ACTIVATE™

solidThinking®

Coselica Toolbox 1.0

for solidThinking Activate 2016

Learn more at solidThinking.com

An Altair Company

Contact Us

solidThinking Activate Support

Welcome to the solidThinking Community! If you are a solidThinking customer and require technical support or more information about solidThinking Activate, you can browse our forum, watch videos, find documentation, and contact us at *www.solidthinking.com*.

Altair Support

If you are an Altair HyperWorks customer and require support information, please visit the HyperWorks website (*www.altairhyperworks.com*). The website also provides tips and tricks, training course schedules, training & tutorial videos, and more!

Intellectual Property Rights Notice:
Copyrights, Trademarks, Trade Secrets, Patents & Third Party Software Licenses

solidThinking Activate™ 2016

Special Notice: Pre-release versions of solidThinking software are provided 'as is', without warranty of any kind. Usage is strictly limited to non-production purposes.

solidThinking Inc. 1820 E. Big Beaver Road, Troy, MI 48083

Third Party Software licenses

Software Security Measures

solidThinking Inc. and its subsidiaries and affiliates reserve the right to embed software security mechanisms in the Software for the purpose of detecting the installation and/or use of illegal copies of the Software. The Software may collect and transmit non-proprietary data about those illegal copies. Data collected will not include any customer data created by or used in connection with the Software and will not be provided to any third party, except as may be required by law or legal process or to enforce

Contents

Chapter 1

Physical Component Modeling in solidThinking Activate

1.1 Introduction

solidThinking Activate blocks communicate data with other blocks through ports that are specified as inputs and outputs. Produced data is placed on a block output port and retrieved through the input port of one or more other blocks.

Even though it is possible to model many physical systems containing components in **solidThinking Activate**, this generally requires translating the implicit equations representing the behavior of the physical components into explicit form and as a result, the resulting **solidThinking Activate** diagrams may differ significantly from the original component-based diagrams.

The fundamental reason for which standard **solidThinking Activate** blocks are not well suited for modeling physical components is that component ports are connectors which, unlike block ports, need not be specified as inputs or outputs.

1.2 Components

A component places general constraints on its port signals. For example an electrical resistor specifies relations between the currents and voltages on its ports (in particular that the sum of currents flowing in is zero and that the difference of voltages is proportional to the current), but the component does not require any current or voltage to be an input or output. So