit Berührungsthermometern, da sich die Systeme des Thermometers und der Umgebung berühren und irgendwann im thermischen Gleichgewicht befinden.

**Thermodynamisches Gleichgewicht:** Das thermodynamische Gleichgewicht impliziert chemisches, mechanisches und thermisches Gleichgewicht mehrerer Systeme. Das bedeutet, dass im thermodynamischen Gleichgewicht der Druck $p$ und die Temperatur $T$ in jedem Teilsystem gleich groß sind.

### 2.2.4 Zustandsgleichungen

**Thermische Zustandsgleichungen:** Befindet sich ein System im thermodynamischen Gleichgewicht, dann kann es durch die Zustandsgrößen $p$, $v$ und $T$ beschrieben werden. Der Zustand wird durch die implizite Funktion

$$F(p, v, t) = 0 \tag{17}$$

festgelegt, die *Zustandsfunktion* oder *Zustandsgleichung* genannt wird.
Diese Gleichung ist im Allgemeinen sehr kompliziert, kann aber in schmalen Temperaturbereichen relativ einfach beschrieben werden. Vor allem bei der Änderung des Aggregatszustandes eines Stoffes ändert sich das Verhalten der Gleichung stark, weshalb in dieser Arbeit nur Systeme mit festem Aggregatszustand betrachtet werden.

Festkörper und Flüssigkeiten haben im Gegensatz zu Gasen die Eigenschaft, dass sie auch bei hohen Drücken ihr Volumen kaum verändern, wodurch sich die Zustandsgleichung (17) zu

$$F(v, t) = 0 \tag{18}$$

vereinfacht.
Im Allgemeinen kann bei kleinen Temperaturänderungen die Ausdehnung des Volumens als linear angenommen werden, hierfür wird für Feststoffe der Längenausdehnungskoeffizient $\alpha$ und für Flüssigkeiten der Volumenausdehnungskoeffizient $\gamma$ eingeführt. Die Volumenausdehnung errechnet sich dann zu

$$\Delta V = 3 \cdot \alpha \cdot \Delta\vartheta \cdot V_0 \tag{19}$$

für Festkörper, beziehungsweise

$$\Delta V = \gamma \cdot \Delta\vartheta \cdot V_0 \tag{20}$$

für Flüssigkeiten, wobei $V_0$ das Referenzvolumen bezeichnet, das im Normalfall für 0°C oder 20°C angegeben wird.

Vernachlässigt man zusätzlich noch die Temperaturabhängigkeit des Volumens, gelangt man zur besonders einfachen Zustandsgleichung

| Stoff | $\alpha$ | Stoff | $\gamma$ |
|---|---|---|---|
| Aluminium | $23,7 \cdot 10^{-6}$ | Wasser | $20,7 \cdot 10^{-5}$ |
| Eisen | $12,3 \cdot 10^{-6}$ | Ethanol | $110 \cdot 10^{-5}$ |
| Kupfer | $17,0 \cdot 10^{-6}$ | Quecksilber | $18,1 \cdot 10^{-5}$ |
| Silber | $19,7 \cdot 10^{-6}$ | Kerosin | $99 \cdot 10^{-5}$ |

Tabelle 3: Ausdehnungskoeffizienten $\alpha$ und $\gamma$ für einige Stoffe in $\frac{1}{\mathrm{K}}$

$$v = \text{const.} \tag{21}$$

und zum Begriff *idealer Festkörper* und *Flüssigkeiten*.

Die Zustandsgleichung eines Gases wird durch die Kombination zweier empirischer Gesetze hergeleitet, dem Gesetz von Gay-Lussac (22) und dem Gesetz von Boyle-Mariotte (23)

$$\frac{V}{T} = \text{const.} \tag{22}$$

$$p \cdot V = \text{const.} \tag{23}$$

Daraus ergibt sich mit Hilfe des spezifischen Volumens $v$ und der Gaskonstante $R$ die thermische Zustandsgleichung idealer Gase:

$$\frac{p \cdot v}{T} = R \tag{24}$$

Gase, die diesem Gesetz gehorchen heißen ideale Gase. Bei niedrigen Drücken und Dichten können die meisten Gase als ideale Gase angenommen werden, Tabelle 4 zeigt Werte der Gaskonstante für einige wichtige Gase.

| Gas | Gaskonstante $R$ |
|---|---|
| Helium | 2,0770 |
| Wasserstoff | 4,1250 |
| Stickstoff | 0,2968 |
| Sauerstoff | 0,2598 |
| Kohlendioxid | 0,1889 |
| Luft | 0,2872 |

Tabelle 4: Gaskonstanten einiger wichtiger Gase in $\frac{\mathrm{kJ}}{\mathrm{kg \cdot K}}$

Mit Hilfe des Gesetzes von Avogrado

*Gleiche Volumen idealer Gase enthalten bei gleicher Temperatur und gleichem Druck die gleiche Anzahl von Molekülen*

kann eine *universelle Gaskonstante* $R_m$ berechnet werden, wodurch sich (24) durch

$$p \cdot v_m = R_m \cdot T \tag{25}$$

beziehungsweise

$$p \cdot V = n \cdot R_m \cdot T \tag{26}$$

darstellen lässt.

Die universelle Gaskonstante berechnet sich dabei zu

$$R_m = 8314{,}471 \ \frac{\mathrm{J}}{\mathrm{kmol} \cdot \mathrm{K}} \tag{27}$$

**Kalorische Zustandsgleichungen:** Da die thermischen Zustandsgrößen den Zustand des Systems beschreiben, bestimmen sie auch die kalorischen Zustandsgrößen innere Energie $U$ und Enthalpie $H$. Meistens werden ihre spezifischen Größen folgendermaßen angegeben, da aus Gleichung (17) folgt, dass jede der drei Zustandsgrößen als Funktion der beiden anderen Zustandsgrößen ausgedrückt werden kann, falls $F$ für die Zustandsgröße auf eine explizite Form gebracht werden kann, können die beiden Größen als Funktion von zwei Variablen beschrieben werden:

$$u = u(T, v) \tag{28}$$
$$h = h(T, p) \tag{29}$$

Wichtige stoffspezifische Größen sind die partiellen Ableitungen der beiden Funktionen nach der Temperatur $T$, die *spezifische isochore Wärmekapazität*

$$c_\mathrm{v}(T, v) = \frac{\partial u}{\partial T}(T, v) \tag{30}$$

und die *spezifische isobare Wärmekapazität*

$$c_p(T,p) = \frac{\partial h}{\partial T}(T,p) \tag{31}$$

Für ideale Festkörper und Flüssigkeiten ergibt sich aus der Zustandsgleichung (18), dass die innere Energie nur von $T$ abhängt. Weiters ergibt sich ein Zusammenhang zwischen $c_\mathrm{v}$ und $c_p$, nämlich

$$c_\mathrm{v}(T) = c_p(T) = c(T) \tag{32}$$

und es wird die Größe $c(T)$ als *spezifische Wärmekapazität* eingeführt.

| Stoff | spez. Wärmekapazität $c$ |
|---|---|
| Aluminium | 0,896 |
| Kupfer | 0,382 |
| Eisen | 0,452 |
| Silber | 0,235 |
| Wasser | 4,182 |
| Ethanol | 2,43 |
| Quecksilber | 0,139 |
| Kerosin | 1,98 |

Tabelle 5: Spezifische Wärmekapazität $c$ einiger wichtiger Stoffe in $\frac{\mathrm{kJ}}{\mathrm{kg \cdot K}}$

Bei kleinen Temperaturdifferenzen kann die Temperaturabhängigkeit von $c$ oft vernachlässigt werden, woraus sich die Zustandsgleichungen für die Differenz der spezifischen inneren Energie und die spezifische Enthalpiedifferenz zwischen zwei Gleichgewichtszuständen sich zu

$$\Delta u = c \cdot \Delta T \tag{33}$$

und

$$\Delta h = c \cdot \Delta T + v \cdot \Delta p \tag{34}$$

ergeben. In Tabelle 5 sind die spezifischen Wärmekapazitäten einiger Feststoffe und Flüssigkeiten aufgeführt.

Für ideale Gase ergibt sich aus empirischen Versuchen ebenfalls, dass die innere Energie nur von der Temperatur abhängt. Daraus folgt wieder, dass

auch die spezifische isochore Wärmekapazität nur von der Temperatur abhängt. Benutzt man weiters die thermische Zustandsgleichung idealer Gase (24) bei der Definition der Enthalpie ergibt sich durch

$$h = u + p \cdot v = u + R \cdot T \tag{35}$$

dass auch die Enthalpie nur von der Temperatur abhängt. Daraus errechnet sich ein Zusammenhang zwischen $c_\mathrm{v}(T)$ und $c_p(T)$ zu

$$c_p(T) = c_\mathrm{v}(T) + R \tag{36}$$

| Gas | $c_\mathrm{v}$ | $c_p$ |
|---|---|---|
| Helium | 3,1610 | 5,3280 |
| Wasserstoff | 10,0750 | 14,2000 |
| Stickstoff | 0,7422 | 1,0390 |
| Sauerstoff | 0,6552 | 0,9150 |
| Kohlendioxid | 0,6280 | 0,8169 |
| Luft | 0,7168 | 1,0040 |

Tabelle 6: spezifische Wärmekapazitäten $c_\mathrm{v}$ und $c_p$ wichtiger Gase in $\frac{\mathrm{kJ}}{\mathrm{kg \cdot K}}$

Mit Hilfe der kinetischen Gastheorie, die bei der Definition des Drucks bereits erwähnt wurde, ergibt sich die innere Energie eines einatomigen idealen Gases zu

$$U = \frac{3}{2} \cdot m \cdot R \cdot T \tag{37}$$

und daraus

$$c_\mathrm{v} = \frac{3}{2} \cdot R \tag{38}$$

beziehungsweise durch Gleichung (36)

$$c_p = \frac{5}{2} \cdot R \tag{39}$$

Bei zweiatomigen idealen Gasen gilt

$$c_{\mathrm{v}} = \frac{5}{2} \cdot R \qquad \text{bzw.} \qquad c_p = \frac{7}{2} \cdot R \tag{40}$$

Tabelle 6 zeigt die spezifische isochoren und isobaren Wärmekapazitäten von wichtigen Gasen.

### 2.2.5 Hauptsätze der Thermodynamik

Im Folgenden werden die drei Hauptsätze der Thermodynamik aufgeführt. Der 1. Hauptsatz, der auch zentrale Bedeutung im Physical Modelling hat, wird ausführlicher behandelt. Die beiden anderen Hauptsätze sind der Vollständigkeit halber ebenfalls angeführt.

**1. Hauptsatz der Thermodynamik:** Aus dem Energieerhaltungssatz kann der 1. Hauptsatz der Thermodynamik abgeleitet werden. Dieser besagt, dass Energie weder erzeugt noch vernichtet werden kann, sondern nur in eine andere Erscheinungsform umgewandelt wird. Die folgenden Ausführungen beziehen sich auf geschlossene Systeme, für offene Systeme muss noch der Materiefluss über die Systemgrenzen in Betracht gezogen werden.
Wird einem System Energie zugeführt, muss sich in gleichem Maße die Gesamtenergie des Systems erhöhen. Um Energie zuzuführen kann entweder Arbeit $W_{12}$ verrichtet werden oder Wärme $Q_{12}$ zugeführt werden, umgekehrt verringert sich die innere Energie des Systems, wenn ihm Wärme oder Arbeit entzogen wird. Diese Überlegungen führen zu

$$E_{g,2} - E_{g,1} = Q_{12} + W_{12} \tag{41}$$

Setzt man voraus, dass sich das System die ganze Zeit in Ruhe befindet, verändern sich die kinetische und potentielle Energie des Systems nicht und mit (13) erhält man

$$U_2 - U_1 = Q_{12} + W_{12} \tag{42}$$

Die an einem System verrichtete Arbeit $W_{12}$ kann in Volumenänderungsarbeit $W_{\mathrm{v},12}$, Wellenarbeit $W_{\mathrm{W},12}$ und Dissipationsarbeit $W_{\mathrm{d},12}$ aufgeteilt werden. Für die Dissipationsarbeit gilt immer

$$W_{\mathrm{d},12} \geq 0 \tag{43}$$

da sie dem System nur zugeführt werden kann. Außerdem gilt für geschlossene Systeme, dass die Wellenarbeit vollständig dissipiert und deshalb in $W_{\mathrm{d},12}$ enthalten ist. Gilt $W_{\mathrm{d},12} = 0$ nennt man die Zustandsänderung *reversibel*, im Folgenden werden nur reversible Prozesse betrachtet.
Für die Volumenänderungsarbeit gilt

$$W_{\mathrm{v},12} = -\int_{\gamma} p(V,T)\,\mathrm{d}V \tag{44}$$

wobei $\gamma : \mathbb{R} \to \mathbb{R}^2$ ein Weg ist, der von Zustand 1 zum Zustand 2 führt. Das Integral ist im Allgemeinen wegabhängig, das heißt es besitzt keine Stammfunktion und die Arbeit kann deshalb **nicht** in der Form $W_{\mathrm{v},12} = W_{\mathrm{v},2} - W_{\mathrm{v},1}$ geschrieben werden.
Aus diesen Überlegungen folgt für die Wärmemenge die dem System zugeführt werden muss oder entnommen werden kann

$$Q_{12} = U_2 - U_1 + \int_{\gamma} p(V,T)\,\mathrm{d}V \tag{45}$$

Für isochore Prozesse, also Prozesse bei denen das Volumen konstant bleibt, und isobare Prozesse, bei denen der Druck konstant ist, lässt sich die Wärmemenge besonders einfach darstellen, hier gilt für isochore Prozesse:

$$Q_{12} = U_2 - U_1 = m \cdot c_{\mathrm{v}} \cdot (T_2 - T_1) \tag{46}$$

Und für isobare Prozesse:

$$Q_{12} = H_2 - H_1 = m \cdot c_p \cdot (T_2 - T_1) \tag{47}$$

Unter der Voraussetzung, dass $c_{\mathrm{v}}$ und $c_p$ konstant sind.

**2. Hauptsatz der Thermodynamik:** Für den 2. Hauptsatz der Thermodynamik gibt es viele Formulierungen, die in ihrem Kern alle das gleiche aussagen:

> *Wärme kann nie von selbst von einem System niederer Temperatur auf ein System höherer Temperatur übertragen werden*

Eine Konsequenz daraus ist, dass Wärme niemals vollständig in Arbeit überführt werden kann.

**3. Hauptsatz der Thermodynamik:** Der *Nernstsche Satz*, der oft als 3. Hauptsatz der Thermodynamik bezeichnet wird, besagt folgendes:

*Der absolute Nullpunkt ist unerreichbar*

An dieser Stelle sei erwähnt, dass durch den Zusammenhang zwischen innerer Energie und Temperatur aus den Gleichungen (33) und (37) folgt, dass bei 0K auch die innere Energie 0 wird, das heißt, dass die Atome und Moleküle in dem System keine Bewegungsenergie mehr haben, also komplett still stehen.

### 2.2.6 Wärmeübertragung

Werden zwei Systeme mit unterschiedlichen Temperaturen miteinander verknüpft, fließt nach dem 2. Hauptsatz Wärme vom wärmeren System in das kältere. Dieser Prozess benötigt Zeit, weshalb es sinnvoll ist, die Ableitung der über die Systemgrenze transportierten Wärme nach der Zeit, den *Wärmestrom* zu definieren.

$$\dot{Q} = \lim_{\Delta t \to 0} \frac{\Delta Q}{\Delta t} \tag{48}$$

Für die Gesamtwärme $Q_{12}$, die während des Prozesses von Zustand 1 zu Zustand 2 zwischen den zwei Systemen fließt, gilt dann

$$Q_{12} = \int_{t_1}^{t_2} \dot{Q}(t)\,\mathrm{d}t \tag{49}$$

Es werden drei Arten der Wärmeübertragung unterschieden:

**Konduktive Wärmeübertragung:** Wärmeleitung bezeichnet den Transport von Wärme durch ruhende, feste, flüssige oder gasförmige Schichten. Für den Wärmestrom gilt

$$\dot{Q} = -\lambda \cdot A \cdot \frac{T_2 - T_1}{\delta} \tag{50}$$

wobei $T_i$ die Temperatur im System $i$, $\delta$ die Dicke der wärmeleitenden Schicht, $A$ die Fläche zwischen den Systemen und $\lambda$ eine stoffspezifische Größe, den *Wärmeleitkoeffizienten* bezeichnet.

**Konvektive Wärmeübertragung:** Die Wärmeübertragung zwischen einer Wand und einem Fluid (flüssiges oder gasförmiges Medium) gehört zum Mechanismus der konvektiven Wärmeübertragung, für den Wärmestrom gilt

$$\dot{Q} = \alpha \cdot A \cdot (T_{\mathrm{w}} - T_{\mathrm{fl}}) \tag{51}$$

wobei $T_{\mathrm{w}}$ die Oberflächentemperatur der Wand, $T_{\mathrm{fl}}$ die mittlere Temperatur des Fluids, $A$ die Fläche zwischen den beiden Systemen bezeichnet. Der *Wärmeübergangskoeffizient* $\alpha$ hängt in komplexer Weise vom Strömungsverlauf und stoffspezifischen Parametern des Fluids ab, ist im Allgemeinen also sogar eine ortsabhängige Größe.

**Wärmeübertragung durch Strahlung:** Für die Wärmestromdichte, die als Wärme von einem Körper abgestrahlt wird, ergibt sich aus dem *Stefan-Boltzmannschen Strahlungsgesetz*:

$$\dot{q} = \varepsilon \cdot \sigma \cdot T^4 \tag{52}$$

Hier ist $0 \leq \varepsilon \leq 1$ der Emissionsgrad des Körpers, der neben dem Material auch von der Beschaffenheit der Oberfläche abhängt, und $\sigma$ die *Stefan-Boltzmann Konstante*. Für $\sigma$ gilt

$$\sigma = 5{,}67051 \cdot 10^{-8} \ \frac{\mathrm{W}}{\mathrm{m^2K^4}} \tag{53}$$

Durch Integration der Wärmestromdichte über die gesamte Oberfläche, wobei diese als konstant angenommen wird, erhält man:

$$\dot{Q} = A \cdot \varepsilon \cdot \sigma \cdot T^4 \tag{54}$$

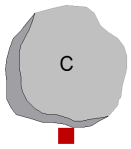
Diese Formel gilt nur, wenn der strahlende Körper nicht von anderen Körpern angestrahlt wird, gibt es mehrere Körper die sich gegenseitig beeinflussen verkompliziert sich die obige Formel.

### 2.2.7 Die HeatTransfer-Library in Modelica

Wie aus Tabelle 1 ersichtlich ist, wird in der hydraulischen Domäne beim Physical Modelling die Temperatur $T$ als Effort- und der Wärmestrom $\dot{Q}$ als

Flow-Variable verwendet.
In der *Thermal*-Library sind die *FluidHeatFlow*-Library, mit der Heiz- und Kühlkreisläufe aufgebaut werden können, und die *HeatTransfer*-Library, in der Grundkomponenten zur Modellierung des Wärmetransportes innerhalb fester, flüssiger oder gasförmiger Stoffe hinterlegt sind. Die Grundkomponenten der HeatTransfer-Library werden im Folgenden genauer erklärt.

**HeatCapacitor:** In dieser Komponente werden die wärmespeichernden Eigenschaften eines Materials modelliert.




```
T = port.T;
der_T = der(T);
C*der(T) = port.Q_flow;
```

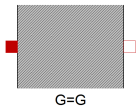
Abbildung 14: Graphische Repräsentation und Gleichungen der Heat Capacitor Komponente

Der Parameter `C` aus Abbildung 14 beschreibt die Wärmekapazität des betrachteten Objekts. Es werden keine Annahmen über die Geometrie getroffen. Zur Berechnung wird die spezifische isobare Wärmekapazität $c_p$ (Tabellen 5 und 6) herangezogen

$$\mathtt{C} = c_p \cdot m \tag{55}$$

In dem Volumen das die Komponente beschreibt ist die Temperatur räumlich konstant.

**ThermalConductor:** Hier wird der Wärmetransport durch ein Medium berechnet.




```
Q_flow = G*dT;
```

Abbildung 15: Graphische Repräsentation und Gleichung der Thermal Conductor Komponente

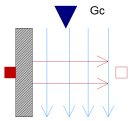
Für komplizierte Geometrien muss der Parameter `G`, der die thermische Konduktivität des Objekts beschreibt, gemessen werden, für einfache Geometrien

kann er auch berechnet werden. Zum Beispiel ist für einen Quader mit Länge $\delta$, Grundfläche $A$ und Wärmeleitkoeffizient $\lambda$ (vergleiche (50)), wobei die Wärmeleitung in Längsrichtung erfolgt,

$$\texttt{G} = \lambda \cdot \frac{A}{\delta} \tag{56}$$

Das `dT` in der Gleichung aus Abbildung 15 beschreibt die Temperaturdifferenz zwischen den zwei Ports.

**Convection:** Die Konvektions-Komponente berechnet den linearen konvektiven Wärmetransport zwischen zwei Materialien, also zum Beispiel zwischen einer Wand und der Luft, die sie umgibt.




```
dT = solid.T - fluid.T;
solid.Q_flow = Q_flow;
fluid.Q_flow = -Q_flow;
Q_flow = Gc*dT;
```

Abbildung 16: Graphische Repräsentation und Gleichungen der Convection Komponente

Den Parameter `Gc` kann man für einfache Geometrien folgendermaßen berechnen:

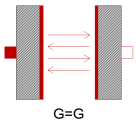
$$\texttt{Gc} = \alpha \cdot A \tag{57}$$

Wie bereits bei der Beschreibung des konduktiven Wärmetransports in Kapitel 2.2.6 beschrieben wurde, ist der Wärmeübergangskoeffizient $\alpha$ nicht konstant, weswegen `Gc` auch über den Signaleingang der Komponente beschrieben wird und somit zeitlich veränderlich sein kann.

**BodyRadiation:** In dieser Komponente wird der Wärmetransport durch Wärmestrahlung berechnet.

Durch Vergleich mit Gleichung (54) ergibt für den Parameter `Gr`

$$\texttt{Gr} = \varepsilon \cdot A \tag{58}$$

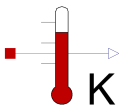



```
Q_flow = Gr*sigma*(port_a.T^4 - port_b.T^4);
```

Abbildung 17: Graphische Repräsentation und Gleichung der Body Radiation Komponente

für einen einzelnen Strahler. Die Konstante `sigma` beschreibt die Stefan-Boltzmann-Konstante wie in Gleichung (53) und ist als Konstante in der Modelica Standard Library hinterlegt.

**TemperatureSensor:** Diese Komponente beschreibt ein ideales Thermometer.

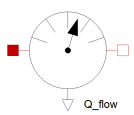



```
T = port.T;
port.Q_flow = 0;
```

Abbildung 18: Graphische Repräsentation und Gleichungen des Temperature Sensors

Dadurch, dass $\dot{Q} = 0$ gesetzt wird, wird gewährleistet, dass das Thermometer dem System keine Wärme entzieht, also ideal ist. Der Output `T` kann als Signal weiterverwendet werden, um zum Beispiel als Regelgröße für einen Temperaturregler zu dienen.

**HeatFlowSensor:** Hier wird der Wärmefluss der durch den Sensor geht gemessen, das heißt, dass der Heat Flow Sensor seriell in das Blockschaltbild eingehängt werden muss. Dabei wird die Temperatur an den beiden Ports der Komponente gleich gehalten, sodass keine Energie in der Komponente verloren geht.

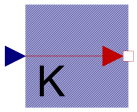



```
port_a.T = port_b.T;
port_a.Q_flow + port_b.Q_flow = 0;
Q_flow = port_a.Q_flow;
```

Abbildung 19: Graphische Repräsentation und Gleichungen des Heat Flow Sensors

Der Output `Q_flow` ist positiv, falls die Wärme von Port `a` zu Port `b` fließt.

**PrescribedTemperature:** Diese Komponente stellt eine variable Temperatur Randbedingung dar.




```
Modelica.Blocks.Interfaces.RealInput T
port.T = T;
```

Abbildung 20: Graphische Repräsentation und Gleichungen der Prescribed Temperature Komponente

Der Input `T` kann über die Zeit variabel sein. es gibt auch noch die Komponente FixedTemperature, bei der `T` ein Parameter und somit konstant ist.

**PrescribedHeatFlow:** In dieser Komponente wird eine variable Wärmequelle dargestellt.

```
Modelica.Blocks.Interfaces.RealInput Q_flow
port.Q_flow = -Q_flow*(1+alpha*(port.T - T_ref));
```

Abbildung 21: Graphische Repräsentation und Gleichungen der Prescribed Heat Flow Komponente

Der Input `Q_flow` kann über die Zeit variabel sein. es gibt auch noch die Komponente FixedHeatFlow, bei der `Q_Flow` ein Parameter und somit konstant ist. Durch den Parameter `alpha` $\geq 0$ können auch temperaturabhängige Energieverluste mitmodelliert werden.

Es gibt noch weitere Komponenten in der HeatFlowLibrary, unter anderem werden Komponenten bereitgestellt, die Temperaturen aus der Kelvin-Skala in die Celsius-, Fahrenheit- oder Rankine-Skala umrechnen, außerdem gibt es noch PrescribedTemperature- und TemperatureSensor-Komponenten für die drei erwähnten Temperaturskalen.

## 2.3 Kopplung von Modellen

Fortschritte in der Computertechnik, wodurch immer leistungsfähigere Rechner zur Verfügung stehen, machen es möglich immer größere Simulationsmodelle zu erstellen und zu simulieren. In diesem Zusammenhang ist die Kopplung von Modellteilen ein wichtiges Thema, da dadurch bestehende Modelle als Modellteile weiter benutzt werden können und außerdem die Validierung

eines sehr großen Modells zusätzliche Schwierigkeiten birgt.
Bei der Kopplung verschiedener Modellteile sind drei Vorgangsweisen denkbar:

- Kopplung mit Hilfe von Eingangsdaten
- Single-Solver Simulation
- Co-Simulation bzw. Multirate-Simulation

All diese Vorgangsweisen haben ihre Vor- und Nachteile, die im Folgenden kurz beschrieben werden, wobei ein Hauptaugenmerk auf die Single-Solver Simulation gelegt wird, da diese in dieser Arbeit genauer untersucht wird.

**Kopplung mit Hilfe von Eingangsdaten:** Bei dieser Art der Kopplung dürfen die einzelnen Teilmodelle sich nicht zu sehr beeinflussen. Die einzelnen Teilmodelle werden separat simuliert, wobei die Ergebnisse einzelner Modelle als Eingangsdaten für andere Modellteile verwendet werden können, zum Beispiel mit Hilfe von LookUp-Tables.
Dieser Zugang ist sehr einfach zu modellieren, erfordert aber, dass es keine Rückkopplung zwischen einzelnen Modellteilen gibt, sondern die Abhängigkeit zwischen zwei Teilmodellen höchsten in eine Richtung geht.

**Single-Solver Simulation:** Hier werden die einzelnen Teilmodelle zu einem Gesamtmodell zusammengefügt und dann in einem einzigen Simulator simuliert. Es wird also ein großes DAE-System aufgebaut, das dann mit einem Solver gelöst wird.
Dabei können Probleme auftreten, falls die einzelnen Systeme zu unterschiedliche *Zeitkonstanten* haben oder das Gesamtsystem einen zu hohen *Index* hat. Diese beiden Begriffe werden im Weiteren genauer erklärt.

**Co-Simulation/Multirate-Simulation:** Bei der Co-Simulation werden einzelne Modellteile in verschiedenen Simulatoren modelliert, wobei sich die Teilmodelle gegenseitig beeinflussen können. Dazu ist notwendig, dass es einen sogenannten *Backbone* gibt, der die Kommunikation zwischen den einzelnen Modellteilen steuert und den Austausch von Informationen zu bestimmten Zeitpunkten veranlasst.
Das Prinzip der Multirate-Simulation ist ähnlich. Der Unterschied liegt darin, dass die Modelle alle in einem Simulator aufgebaut werden, wobei es einen speziellen *Multirate-Solver* gibt, der die Solver, die die Lösungen der einzelnen Teilmodelle berechnen, koordinieren muss.

Die größten Probleme liegen hier, bei der Kommunikation zwischen den Teilmodellen. Der Backbone beziehungsweise der Multirate-Solver müssen Zeitpunkte vorgeben, an denen ein Informationsaustausch stattfindet, an diesen Zeitpunkten müssen die jeweiligen Solver deshalb auch Ergebnisse liefern. Das heißt in der Regel, dass die Lösung an diesem Zeitpunkt ausiteriert werden muss.

### 2.3.1 Zeitkonstante

Die Zeitkonstante eines Systems bestimmt im Wesentlichen die Zeit, die ein System braucht um auf einen Sprung im Eingangssignal zu reagieren. Der Begriff stammt aus der Elektrotechnik. Betrachtet man zum Beispiel das RL-Glied aus Abbildung 2 beziehungsweise die dazugehörige Differentialgleichung (1) und der Anfangsbedingung $i(0) = 0$, das heißt das die Spule am Anfang komplett entladen ist, so kann die Anfangswertproblem eindeutig gelöst werden, mit

$$i(t) = \frac{U}{R}\left(1 - e^{-\frac{R}{L}t}\right) \tag{59}$$

Vergleicht man diese Lösung mit der allgemeinen Form der Lösung

$$i(t) = i_{max}\left(1 - e^{-\frac{t}{\tau}}\right) \tag{60}$$

in der $\tau$ die Zeitkonstante ist, ergibt sich

$$\tau = \frac{L}{R} \tag{61}$$

Analog können auch für andere physikalische Domänen Zeitkonstanten eingeführt werden. Betrachtet man in der Thermodynamik das System aus einem Wärmespeichers mit Wärmekapazität $C$ (vgl. (55)) , aus dem Wärme durch Wärmeleitung über eine Wand mit thermischer Konduktivität $G$ (vgl. (56)) entzogen wird, so ergibt sich

$$\tau_{th} = \frac{C}{G} = \frac{\delta \cdot m \cdot c_p}{A \cdot \lambda} \tag{62}$$

Sind die Zeitkonstanten in verschiedenen Teilsystemen zu unterschiedlich, kommt es bei der Singlesolver-Simulationen zu Problemen, da der Solver zur

Auflösung des Verhaltens des schnellen Systems eine sehr kleine Schrittweite wählen muss, die in dem langsamen System aber zu zusätzlichen numerischen Fehlern führen kann.

### 2.3.2 Index

Wird eine differentialalgebraische Gleichung (DAE) in ihrer impliziten Form beschrieben

$$F(t, u, u') = 0 \tag{63}$$

so ist der Index der DAE $d$ definiert als

> $d \in \mathbb{N}$ ist die kleinste Zahl für die der Ableitungsvektor $u'(t)$ durch die $d+1$ Gleichungen:
>
> $$\frac{\mathrm{d}^i}{\mathrm{d}t^i} F(t, u, u') = 0 \qquad i = 0, \ldots, d$$
>
> eindeutig als System erster Ordnung in Ausdrücken von $u(t)$ bestimmt ist.

Trennt man den Zustandsvektor $u$ in seine differentiellen Zustände $y$ und seine algebraischen Zustände $z$ kommt man zur semiexpliziten Form des DAEs:

$$y' = f(t, y, z) \tag{64}$$
$$0 = g(t, y, z) \tag{65}$$

Hier muss natürlich nur noch die Funktion $g$ differenziert werden, um den Index zu bestimmen. Weiters erkennt man, dass falls die Jacobimatrix von $g$ regulär ist, das System Index 1 hat. Daraus folgt, dass Index 1 Probleme aber auch direkt lösbar sind indem die algebraischen Zustände mit Hilfe des Newton-Verfahrens berechnet werden. Es gibt auch noch Lösungsstrategien für Index 2 Probleme, für alle Probleme die einen höheren Index haben, muss aber das System selbst verändert werden um es zu lösen.
Ein weit verbreitetes Vorgehen dabei ist die *Indexreduktion*. Dabei wird die algebraische Zustandsgleichung solange symbolisch differenziert bis ein Index 1 Problem vorliegt, das dann wie oben erwähnt numerisch gelöst werden kann. Ein Problem das dabei auftritt ist der *Drift-Off-Effekt*, das bedeutet,

dass die algebraischen Zwangsbedingungen durch numerische Fehler nicht genau eingehalten werden. Durch die zusätzlichen Gleichungen die durch das symbolische Differenzieren gewonnen wurden verstärkt sich dieser Effekt, da sich die numerischen Fehler in den einzelnen Gleichungen aufsummieren.
Koppelt man nun mehrere Systeme die verschiedenen algebraischen Zwangsbedingungen unterliegen, ist es sehr schwer vorherzusagen wie sich der Index des Gesamtsystems verhält und das System wird möglicherweise numerisch instabil.

# 3 Aufgabenstellung

In diesem Kapitel werden die Teilmodelle, die gekoppelt werden sollen, vorgestellt. Zusätzlich werden noch die Versuchsaufbauten, die zur Validierung der Teilmodelle dienen, beschrieben.

## 3.1 Der FILL-Versuchsstand

Die Linearführung, die als einfaches Maschinenmodell dient, wurde als Projekt im Rahmen einer Bachelorarbeit am Institut für Fertigungstechnik und Hochleistungslasertechnik der Technischen Universität Wien aufgebaut [5].

Abbildung 22: Der Fill-Versuchsstand aus zwei verschiedenen Blickwinkeln

Die Linearführung besteht aus einer Spindel, auf der ein Führungstisch sitzt, die über einen Zahnriemen von einem Elektromotor angetrieben wird. Eine weitere wichtige Komponente ist die Steuereinheit, die über einen Kaskadenregler den Versuchsstand steuert. Die Einzelteile des Versuchsaufbaus wurden größtenteils aus alten Werkzeugmaschinen entnommen.
Für den Aufbau eines Modells sind natürlich die Parameter die das System beschreiben von Interesse. Im Folgenden werden die einzelnen Komponenten, die im Modell abgebildet werden und ihre bestimmenden Parameter beschrieben.

### 3.1.1 Elektromotor

Der in dem Versuchsaufbau verwendete Elektromotor stammt aus der Baureihe 1FT6 von Siemens, die mehrere permanenterregte Synchronmotoren beinhaltet. Die Parameter für das Modell stammen aus den Datenblättern für diese Baureihe [6].

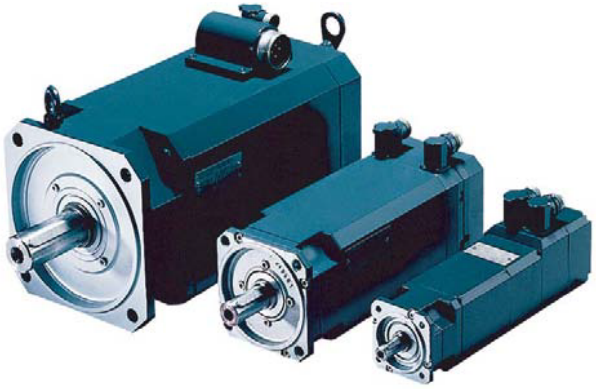


Abbildung 23: 1FT6-Synchronmotoren von Siemens

Die genaue Bezeichnung des verwendeten Motors lautet *1FT6044-4AF71-3AA1*. In Tabelle 7 werden die Parameter die zur Modellierung verwendet werden aufgelistet.

| Technische Daten | Kurzbezeichnung | Einheit | Wert |
|---|---|---|---|
| Trägheitsmoment | $J_{mot}$ | $10^{-4} \cdot \mathrm{kg} \cdot \mathrm{m}^2$ | $6,18$ |
| Bemessungsstrom | $I_N$ | A | $2,9$ |
| Bemessungsdrehzahl | $n_N$ | 1/min | 3000 |
| Wicklungswiderstand bei 20°C | $R_{str}$ | $\Omega$ | $3,05$ |
| Drehfeldinduktivität | $L_D$ | mH | 16 |

Tabelle 7: Motordaten aus dem Datenblatt des Synchronmotors

In der Modelica-Standard-Library wird ein Modell eines permanenterregten Synchronmotors zur Verfügung gestellt, das in Kapitel 4.3.1 genauer vorgestellt wird.

### 3.1.2 Zahnriemengetriebe

Der verwendete Zahnriemen hat die Bezeichnung *PU 25T10-880 Z88* und ist aus Polyurethan gefertigt. Die zur Beschreibung des Zahnriemengetrie-

bes verwendeten Parameter sind die Übersetzungszahl $i$, die sich aus dem Verhältnis der Zähnezahlen der beiden Zahnräder, die der Riemen verbindet ergibt

$$i = \frac{z_2}{z_1} \tag{66}$$

und die Federkonstante $c_R$ des Zahnriemens.
Für die Übersetzungszahl wurde $i = 2$ verwendet. Da in der Produktbeschreibung des verwendeten Zahnriemens die Federkonstante nicht aufgelistet ist, wurde die Federkonstante eines ähnlichen Zahnriemens verwendet:

$$c_R = 2 \cdot 10^7 \frac{\mathrm{N}}{\mathrm{m}} \tag{67}$$

### 3.1.3 Gewindestange

Die Gewindestange wurde für den Versuchsaufbau aus einer alten Werkzeugmaschine entnommen und aufbereitet. Die für das Modell relevanten Parameter sind die Steigung $h$, die den abstand zweier Gewindestufen entlang der Gewindeachse und somit den Weg, der bei einer Umdrehung zurückgelegt wird, beschreibt und das Trägheitsmoment der Gewindestange. Die Werte, die im Modell verwendet wurden, sind:

$$h = 30\mathrm{mm} \tag{68}$$

$$J_G = 7,01 \cdot 10^{-3}\mathrm{kg} \cdot \mathrm{m}^2 \tag{69}$$

### 3.1.4 Führungstisch

Der Führungstisch wurde ebenfalls aus einer alten Werkzeugmaschine entnommen und aufbereitet. Auf dem Tisch kann zusätzlich eine Masse befestigt werden. Für die Modellierung wurde eine Gesamtmasse aus Tisch und Masse $m_{ges} = 555,5\mathrm{kg}$ verwendet.

### 3.1.5 Leistungselektronik

Die *SINUMERIK 840D* übernimmt die Regelung des Verfahrvorganges. Die Regelung erfolgt durch einen Kaskadenregler, das heißt das mehrere Regelkreise ineinander verschachtelt sind. Abbildung 24 zeigt das Blockschaltbild eines einfachen Kaskadenregelkreises, der vom Grundprinzip her dem Regelkreis in der Regelung des Versuchsstandes sehr ähnlich ist.

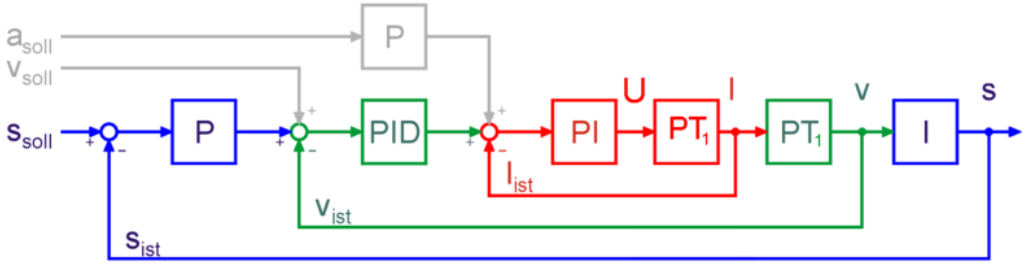



Abbildung 24: Blockschaltbild eines Kaskadenregelkreises

Das Grundprinzip, dass über den Sollwert der Position $s$, die Geschwindigkeit $v$ und dann der Strom $I$ beziehungsweise die Spannung $U$ geregelt wird, ist gleich, allerdings werden in der echten Schaltung auch noch einige Filter und Begrenzer verwendet. Außerdem ist der komplette Aufbau des Reglers natürlich nicht vollständig dokumentiert, sodass ein Nachbauen des Reglers sehr schwierig wird. Deshalb wurde in dieser Arbeit darauf verzichtet. Stattdessen werden als Eingangsdaten für die Modelle Messdaten verwendet.

## 3.2 Das Kompartment-Modell des Raumes

Wie bereits in Kapitel 2.2.7 gezeigt wurde, kann in Modelica ein Raum nur durch seine Wärmekapazität $C$ dargestellt werden. Innerhalb dieses Volumens ist die Temperatur an jedem Punkt gleich. Betrachtet man nun eine Maschinenhalle ist die Annahme, dass die Temperatur über den Raum konstant ist, sehr unrealistisch. Deshalb wird in dieser Arbeit der Raum durch Kompartments aufgeteilt, die sich gegenseitig beeinflussen können, indem Wärme zwischen angrenzenden Kompartments fließt. Abbildung 25 zeigt eine schematische Darstellung dieser Aufteilung, wobei die roten Pfeile die möglichen Wege des Wärmeflusses darstellen.

Es werden noch weitere Modellannahmen getroffen, die das Modell vereinfachen, um wirklich nur die Auswirkung der Kopplung auf das Gesamtmodell betrachten zu können:

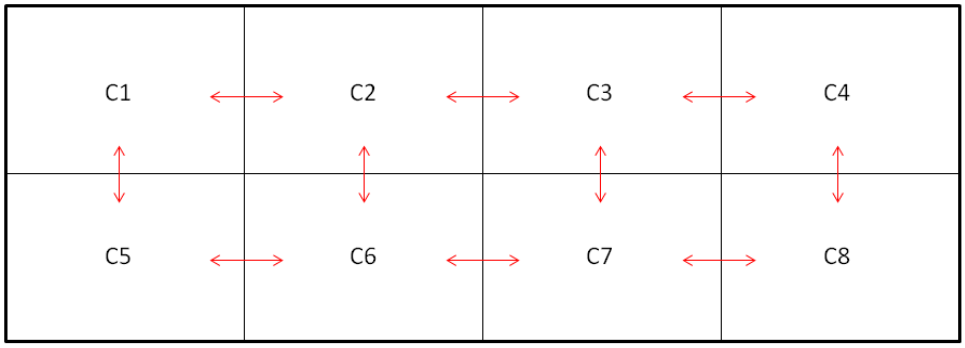



Abbildung 25: 2-dimensionale Aufteilung eines Raumes in Kompartments

- Die Wände des Raumes sind vollständig isoliert, das heißt es tritt kein Wärmeaustausch zwischen einem Kompartment und der angrenzenden Wand auf. Weiters folgt daraus, dass die gesamte Energie im System erhalten bleibt.
- Die einzige Form des Wärmetransports ist Wärmeleitung, Konvektion und Wärmestrahlung werden in dem Modell nicht in Betracht gezogen.
- Die Parameter in dem Modell sind alle temperaturunabhängig, dies gilt im weiteren Verlauf auch für die Parameter des Maschinenmodells beim gekoppelten System.
- Das Medium, aus dem das Kompartment besteht, ist isotrop, hat also an jeder Stelle die gleichen materialspezifischen Eigenschaften.

Zur Vereinfachung des Modellaufbaus wird für die einzelnen Kompartments eine eigene Komponente in Modelica erstellt, diese wird im Folgenden vorgestellt.

### 3.2.1 Kompartment

Im Gegensatz zum Schema in Abbildung 25 wird eine Aufteilung des Raumes in nur zwei Dimensionen im Allgemeinen zu wenig sein, deshalb werden die Kompartment-Komponenten so angefertigt, dass sie auch für eine Aufteilung in alle drei Dimension verwendet werden können. Abbildung 26 zeigt den Aufbau des Kompartments in Dymola. Neben der HeatCapacitor-Komponente, die die Temperatur in dem Kompartment bestimmt, werden ThermalConductor-Komponenten, die Wärme in jede der sechs möglichen

Richtungen leiten, verwendet. Weiters wird ein Temperatursensor verwendet um die Simulationsergebnisse leichter ersichtlich zu machen.

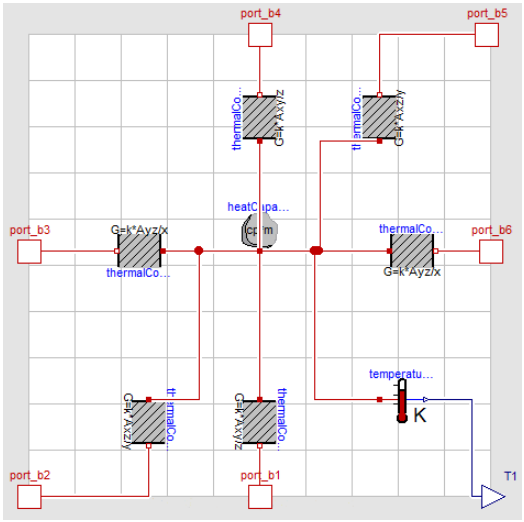



Abbildung 26: Modell des Kompartments in Dymola

Die Parameter die in der Komponente verwendet werden, sind in Tabelle 8 ersichtlich. Einige der Parameter werden aus anderen Parametern berechnet, dies wird in der Tabelle zusätzlich angeführt. Weiters wird die Einheit, in der die Variable angegeben werden muss und eine kurze Beschreibung der Variable aufgeführt.

| | | | |
|---|---|---|---|
| $x$ | | m | Ausdehnung in $x$-Richtung |
| $y$ | | m | Ausdehnung in $y$-Richtung |
| $z$ | | m | Ausdehnung in $z$-Richtung |
| $V$ | x $\cdot y \cdot z$ | $m^3$ | Volumen |
| $A_{xy}$ | $x \cdot y$ | $m^2$ | Fläche des Kompartmens parallel zur $xy$-Ebene |
| $A_{xz}$ | $x \cdot z$ | $m^2$ | Fläche des Kompartmens parallel zur $xz$-Ebene |
| $A_{yz}$ | $y \cdot z$ | $m^2$ | Fläche des Kompartmens parallel zur $yz$-Ebene |
| $\rho$ | | $\frac{kg}{m^3}$ | Dichte des Mediums |
| $m$ | $V \cdot \rho$ | kg | Gesamtmasse des Kompartments |
| $c_p$ | | $\frac{J}{kg \cdot K}$ | spezifische Wärmekapazität des Mediums |
| $k$ | | $\frac{W}{m \cdot K}$ | Wärmeleitkoeffizient des Mediums |

Tabelle 8: Parameterliste für die Kompartment-Komponente

Für die Wärmekapazität $C$ gilt nun analog zu Gleichung (55)

$$C = c_p \cdot m \tag{70}$$

Für die thermische Konduktivität $G$ wird angenommen, dass die Wärme vom Mittelpunkt des Kompartments ausgeht, dass die Wärme also durch die Hälfte des Kompartments durchfließen muss, bevor es zum nächsten Kompartment kommt, dies liefert analog zu Gleichung (56)

$$G = k \cdot \frac{2A_{yz}}{x} \tag{71}$$

für die Wärmeleitung in Richtung der $x$-Koordinate. Die anderen Richtungen werden natürlich analog berechnet.

### 3.2.2 Versuchsaufbau

Das Modell des Raumes in seiner vereinfachten Form wird durch einen selbstgebauten Versuchsaufbau validiert. Dieser besteht aus einer Styroporbox mit den Abmessungen $53,5\text{cm} \times 34\text{cm} \times 24\text{cm}$, die mit Hilfe eines Lötkolbens aufgeheizt wird. Zur Temperaturmessung wurden programmierbare digitale Temperatursensoren von MAXIM mit der Bezeichnung *DS18B20* verwendet [7]. Abbildung 27 zeigt den Versuchsaufbau.

Abbildung 27: Versuchsaufbau zur Valdierung des Raummodells

Zur Aufzeichnung der Daten wurde ein *Arduino Uno*-Mikrocontroller verwendet [8].

**Mikrocontroller:** Arduino-Mikrocontroller sind Open-Source Prototyping Plattformen, es können sowohl Daten über Sensoren empfangen werden, als auch Lampen oder kleine Elektromotoren damit gesteuert werden.

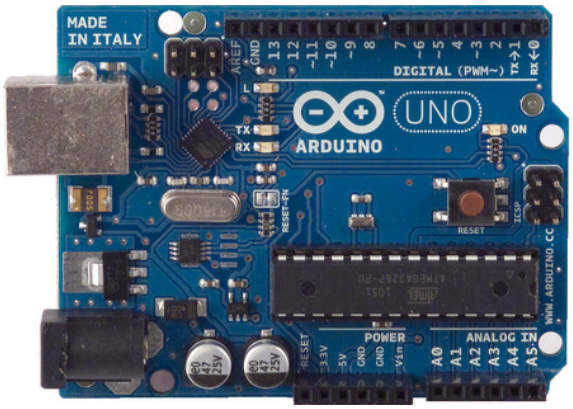



Abbildung 28: Mikrocontroller Arduino Uno

Zum Programmieren der Hardware wird auch eine Entwicklungsumgebung zur Verfügung gestellt. Mit Hilfe dieser Entwicklungsumgebung, der *Arduino Development Environment* können Programme in einer eigens für den Controller entwickelten Programmiersprache, der *Arduino Programming Languange*, erstellt werden und dann auf den Controller hochgeladen werden.

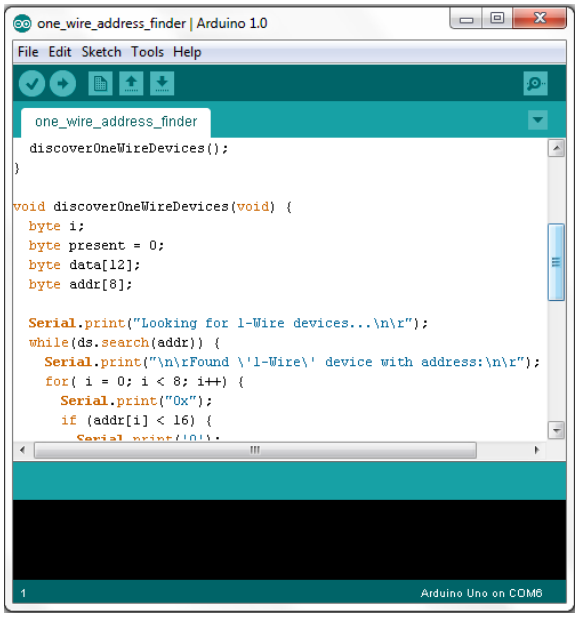



Abbildung 29: Screenshot der Arduino Development Environment

**Temperatursensoren:** Mit den Temperatursensoren können Temperaturen zwischen $-55°C$ und $125°C$ mit einer Genauigkeit von $\pm 0,5°C$ gemessen werden. Die Datenübertragung zum Mikrocontroller erfolgt über einen seriellen Eindraht-Bus.

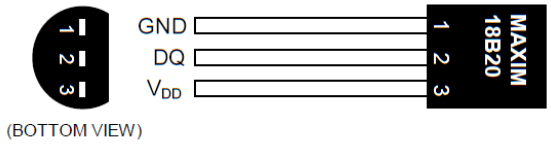



Abbildung 30: Darstellung des Temperatursensors DS18B20

Die drei Pins des Temperatursensors sind folgendermaßen zu verwenden:

- `VDD`: Hier kann eine optionale Stromversorgung angehängt werden. Der Sensor kann auch im Parasiten-Modus betrieben werden, das heißt, dass der Sensor den Strom auch über die Datenleitung bezieht. In diesem Fall muss der `VDD` ebenfalls geerdet werden.
- `DQ`: Daten Eingang/Ausgang, kann auch zur Stromversorgung benutzt werden.
- `GND`: Masse

Abbildung 31 zeigt ein Blockschaltbild des Aufbaus des Temperatursensors.

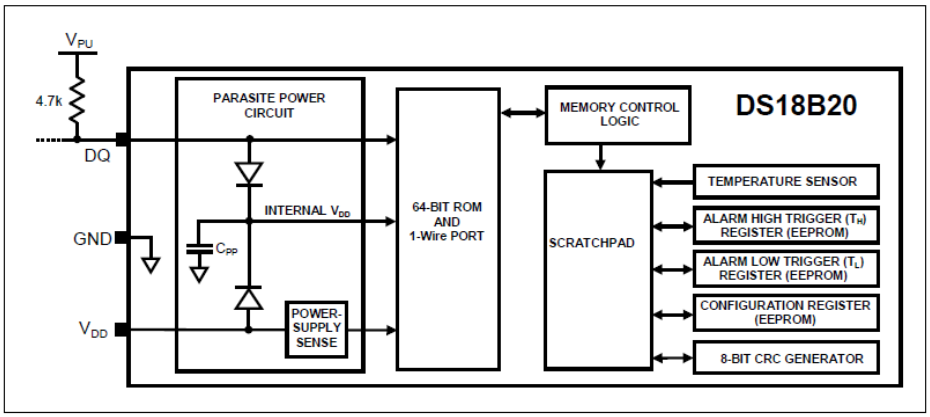



Abbildung 31: Blockschaltbild des Temperatursensors DS18B20

In dem 64-bit ROM Speicher wird die einzigartige Adresse des Temperatursensors gespeichert, dadurch können mehrere Sensoren in Serie geschaltet werden und trotzdem einzeln vom Mikrocontroller angesprochen werden.

**Schaltung:** Der Versuchsaufbau wurde in acht Kompartments unterteilt, deshalb mussten acht Temperatursensoren zur Messung in den einzelnen Kompartments angebracht werden, da keine externe Stromversorgung vorgesehen war, wurden die $V_{DD}$-Anschlüsse der einzelnen Sensoren ebenfalls geerdet. Das Blockschaltbild dieser Schaltung wird in Abbildung 32 gezeigt.

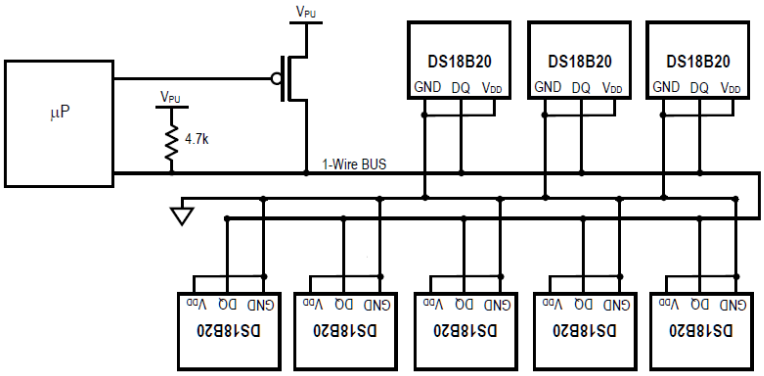



Abbildung 32: Blockschaltbild der Verschaltung der Temperatursensoren des Versuchsaufbaus

**Programmierung:** Um die Daten auch auslesen zu können, mussten zuerst die einzelnen Adressen der Temperatursensoren abgefragt werden, ein fertiges Programm dazu ist im Internet frei erhältlich [9], der Code dazu ist im Appendix in Abschnitt A.1 zu finden. Dieses Programm wurde auf den Mikrocontroller geladen und es wurden alle Adressen der einzelnen Temperatursensoren abgefragt.

Danach wurde ein anderes Programm auf dem Mikrocontroller installiert, das die von den Temperatursensoren gemessenen Daten an die serielle Schnittstelle des Computers schickt. Das dazugehörige Programm findet sich in Abschnitt A.2 des Appendix. Hier werden alle 100 ms die Temperaturen an den einzelnen Sensoren abgefragt und in einen String geschrieben, der dann an die serielle Schnittstelle des Computers geschickt wird. Diese Prozedur dauert insgesamt circa eine Sekunde.

Um das Signal das von dem Mikrocontroller geschickt wurde, auswerten zu können wurde weiters in C♯ ein Programm geschrieben, das die Strings, die der Mikrocontroller schickt, in ein CSV-File schreibt. auch dieses Programm findet sich in Appendix A.3. Zusätzlich wurde auch eine GUI mitprogrammiert, ein Screenshot ist in Abbildung 33 zu sehen.

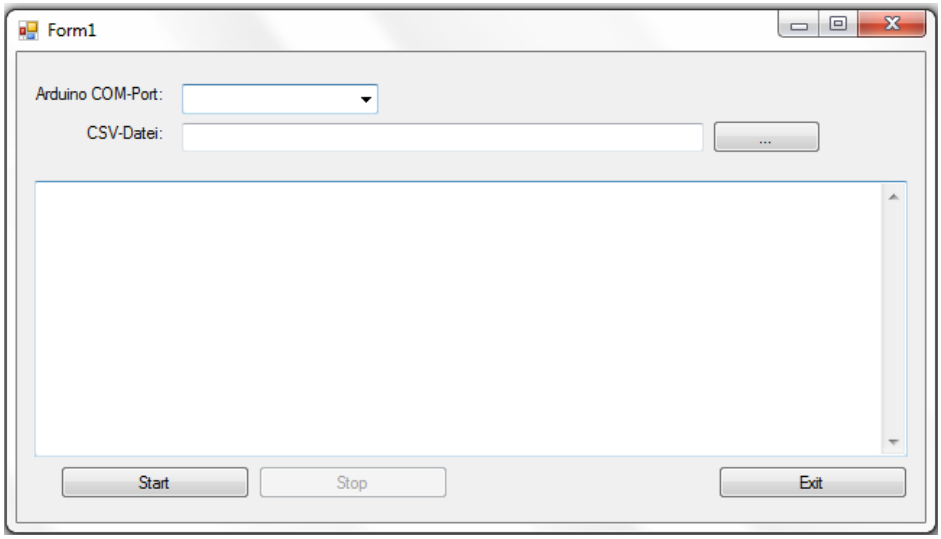



Abbildung 33: Graphical User Interface des C♯-Programmes

In der GUI kann der Port an dem der Mikrocontroller angeschlossen ist ausgewählt werden, außerdem kann eine CSV-Datei angegeben werden, in die die Daten geschrieben werden. Weiters werden die empfangenen Daten auch ausgegeben und es wurden ein Start und ein Stop Button eingefügt um die Datenübertragung zu starten oder zu stoppen.

# 4 Implementierung

Für die Implementierung der Modelle wurden zwei Simulatoren ausgewählt, diese sollen jetzt kurz vorgestellt werden, wobei auch auf die Verfahren und Solver, die eingesetzt werden um das entstehende Gleichungssystem zu lösen, eingegangen wird. Danach beschäftigt sich das Kapitel mit der Implementierung der Teilmodelle und ihren Besonderheiten in den beiden Simulatoren. Beide Simulatoren benutzen den Modelica-Standard als Grundlage für ihre Modelle.

## 4.1 Dymola

Dymola (Dynamic Modelling Laboratory) [10] ist aus geschichtlichen Gründen immer schon der Simulator, der den Modelica-Standard am besten versteht. Dies liegt daran, dass Modelica aus Dymola heraus entwickelt wurde. Die Version von Dymola, die in dieser Arbeit benutzt wird ist Dymola 2012, sie unterstützt den Modelica-Standard 3.2. In Abbildung 34 sieht man einen Screenshot der Modellierungsoberfläche in Dymola.

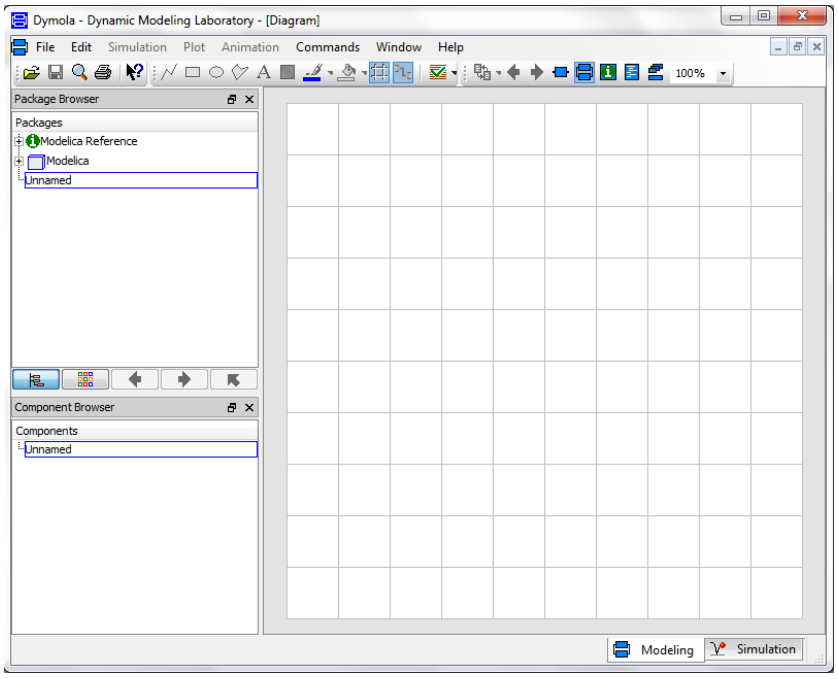



Abbildung 34: Modellierungsfenster in Dymola

Die Modellierungsumgebung ist in Layers unterteilt, zwischen denen hin und her geschaltet werden kann.

- *Icon-Layer*: Hier kann eine graphische Repräsentation für das Modell angefertigt werden.
- *Diagram-Layer*: Hier kann das Modell graphisch mit Komponenten aus der Modelica-Standard-Library und selbst erstellten Komponenten aufgebaut werden.
- *Information-Layer*: Hier kann eine Dokumentation zum Modell erstellt werden.
- *Modelica Text-Layer*: Das Modell kann auch textuell aufgebaut werden, das funktioniert genau so ,wie es in Kapitel 2.1.2 beschrieben ist.
- *Used Classes-Layer*: Hier werden alle Klassen, die in dem Modell benutzt werden, aufgeführt, insbesondere auch eventuelle `partial`-Klassen. In diesem Layer können also alle Gleichungen des Modells, die nicht aus `connections` entstehen, eingesehen werden.

Neben der Modellierungsumgebung gibt es noch die Simulationsumgebung, wie sie Abbildung 35 zeigt.

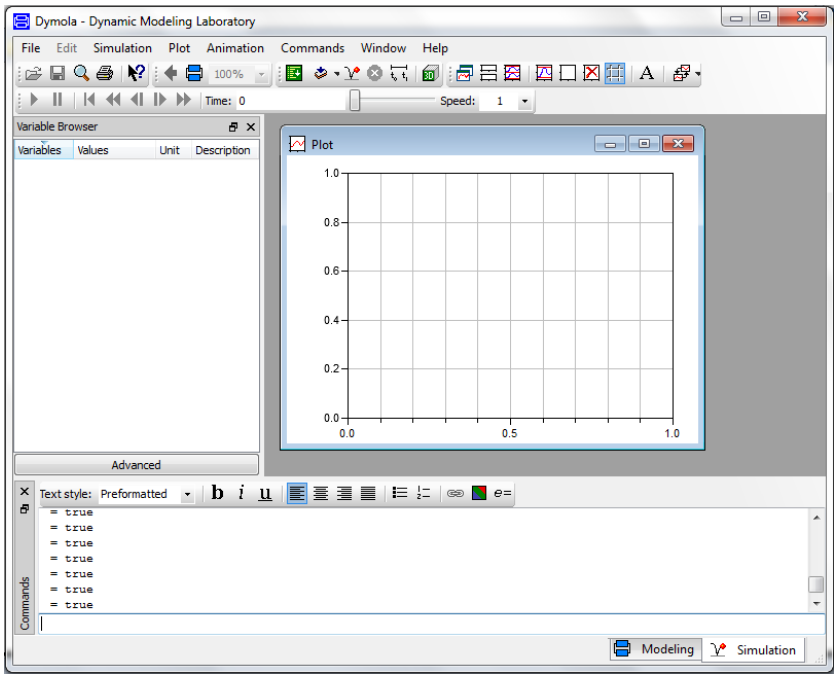



Abbildung 35: Simulationsfenster in Dymola

Hier werden im Variable-Browser nach der Simulation alle Variablen, die in dem Modell verwendet werden, angezeigt und können sofort geplottet werden.

### 4.1.1 Simulation in Dymola

Zur Simulation eines Modells müssen folgende Schritte durchgeführt werden:

- **Flattening:** Zuerst muss das DAE-System aufgestellt werden. Dazu wird die Hierarchie in dem Modell aufgebrochen, das heißt, dass alle Gleichungen aus Subkomponenten in die oberste Ebene gezogen werden, und die Gleichungen aus den `connections` werden generiert, sodass alle Gleichungen die das System beschreiben vorhanden sind.
- **Sortieren der Gleichungen:** Die Gleichungen müssen horizontal und vertikal sortiert werden.
  - *horizontal:* Für jede Variable wird eine Gleichung festgelegt, durch die sie berechnet wird.
  - *vertikal:* Die Gleichungen müssen so sortiert sein, dass jede Variable aus dieser Gleichung berechnet werden kann, dass also alle anderen vorkommenden Variablen bereits in früheren Gleichungen berechnet wurden.
- **Eliminieren trivialer Gleichungen:** Beim Physical Modelling treten sehr häufig triviale Gleichungen der Form $x = y$ auf, diese müssen eliminiert werden. Im Allgemeinen reduziert sich dadurch die Größe des betrachteten Gleichungssystems drastisch.
- **Erkennen & Aufbrechen algebraischer Schleifen:** Eventuell vorhandene algebraische Schleifen müssen erkannt und aufgebrochen werden, sodass möglichst wenige algebraische Zwangsbedingungen übrig bleiben.
- **Index-Reduktion:** Bleiben nach dem Aufbrechen der algebraischen Schleifen noch Zwangsbedingungen übrig und ist die Jacobi-Matrix des Systems singulär, müssen Verfahren zur Index-Reduktion eingesetzt werden, bis das System höchstens Index 1 hat.
- **Lösen des Gleichungssystems:** Das übrig geblieben Gleichungssystem kann nun mit Hilfe eines ODE-Solvers gelöst werden.

Diese Schritte muss prinzipiell jeder Simulator, der auf den Modelica-Standard oder einen ähnlichen Standard aufbaut, beherrschen.

**Tarjan-Algortihmus:** In Dymola wird zum erkennen algebraischer Schleifen der *Tarjan-Algortihmus*, der aus der Graphentheorie stammt, benutzt. Dafür wird zuerst ein Graph aufgebaut, der jeder Variable die Gleichungen zuordnet in der sie vorkommt. Der Algorithmus sucht dann Schritt für Schritt einen Weg durch den Graphen ohne im Kreis zu laufen, hat er irgendwann keine Möglichkeit mehr weiterzukommen ohne jede Variable eindeutig einer Gleichung zugeordnet zu haben, bricht er ab und eine algebraische Schleife ist gefunden. Der Vorteil dieser Methode ist, dass durch den Weg, den der Algorithmus wählt, gleichzeitig auch eine horizontale und vertikale Sortierung der Gleichungen erfolgt. Ist der Algorithmus erfolgreich, hat die zum Graphen gehörige Adjazenzmatrix die Gestalt einer unteren Dreiecksmatrix. Unabhängig vom Ergebnis liefert der Algorithmus immer eine untere Blockdreiecksmatrix als Adjazenzmatrix, in der die algebraischen Schleifen leicht identifiziert werden können.

**Tearing-Algorithmus:** Zum Aufbrachen der algebraischen Schleifen wird ein sogenannter *Tearing-Algorithmus* verwendet, eine genaue Beschreibung zum Vorgehen dieses Algorithmus findet sich in [11]. Grundsätzlich gilt, dass die Struktur der algebraischen Schleife erkannt werden muss. Ist das algebraische System linear, kann es mit Verfahren zur Lösung linearer Gleichungssysteme gelöst werden. Bei nichtlinearen algebraischen Schleifen ist in den meisten Fällen die Newton-Iteration die beste Wahl.

**Index-Reduktion:** Zur Index-Reduktion wird der *Pantelides-Algorithmus* verwendet um den Index des DAEs zu bestimmen und um festzulegen welche algebraische Gleichung wie oft symbolisch differenziert werden muss um den Index des DAEs auf 0 oder 1 zu bringen. Da beim Ableiten neue Gleichungen entstehen, wird das Gleichungssystem überbestimmt. Um dem entgegenzuwirken werden sogenannte *Dummy-Variablen* eingeführt, dazu wird eine abgeleitete Variable $\frac{\mathrm{d}x}{\mathrm{d}t}$ ausgesucht und durch die Variable $dx$ ersetzt. Der Zusammenhang zwischen $x$ und seiner Ableitung wird so aufgehoben. Durch die neuen Variablen wird die Überbestimmtheit des Gleichungssystems ebenfalls aufgehoben.
Bei der Auswahl, welche Ableitung durch eine Variable ersetzt werden soll, ist die Wahl der richtigen Variable sehr wichtig, da es sonst dazu führen kann, dass neue Zwangsbedingungen entstehen oder die alten nicht aufgelöst werden. Dafür gibt es leider keine Strategie, die in jedem Fall zum Erfolg führt, das Verfahren ist deshalb zu einem gewissen Grad heuristisch.

**DASSL-Solver** Der *DASSL-Solver* [12] ist ein numerischer Solver, der im Gegensatz zu den meisten Standard-Solvern darauf ausgelegt ist, Differentialgleichungen in impliziter Form zu lösen, also Gleichungen der Form

$$F(t, y, y') = 0 \tag{72}$$
$$y(0) = y_0 \tag{73}$$
$$y'(0) = y'_0 \tag{74}$$

In jedem Zeitschritt wird die Ableitung durch einen Differenzenquotienten ersetzt, im einfachsten Fall durch den Rückwärtsdifferenzenquotienten. Die entstehende Gleichung

$$F\left(t_n, y_n, \frac{y_n - y_{n-1}}{h}\right) = 0 \tag{75}$$

mit Schrittweite $h$, wird durch Newton-Iteration gelöst. Dieses Vorgehen ist das gleiche wie bei dem BDF (Backwards Differential Formula) erster Ordnung. In DASSL sind die BDF-Verfahren der Ordnung 1 bis 5 implementiert. So kann in jedem Zeitschritt neben der Schrittweite auch die Ordnung an das momentane Verhalten der Lösung angepasst werden. Das verwendete Newton-verfahren ist eine leicht modifizierte Version des klassischen Newton-Verfahrens, in der die Jacobimatrix nicht in jedem Schritt neu berechnet werden muss, sondern für einige Schritte behalten wird und erst dann verworfen wird, wenn der geschätzte Fehler zu groß wird.

Dadurch, dass nur das implizite System betrachtet wird, ist der DASSL-Algorithmus auch geeignet um Probleme mit Index 1 zu lösen, wegen der Newton-Iteration können aber keine Systeme mit höherem Index gelöst werden, da die Jacobi-Matrix sonst singulär wäre.

Ein weiteres Problem bei der Lösung von DAEs ist, dass für die bei der Index-Reduktion eingeführten neuen Variablen keine Anfangsbedingungen gegeben sind. In DASSL ist dazu eine Routine implementiert, die bei der Initialisierung des Systems eine Newton-Iteration mit den bekannten Werten von $y_0$ und $y'_0$ startet und die unbekannten Elemente der beiden Vektoren zu berechnen.

## 4.2 MapleSim

MapleSim [13] ist eine Entwicklung von MapleSoft, das zur Simulation von physikalischen Systemen eingesetzt wird. In dieser Arbeit wird MapleSim 4.5 verwendet. MapleSim ist ebenfalls auf Modelica aufgebaut, verwendet aber nur den Modelica-Standard 3.1. Eine Neuerung, die im Modelica-Standard 3.2 eingeführt wurde und für diese Arbeit relevant ist, ist, dass bei allen Komponenten, die dissipative Elemente beschreiben, ein optionaler Heat-Port hinzugefügt wurde, um die Anbindung an thermische Systeme zu erleichtern. Da dies bei MapleSim noch nicht implementiert wurde, müssen diese Komponenten, die im Maschinenmodell verwendet werden, für die Kopplung neu geschrieben werden.

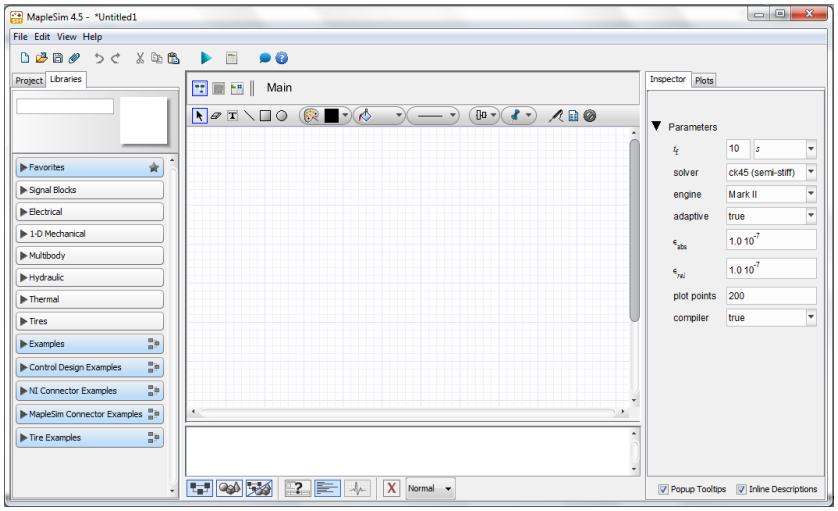



Abbildung 36: Graphische Oberfläche von Maplesim

Weitere Unterschiede, die die Modellbildung betreffen, sind

- Es kann nicht auf die textuelle Beschreibung der Komponenten zugegriffen werden, das bedeutet, dass die Modelle auf jeden Fall rein graphisch aufgebaut werden müssen.
- Um eine Variable als Simulationsergebnis zu erhalten müssen sogenannte *Probes* gesetzt werden, das heißt es werden nicht wie in Dymola standardmäßig alle Systemvariablen bei Simulationsende zurückgeliefert.
- Zur Erstellung neuer Komponenten in MapleSim muss nicht unbedingt Modelica-Code benutzt werden. Es wird ein Maple-Template zur Verfügung gestellt, in dem das System aus Variablen, Parametern, Gleichungen und Anfangswerten in ein Gleichungssystem umgewandelt wird, das

dann gemeinsam mit der Definition der Ports, die graphisch erfolgt, zur Definition der Komponente benutzt wird. Hier kann dann in weiterer Folge auch der Modelica-Code der generierten Komponente eingesehen und verändert werden.

### 4.2.1 Simulation in MapleSim

Für die Simulation eines Modells gilt auch in MapleSim das gleiche wie in Dymola (siehe 4.1.1), die ersten Schritte die dann zu dem Gleichungssystem, das von dem Solver gelöst werden kann, werden alle mit der sogenannten *MarkII-Engine* durchgeführt. Wie die einzelnen Schritte in diesem Prozess aussehen ist leider unklar, es wird aber auf jeden Fall auch eine Form der Indexreduktion eingesetzt um das System numerisch lösen zu können. Es ist zu erwarten, dass MapleSim mit Maple im Hintergrund ausgereifte Taktiken zur symbolischen Umformung der Gleichungen nutzt.

Es werden drei Solver zur numerischen Berechnung der Löser zur Verfügung gestellt:

- *RKF45* für nicht steife Systeme
- *CK45* für fast steife Systeme
- *Rosenbrock* für steife Systeme

Da für die hier betrachteten Modelle der Rosenbrock-Solver die beste Wahl ist, wird dieser etwas genauer betrachtet. Der implementierte Rosenbrock-Solver hat die Ordnung 3/4.

Prinzipiell sind die Rosenbrock-Methoden von den impliziten Runge-Kutta-Verfahren abgleitet, wobei der Newtonschritt zur Berechnung der internen Ableitung explizit eingesetzt wird (siehe [14]). Die Rosenbrock-Verfahren bilden eine Familie von impliziten Lösern, die sehr gut geeignet sind um sehr große steife Systeme zu lösen. Auch Index-1-DAEs können mit ihnen gelöst werden. Diese Eigenschaften machen sie zu einer sehr guten Wahl zur Lösung von physikalischen Modellen.

## 4.3 Modelle

In diesem Abschnitt werden die Modelle und ihre Modellierung in den beiden Simulatoren vogestellt, es wird auch auf etwaige Unterschiede in der Implementierung eingegangen. Zuerst werden die Implementierungen des Maschi-

nenmodells vorgestellt, danach die des Raummodells in der Version, die zur Validierung verwendet wird, und zum Abschluss noch die Implementierung des gekoppelten Modells.

### 4.3.1 Maschinenmodell

Das Maschinenmodell wird in Dymola und MapleSim prinzipiell gleich aufgebaut, einziger Unterschied ist, dass in Modelica 3.2 das Modell des permanenterregten Synchronmotors abgeändert wurde, da es die Möglichkeit gibt, zusätzliche Heat-Ports zu implementieren. Abbildung 37 zeigt das Motormodell in MapleSim und Abbildung 38 das Modell in Dymola zum Vergleich.

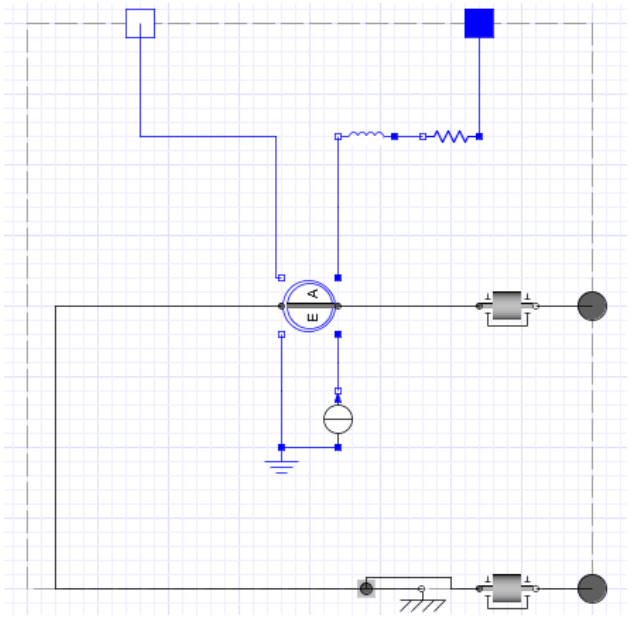


Abbildung 37: Modell des permanenterregten Synchronmotors in MapleSim

Für die Erstellung des einfachen Maschinenmodells hat die Änderung des Aufbaus der Komponente keine Auswirkungen, denn wie man sieht, sind nur Komponenten hinzugefügt worden, die einen Heat-Port haben und deshalb können diese Komponenten in diesem Modell außer Acht gelassen werden. Die notwendigen Änderungen für den Aufbau des gekoppelten Gesamtmodells werden später erklärt.

Der Ankerkreis des Motors wird durch eine Reihenschaltung eines Widerstandes und einer Spule modelliert, der Permanentmagnet durch eine konstante Stromquelle. In der Komponente AirGapDC wird das Drehmoment das der

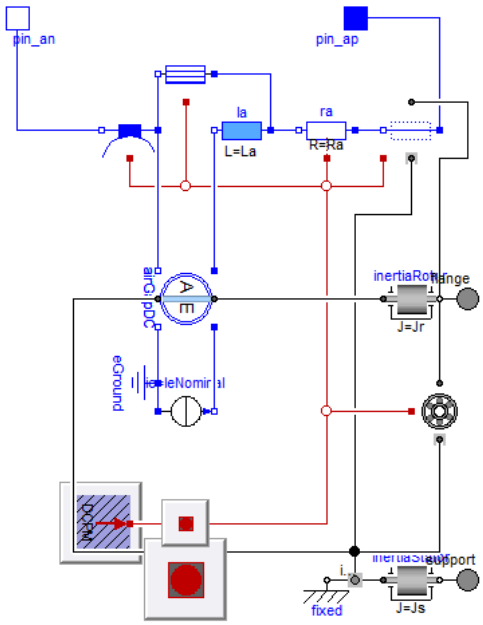



Abbildung 38: Modell des permanenterregten Synchronmotors in Dymola

Motor erzeugt berechnet, eine genauere Beschreibung der Komponente findet sich in [15].

Zur Erstellung des Modells zeigt Tabelle 9 noch einmal die gesammelten Parameter aus Kapitel 3.1.

| | | |
|---|---|---|
| $J_{mot}$ | $6,18 \cdot 10^{-4}$ | $\mathrm{kg} \cdot \mathrm{m}^2$ |
| $I_N$ | $2,9$ | A |
| $n_N$ | 3000 | 1/min |
| $R_{str}$ | $3,05$ | $\Omega$ |
| $L_D$ | 16 | mH |
| $i$ | 2 | |
| $c_R$ | $2 \cdot 10^7$ | $\frac{\mathrm{N}}{\mathrm{m}}$ |
| $h$ | 30 | mm |
| $J_G$ | $7,01 \cdot 10^{-3}$ | $\mathrm{kg} \cdot \mathrm{m}^2$ |
| $m_{ges}$ | $555,5$ | kg |

Tabelle 9: Parameterliste des Maschinenmodells

Der Aufbau des Modells in Dymola und MapleSim wird in Abbildung 39 gezeigt.

Der scheinbare Unterschied zwischen den Modellen bei der Massenkompo-

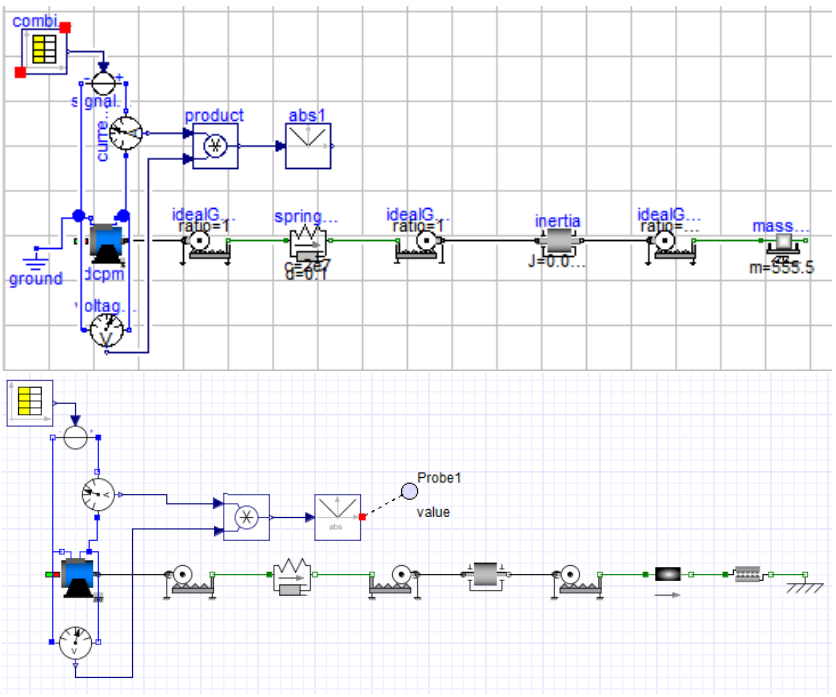



Abbildung 39: Maschinenmodell in Dymola (oben) und MapleSim (unten)

nente kommt daher, dass es in Dymola ein vorgefertigtes Modell einer reibungsbehafteten Masse gibt, das in MapleSim noch nicht implementiert ist. Am Modell selbst ändert sich dadurch nichts.

Die Federeigenschaften des Zahnriemens wurden über ein translatorisches Federdämpfer-Modell implementiert, der Dämpfer dient nur dazu um ein Aufschwingen der Feder zu verhindern und die Dämpferkonstante wird sehr klein gewählt. Für das Übersetzungsverhältnis $i$ ist das zweite ideale Getriebe, das von translatorischer auf rotatorische Bewegung übersetzt, verantwortlich, da hier für die Verhältniszahl genau das umgekehrte Verhältnis im Vergleich zur Konvention herangezogen wird, wird der Parameter auf $i^{-1}$ gesetzt. Ähnliches gilt für die Übersetzung von der Gewindestange auf den Verschubwagen, hier wird als Übersetzungsverhältnis $\frac{2\pi}{h}$ gewählt. Die restlichen Parameter konnten entsprechend ihren Werten in der Tabelle eingesetzt werden.

Der Eingang der Spannungsquelle wurde aus Messungen, die im Zuge der bereits erwähnten Bachelorarbeit [5] gemacht wurden, berechnet. Bei den Messungen wurde der Strom und die vom Motor aufgenommene Leistung gemessen. Da die Maschine an das 3-phasige Starkstromnetz angeschlossen ist, wurde an jeder der drei Phasen Strom, Leistung und der Phasenwinkel gemessen. Aus diesen Messdaten wurde die Spannung in jeder der drei Phasen

errechnet und dann zusammenaddiert um einen Eingang für die Spannungsquelle zu erhalten. Abbildung 40 zeigt, die erhaltenen Daten.

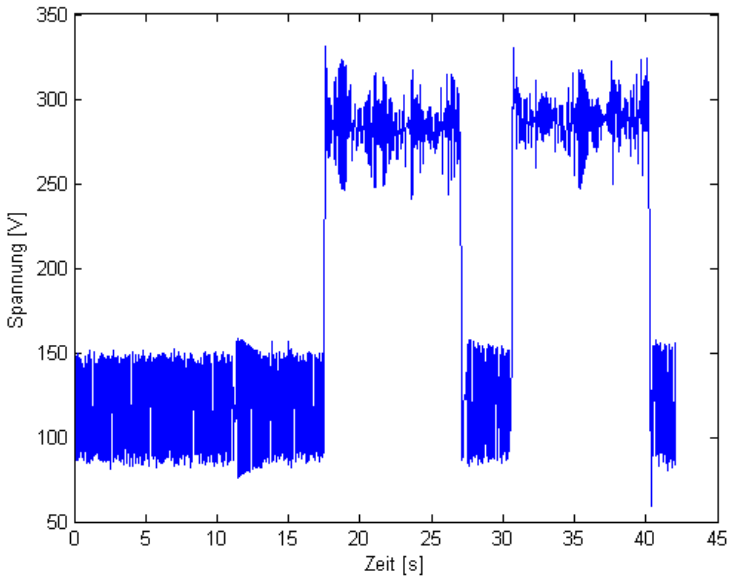



Abbildung 40: Spannungsdaten aus den Messungen

Der Graph zeigt den Spannungsverlauf über eine Verschubperiode hinweg, wobei die Masse einmal hin und dann wieder zurück bewegt wurde. Wie zu erkennen ist, wird dieses Verhalten durch den Graphen nicht ersichtlich, was im Modell dazu führen würde, dass die Masse immer nur in eine Richtung bewegt wird. Außerdem ist noch zu sehen, dass auch in der Stillstandsphase Strom verbraucht wird, das liegt daran, dass die Messung vor der Leistungselektronik vorgenommen wurde, die natürlich auch Strom verbraucht.

Deshalb wurden die Daten leicht modifiziert, zuerst wurde der Mittelwert aus allen Werten genommen, bei denen nicht verschoben wurde und dann wurden die Werte der zweiten Verschubphase mit $-1$ multipliziert, um einen negativen Spannungsverlauf zu erhalten. Das Resultat ist in Abbildung 41 zu sehen.

Um die Daten in die TimeTable-Komponenten der beiden Simulatoren einzulesen, wurden die Daten wieder abgespeichert. Für Dymola müssen Daten in einer .mat-Datei hinterlegt sein, während in MapleSim eine .csv-Datei oder eine .xls-Datei eingelesen werden kann.

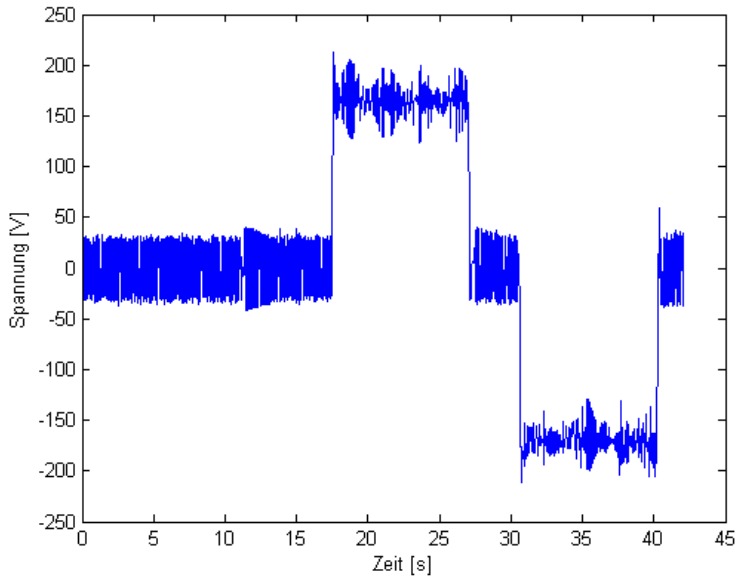



Abbildung 41: Modifizierte Spannungsdaten aus den Messungen, die als Eingang für das Maschinenmodell dienen

### 4.3.2 Raummodell

Wie bereits erwähnt wird hier das Kompartmentmodell des Raumes, das zur Validierung des Modells verwendet wird, vorgestellt. Die Styroporbox hat die Abmessungen $53,5\text{cm} \times 34\text{cm} \times 24\text{cm}$, diese wird in acht Kompartments aufgeteilt. Das ergibt für die Parameter

$$x = 0,2675\text{m} \tag{76}$$
$$y = 0,17\text{m} \tag{77}$$
$$z = 0,12\text{m} \tag{78}$$

Die Parameter der Luft, die noch festzulegen sind, kann zum Beispiel auf [16] zurückgegriffen werden

$$\rho = 1,204 \frac{\text{kg}}{\text{m}^3} \tag{79}$$

$$c_p = 1012 \frac{\text{J}}{\text{kg} \cdot \text{K}} \tag{80}$$

$$k = 0,0257 \frac{\text{W}}{\text{m} \cdot \text{K}} \tag{81}$$

In Abbildung 42 wird das Raummodell in MapleSim gezeigt, das Raummodell in Dymola schaut genau so aus, es gibt hier keine Unterschiede in der Modellierung.

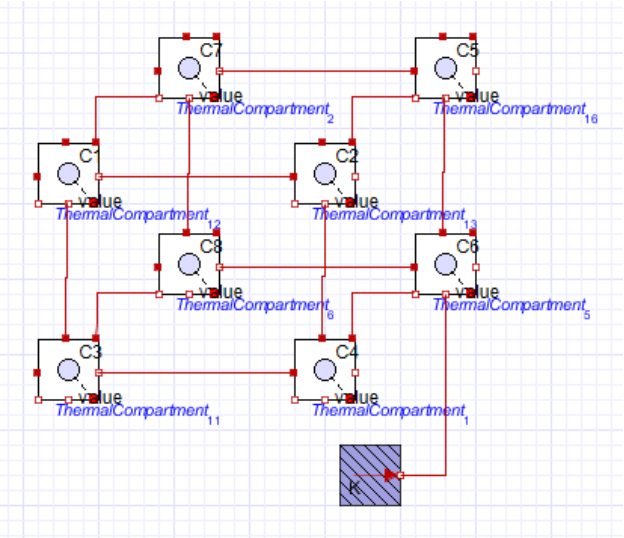



Abbildung 42: Aufbau des Raummodells in MapleSim

Die Temperaturquelle, die den Lötkolben darstellen soll, ist am Boden des rechten hinteren Kompartments angebracht und hat eine Temperatur von 250°C.

### 4.3.3 Gesamtmodell

Für das Gesamtmodell bleiben die Parameter des Maschinenmodells unverändert, der Raum wird mit den Maßen 20m × 10m × 6m angenommen um die Verhältnisse einer Maschinenhalle widerzuspiegeln, er wird in 16 Kompartments unterteilt zu je 5m × 5m × 3m. Die Parameter der Luft bleiben ebenfalls gleich. Es wird angenommen, dass die Maschine in den beiden vorderen, mittleren Kompartments steht, wobei das rechte Kompartment von der

Wärmeabgabe der mechanischen Komponenten, also dem Reibungsverlust in der SlidingMass-Komponente und das linke Kompartment von den Verlusten in den elektrischen Komponenten, also dem Motor erhitzt wird. Um die Vergleichbarkeit der Modelle in den beiden Simulatoren zu gewährleisten, wird nur die Verlustleistung in der Wicklung, also in der Widerstands-Komponente in Betracht gezogen.

Da für die Kopplung der Modelle in den beiden Simulatoren einige kleine Änderungen vorzunehmen sind, werden sie im Folgenden einzeln betrachtet.

**Dymola:** In Dymola muss dadurch, dass die dissipativen Elemente bereits Heat-Ports besitzen sehr wenig geändert werden, Abbildung 43 zeigt den Aufbau des gekoppelten Modells.

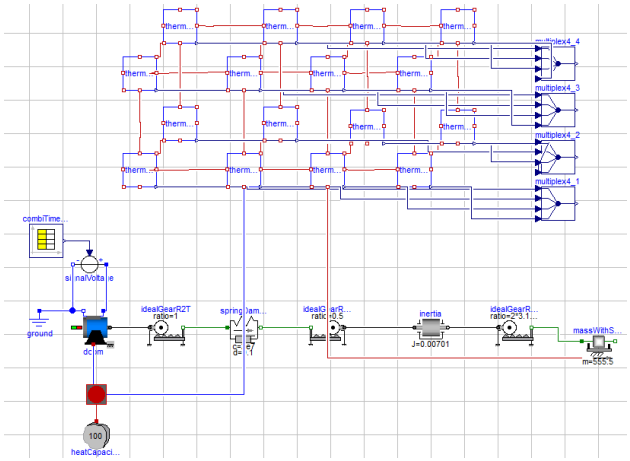



Abbildung 43: Aufbau des gekoppelten Gesamtmodells in Dymola

Die einzelnen Verlustleistungen in den dissipativen Elementen des Motors werden im Modell des Motors nicht addiert, sondern als Vektor behandelt, deshalb muss eine ThermalPortDCPM-Komponente verwendet werden um die einzelnen Wärmeflüsse wieder verwenden zu können. Dabei wird nur der Wärmefluss, der vom elektrischen Widerstand erzeugt wird in das Raummodell eingeführt, während die anderen Wärmeflüsse in eine andere Wärmekapazitäts-Komponente führen. Da die Parameter die für die Berechnung der Verluste in den anderen dissipativen Komponenten des Motors alle zu 0 gewählt wurden, findet in diesem Kompartment keine Erwärmung statt. Die Verbindungen dienen nur dazu das Gleichungssystem konsistent zu halten.

**MapleSim:** Da in MapleSim die optionalen Heat-Ports nur in der elektrischen Library bis jetzt eingeführt wurden, muss die abfallende Leistung an der Reibungs-Komponente durch messen herausgelesen werden. Sie wird dann mit Hilfe einer Prescribed HeatFlow Komponente in das Kompartmentmodell übergeben. Weiters muss das Modell des Motors nachgebaut werden, da es keine Option gibt, den Heat-Port des Resistors, der in dem Modell enthalten ist einzuschalten. An dem Modell selbst und den Parametern ändert sich dadurch aber nichts. In Abbildung 44 ist der nachgebaute Motor zu sehen und in Abbildung 45 das gekoppelte Gesamtmodell.

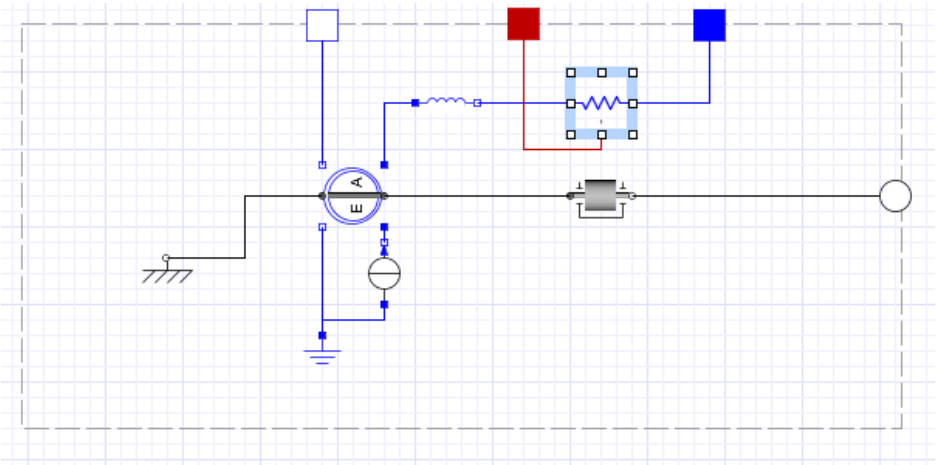


Abbildung 44: Modell des permanenterregten Synchronmotors mit aktiviertem Heat-Port in MapleSim

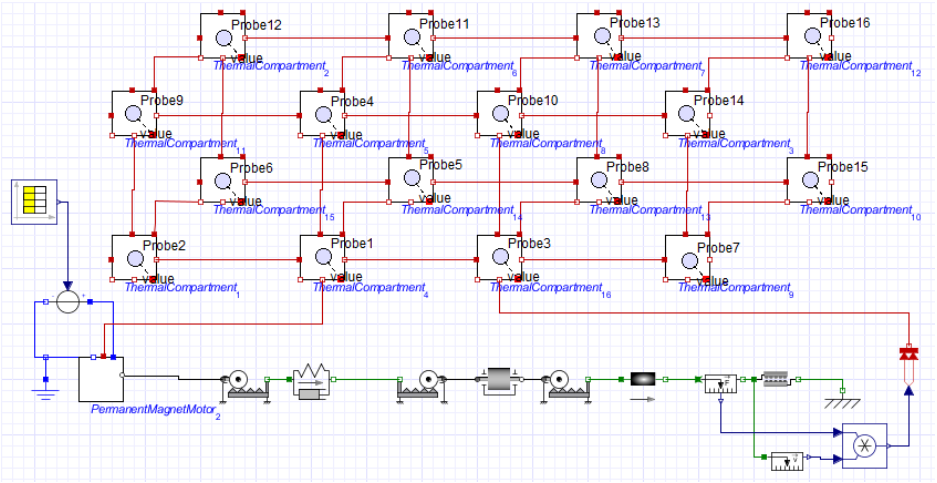



Abbildung 45: Aufbau des gekoppelten Gesamtmodells in MapleSim

# 5 Ergebnisse

In diesem Kapitel werden die Simulationsergebnisse zu den in den vorherigen Kapiteln vorgestellten Modellen präsentiert. Zu Beginn wird überprüft, ob die beiden Motormodelle in Dymola und MapleSim überhaupt vergleichbar sind. Danach werden die Ergebnisse der beiden betrachteten Teilmodelle und des gekoppelten Gesamtmodells vorgestellt.

## 5.1 Motorenvergleich

Zum Vergleich der beiden Motormodelle wird ein sehr einfaches Modell erstellt. Der Motor wird für 5 Sekunden mit konstanter Spannung betrieben, danach wird die gleiche Spannung negativ angelegt. Dabei muss der Motor eine Last bewegen, die mit einem Trägheitsmoment von 10 kg·m$^2$ entgegenwirkt. Die beiden Motormodelle sind natürlich gleich parametrisiert.

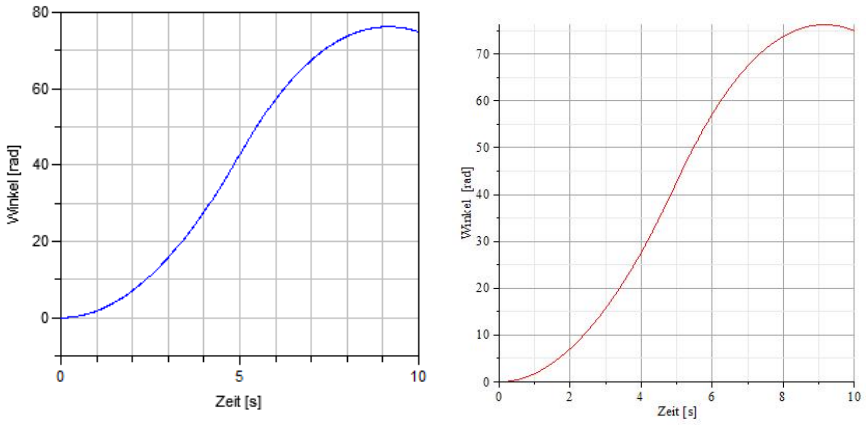



Abbildung 46: Ergebnisse des Motorvergleichsmodells in Dymola (links) und MapleSim (rechts)

Wie man sieht leisten die beiden Motormodelle das gleiche, das heißt trotz des unterschiedlichen Aufbaus leisten die beiden Modelle dasselbe.

## 5.2 Maschinenmodell

Das Maschinenmodell wird gegenüber Messdaten validiert, dazu wurden die Simulationsergebnisse mit Hilfe von Matlab ausgewertet und mit den Mes-

sergebnissen in einer Grafik geplottet, zusätzlich wurde noch die mittlere Leistungsaufnahme eingezeichnet.

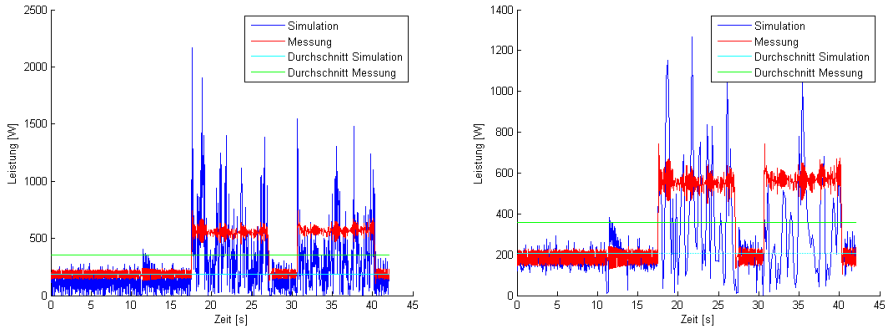



Abbildung 47: Simulationsergebnisse und Vergleich zu Messdaten der Maschinenmodelle in Dymola (links) und MapleSim (rechts)

Wie deutlich zu erkennen ist, pflanzen sich die Störungen aus den Eingangsdaten durch das Modell fort und verstärken sich deutlich, dennoch bleiben die Mittelwerte vergleichbar, wenn man in Betracht zieht, dass bei der Messung auch der Energieverbrauch der Leistungselektronik mitgemessen wurde. Beim Vergleich der beiden Simulatoren sieht man in diesem Modell doch deutliche Unterschiede zwischen den Simulationsergebnissen. Dies liegt höchstwahrscheinlich an dem sehr gestörten Eingangssignal, wodurch jeder Unterschied in der Schrittweite oder des verwendeten Algorithmus sofort sichtbare Konsequenzen hat. Eine Lösung dafür wäre sicherlich eine Glättung des Eingangssignals.

## 5.3 Raummodell

Für die Validierung des Raummodells wurde am Versuchsaufbau die Temperaturausbreitung über den Zeitraum von 54 min gemessen. Bei den Daten wurden Fehlmessungen der Temperatursensoren herausgefiltert.

Im Vergleich dazu werden die beiden Simulationsergebnisse gezeigt.

Man sieht, dass die beiden Simulationsergebnisse übereinstimmen. Im Vergleich zur Messung zeigt sich, dass die Modellannahmen die getroffen wurden wohl zu restriktiv sind. Zum einen ist es nicht gelungen die Styroporbox komplett zu isolieren, da für die Temperatursensoren und das Kabel des Lötkolbens Löcher in die Wand gebohrt werden mussten, zum anderen ist die Annahme, dass der Wärmetransport nur über Konduktion zwischen

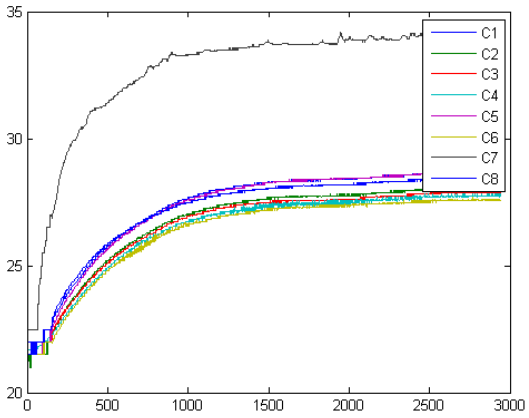



Abbildung 48: Messergebnisse am Versuchsaufbau

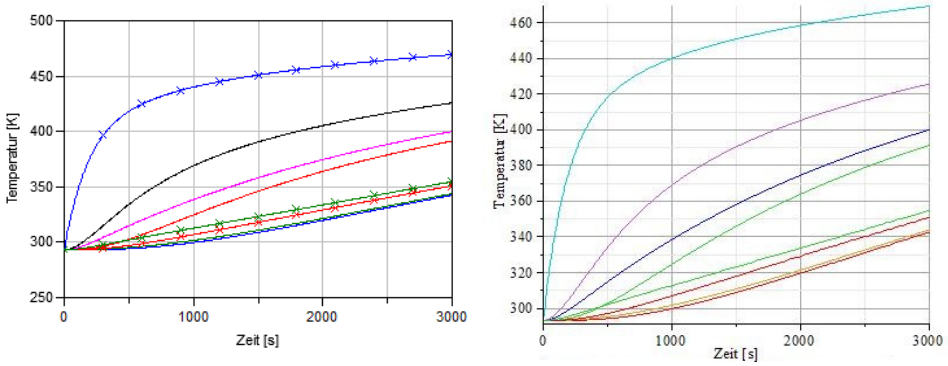



Abbildung 49: Simulationsergebnisse des Raummodells in Dymola (links) und MapleSim (rechts)

zwei angrenzenden Kompartments erfolgt, offensichtlich falsch, da sich so die gleichmäßige Erwärmung der Kompartments bei der Messung nicht erklären ließe.

## 5.4 Gesamtmodell

Zum Abschluss werden noch die Ergebnisse der gekoppelten Gesamtsimulation präsentiert, um die Rechenzeit nicht unnötig lang zu machen wurde die Temperaturausbreitung über 6 Stunden simuliert. Da der Raum in insgesamt 16 Kompartments aufgeteilt wurde, wäre die Darstellung in einer Grafik zu unübersichtlich, deshalb wurden die Kompartments dahingehend aufgeteilt,

wo ein ähnlicher Temperaturverlauf zu erwarten ist. In der Grafik links oben sieht man den Temperaturverlauf des Kompartments, in dem der elektrische Teil der Maschine, also der Elektromotor, steht. Die Grafik rechts daneben zeigt die daran direkt angrenzenden Kompartments, links unten folgen die Kompartments, für die die Wärme durch genau ein anderes Kompartment fließen muss, und rechts daneben werden die restlichen Kompartments abgebildet, für die die Wärme am längsten braucht um sich dorthin auszubreiten. Diese Aufteilung gilt für beide Simulationsergebnisse.

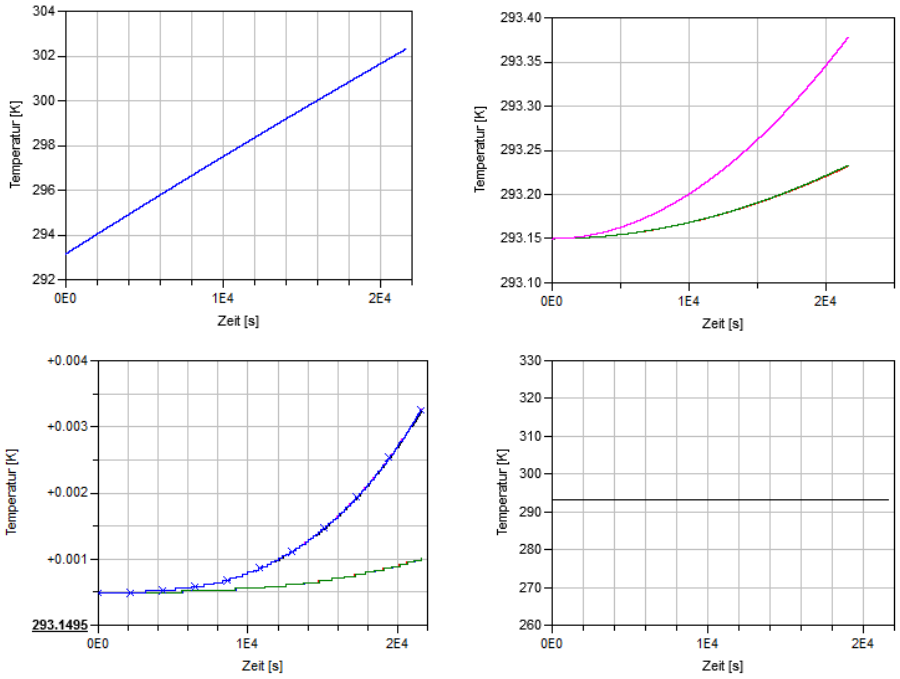



Abbildung 50: Simulationsergebnisse des Gesamtmodells in Dymola

Interessant an diesen beiden Simulationsergebnissen ist, dass die beiden Simulatoren zwei vollkommen unterschiedliche Temperaturverläufe berechnen. Während bei Dymola im Kompartment, in dem die Maschine steht, die Temperatur annähernd linear steigt, steigt die Temperatur bei MapleSim in diesem Kompartment exponentiell. Das Ergebnis von Dymola sieht realistischer aus, da der Wärmefluss, der vom Elektromotor über die Periodendauer konstant bleibt und nicht mehr wird, wie es das Ergebnis von MapleSim vermuten lässt. Ein weiterer interessanter Punkt ist, dass die Abwärme die durch mechanische Reibung entsteht, in diesem Modell keine Auswirkungen hat, vergleicht man die Spitzen der abgegebenen Wärme der zwei betrachteten dissipativen Komponenten so, ist der Wärmefluss beim Elektromotor circa 25 mal höher als der der Reibungskomponente.

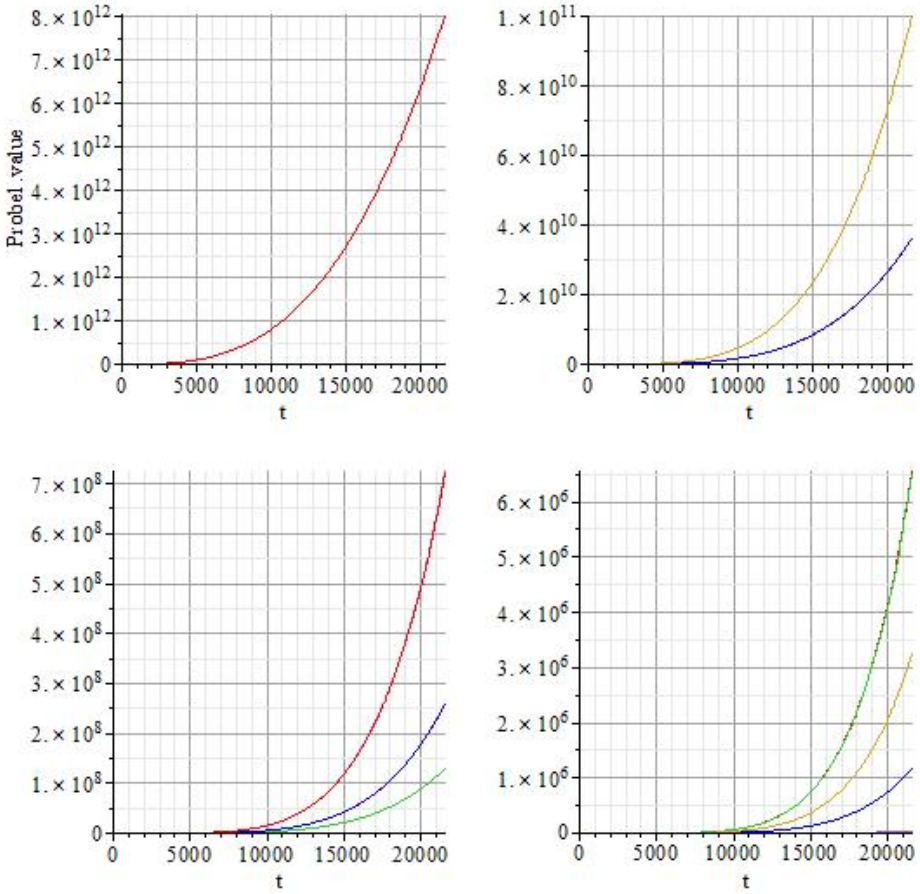



Abbildung 51: Simulationsergebnisse des Gesamtmodells in MapleSim

Ein weiterer Punkt der noch anzusprechen ist, sind die Rechenzeiten. Auf dem Rechner auf dem diese Arbeit geschrieben wurde, brauchte das MapleSim Modell circa 2 Stunden für die Berechnung, Dymola kam mit circa einer Stunde Rechenzeit aus.

# 6 Fazit

In dieser Arbeit wurde gezeigt, dass es prinzipiell möglich ist verschiedene Modelle im Bereich der Thermodynamik miteinander in einem Simulator zu koppeln. Als Ergebnis für das Projekt Info, im Zuge dessen diese Arbeit geschrieben wurde, bleibt zu sagen, dass sich diese Art der Kopplung nicht wirklich für die Gesamtsimulation einer Maschinenhalle eignet, da die entstehenden Rechenzeiten auch schon bei kleinen Modellen, die sehr restriktive Modellannahmen treffen und deshalb die Wirklichkeit nicht wirklich widerspiegeln, sehr groß werden. Für sehr große Modelle im Bereich der Thermodynamik gibt es spezielle Simulatoren, sodass eine Simulation mit Hilfe von Co-Simulationstools sicher die beste Wahl ist.

Zum Vergleich der beiden Simulatoren, Dymola und MapleSim, ist zu sagen, dass sich bei der Simulation der einfachen Modelle des Motorvergleichs und des Raummodells gezeigt hat, dass bei genügend glatten Eingangsdaten und einfachen Modellen, die Simulatoren gleiche Ergebnisse liefern. Beim Maschinenmodell, bei dem das gestörte Eingangssignal eine große Herausforderung für die beiden Simulatoren darstellte, konnten beide Simulatoren keine realistischen Ergebnisse liefern, was es unmöglich macht zu entscheiden, welcher Simulator besser war. Beim Gesamtmodell liefert Dymola die besseren Ergebnisse, da sie realistischer sind.
Bei der Modellbildung hat Dymola den großen Vorteil, dass es bereits Modelica 3.2 implementiert hat, in dem eine größere Palette an Komponenten angeboten wird. Außerdem ist die Kombination aus graphischer und textueller Modellierung sehr vorteilhaft, wenn Modelle debugged werden müssen. Alles in allem ist Dymola momentan noch der bessere Simulator, MapleSim hat aber sicher die Möglichkeit, diesen Rückstand aufzuholen, vor allem weil mit Maple ein leistungsstarkes Programm für die symbolischen Umformungen der Gleichungen zur Verfügung steht.

# 7 Ausblick

Als weitere Arbeiten wären Verfeinerungen der Modellteile auf jeden Fall interessant.

Beim Maschinenmodell wäre ein erster Schritt die Glättung der Daten um zu sehen, ob und wie die glatteren Daten das Simulationsergebnis beeinflussen. Eine weitere Möglichkeit wäre, den Regler in das Modell mit einzubeziehen.

Für das Raummodell wäre eine Verfeinerung der Diskretisierung möglich, auch hier könnten die Auswirkungen auf die Simulationsergebnisse untersucht werden. Weiters könnte der Versuchsaufbau weiter idealisiert werden, denkbar wäre, statt Luft als Medium ein wärmeleitendes Gel zu verwenden um die Effekte von Konvektion und Wärmestrahlung so weit wie möglich zu unterdrücken.

Für das Gesamtmodell ist eine genauere Untersuchung, warum die Simulatoren so unterschiedliche Ergebnisse liefern, auf jeden Fall notwendig. Da ein weiterer Anstieg der Rechenzeit zu erwarten ist, wenn verfeinerte Teilmodelle verwendet werden, ist eine erneute Kopplung der Modellteile in einem Gesamtmodell wahrscheinlich nicht erstrebenswert.

# A Programmcodes des Mikrocontrollers

## A.1 AddressErmittlung

```
#include <OneWire.h>

OneWire  ds(8);  // Connect your 1-wire device to pin 8

void setup(void) {
  Serial.begin(9600);
  discoverOneWireDevices();
}

void discoverOneWireDevices(void) {
  byte i;
  byte present = 0;
  byte data[12];
  byte addr[8];

  Serial.print("Looking for 1-Wire devices...\n\r");
  while(ds.search(addr)) {
    Serial.print("\n\rFound \'1-Wire\'
                  device with address:\n\r");
    for( i = 0; i < 8; i++) {
      Serial.print("0x");
      if (addr[i] < 16) {
        Serial.print('0');
      }
      Serial.print(addr[i], HEX);
      if (i < 7) {
        Serial.print(", ");
      }
    }
    if ( OneWire::crc8( addr, 7) != addr[7]) {
        Serial.print("CRC is not valid!\n");
        return;
    }
  }
  Serial.print("\n\r\n\rThat's it.\r\n");
  ds.reset_search();
```

```
  return;
}

void loop(void) {
  // nothing to see here
}
```

## A.2 Übermittlung der gemessenen Daten

```
#include <OneWire.h>
#include <DallasTemperature.h>

// Data wire is plugged into pin 8 on the Arduino
#define ONE_WIRE_BUS 8

// Setup a oneWire instance to
// communicate with any OneWire devices
OneWire oneWire(ONE_WIRE_BUS);

// Pass our oneWire reference to Dallas Temperature.
DallasTemperature sensors(&oneWire);

DeviceAddress Thermometer1 =
{ 0x28, 0x50, 0x03, 0xB7, 0x03, 0x00, 0x00, 0x8D };
DeviceAddress Thermometer2 =
{ 0x28, 0x8C, 0xD7, 0x49, 0x03, 0x00, 0x00, 0x11 };
DeviceAddress Thermometer3 =
{ 0x28, 0xE9, 0x88, 0x49, 0x03, 0x00, 0x00, 0xC3 };
DeviceAddress Thermometer4 =
{ 0x28, 0xB7, 0xA7, 0x49, 0x03, 0x00, 0x00, 0x69 };
DeviceAddress Thermometer5 =
{ 0x28, 0x93, 0xA5, 0x60, 0x03, 0x00, 0x00, 0x63 };
DeviceAddress Thermometer6 =
{ 0x28, 0x37, 0x2C, 0x61, 0x03, 0x00, 0x00, 0x54 };
DeviceAddress Thermometer7 =
{ 0x28, 0xC9, 0xB2, 0x49, 0x03, 0x00, 0x00, 0x4C };
DeviceAddress Thermometer8 =
{ 0x28, 0x17, 0x3E, 0xB7, 0x03, 0x00, 0x00, 0x15 };

void setup(void)
```

```
{
  // start serial port
  Serial.begin(9600);
  // Start up the library
  sensors.begin();
  // set the resolution to 16 bit (good enough?)
  sensors.setResolution(Thermometer1, 16);
  sensors.setResolution(Thermometer2, 16);
  sensors.setResolution(Thermometer3, 16);
  sensors.setResolution(Thermometer4, 16);
  sensors.setResolution(Thermometer5, 16);
  sensors.setResolution(Thermometer6, 16);
  sensors.setResolution(Thermometer7, 16);
  sensors.setResolution(Thermometer8, 16);
}

void loop(void)
{
  delay(100);
  //Serial.print("Getting temperatures...\n\r");
  sensors.requestTemperatures();

  float tempsend1 = sensors.getTempC(Thermometer1);
  float tempsend2 = sensors.getTempC(Thermometer2);
  float tempsend3 = sensors.getTempC(Thermometer3);
  float tempsend4 = sensors.getTempC(Thermometer4);
  float tempsend5 = sensors.getTempC(Thermometer5);
  float tempsend6 = sensors.getTempC(Thermometer6);
  float tempsend7 = sensors.getTempC(Thermometer7);
  float tempsend8 = sensors.getTempC(Thermometer8);
  char temp[10];
  String tempstring;

  dtostrf(tempsend1,1,2,temp);
  tempstring = String(temp) + ",";

  dtostrf(tempsend2,1,2,temp);
  tempstring += String(temp) + ",";

  dtostrf(tempsend3,1,2,temp);
  tempstring += String(temp) + ",";
```

```
  dtostrf(tempsend4,1,2,temp);
  tempstring += String(temp) + ",";

  dtostrf(tempsend5,1,2,temp);
  tempstring += String(temp) + ",";

  dtostrf(tempsend6,1,2,temp);
  tempstring += String(temp) + ",";

  dtostrf(tempsend7,1,2,temp);
  tempstring += String(temp) + ",";

  dtostrf(tempsend8,1,2,temp);
  tempstring += String(temp)+"\n\r";

    Serial.print(tempstring);
}
```

## A.3 Schreiben der Daten in CSV-Datei

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
using System.IO;

namespace thermobox
{
  public partial class Form1 : Form
  {
    private SerialPort port;
    private string file;

    private BackgroundWorker bw = new BackgroundWorker();
```

```
public Form1()
{
  InitializeComponent();
  System.Globalization.CultureInfo ci =
    new System.Globalization.CultureInfo("en-US");
  System.Threading.Thread.CurrentThread.CurrentCulture
     = ci;
  bw.WorkerSupportsCancellation = true;
  bw.DoWork += new DoWorkEventHandler(bw_DoWork);
  bw.WorkerReportsProgress = true;
  bw.ProgressChanged +=
    new ProgressChangedEventHandler(bw_ProgressChanged);
  string[] ports = SerialPort.GetPortNames();
  comboBoxPorts.Items.Clear();
  // Display each port name to the console.
  foreach (string port in ports)
  {
    comboBoxPorts.Items.Add(port);
  }
  buttonStop.Enabled = false;
}

private void buttonBrowseCSV_Click(
             object sender, EventArgs e)
{
  if (saveFileDialog.ShowDialog() == DialogResult.OK)
  {
    textBoxCSVFile.Text = saveFileDialog.FileName;
    file = saveFileDialog.FileName;
  }
}

private void buttonStart_Click(object sender, EventArgs e)
{
  if (comboBoxPorts.SelectedIndex == -1) return;
  port = new SerialPort(comboBoxPorts.Items
      [comboBoxPorts.SelectedIndex].ToString(),9600);
  try
  {
    writeTitleCSV(file);
    port.Open();
```

```
      port.DiscardInBuffer();
      //port.DataReceived +=
        new SerialDataReceivedEventHandler(
            DataReceivedHandler);
      if (bw.IsBusy != true)
      {
         bw.RunWorkerAsync();
         buttonStart.Enabled = false;
         buttonStop.Enabled = true;
      }
    }
    catch (Exception)
    {
    }
  }

private void DataReceivedHandler(
                    object sender,
                    SerialDataReceivedEventArgs e)
{
   SerialPort sp = (SerialPort)sender;
   string indata = sp.ReadExisting();
   //textBoxLog.AppendText(indata);
   indata=DateTime.Now.ToLongTimeString()+","+indata;
   writeToCSV(file, indata);
}

private void buttonExit_Click(object sender, EventArgs e)
{
  if (port != null)
  {
    if (port.IsOpen) port.Close();
  }
   this.Close();
}

private void buttonStop_Click(object sender, EventArgs e)
{
  if (bw.WorkerSupportsCancellation == true)
  {
    bw.CancelAsync();
```

```
  }
  buttonStart.Enabled = true;
  buttonStop.Enabled = false;
}

public bool writeToCSV(string fileName, string row)
{
  StreamWriter writer = new StreamWriter(fileName, true);
  writer.WriteLine(row);
  writer.Close();
  return true;
}

public bool writeTitleCSV(string fileName)
{
  StreamWriter writer = new StreamWriter(fileName, false);
  writer.WriteLine("time,T1,T2,T3,T4,T5,T6,T7,T8");
  writer.Close();
  return true;
}

private void bw_ProgressChanged(
  object sender, ProgressChangedEventArgs e)
{
  textBoxLog.AppendText(e.UserState.ToString()+"\n");
}

private void bw_DoWork(object sender, DoWorkEventArgs e)
{
  BackgroundWorker worker = sender as BackgroundWorker;

  while(true)
  {
     if ((worker.CancellationPending == true))
     {
        e.Cancel = true;

        break;
     }
     else
     {
```

```
        string indata = port.ReadLine();
        //textBoxLog.AppendText(indata);

        indata = DateTime.Now.ToLongTimeString()+"."+
           DateTime.Now.Millisecond.ToString() + "," + indata;

        writeToCSV(file, indata);

        worker.ReportProgress(100, indata);
        System.Threading.Thread.Sleep(50);
        }
    }
    if (port != null)
    {
        if (port.IsOpen) port.Close();
    }
  }
 }
}
```

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Literatur

[1] http://www.projekt-info.org/.

[2] P. Fritzson, *Principles of Object-Oriented Modeling and Simulation With Modelica 2.1.* IEEE Press, 2004.

[3] Modelica Association, *Modelica: A Unified Object-Oriented Language for Physical Systems Modeling, Language Specification Version 3.2.* 2010.

[4] W. Geller, *Thermodynamik für Maschinenbauer: Grundlagen für die Praxis.* Springer-Lehrbuch, Springer, 2006.

[5] C. Salvatori, *Energieeffizienz von Werkzeugmaschinen – Planung und Aufbau eines Versuchstandes zur energetischen Untersuchung eines Kugelgewindetriebes.* Bachelorarbeit, Technische Universität Wien, 2012.

[6] Siemens AG, *Projektierungshandbuch Synchronmotoren 1FT6.* Ausgabe 10, 2005.

[7] MAXIM, *DS18B20 Programmable Resolution 1-Wire Digital Thermometer.*

[8] http://www.arduino.cc/.

[9] http://www.hacktronics.com/.

[10] http://www.3ds.com/products/catia/portfolio/dymola.

[11] Elmqvist, H. and Otter, M., "Methods for tearing systems of equations in object-oriented modelling," in *Proceedings ESM'94 European Simulation Multiconference, Barcelona, Spane, June 1.-3.*, pp. 326–332, 1994.

[12] Petzold, L.R., *A description of DASSL: a differential/algebraic system solver.*

[13] http://www.maplesoft.com/products/maplesim/index.aspx.

[14] J. Wensch, "Beiträge zur geometrischen integration und anwendungen in der numerischen simulation, kapitel 5." Habilitationsschrift Martin-Luther-Universität Halle-Wittenberg, 2004.

[15] C. Kral and Haumer, *Object Oriented Modeling of Rotating Electrical Machines, Advances in Computer Science and Engineering.* 2011.

[16] http://www.engineeringtoolbox.com/air-properties-d_156.html.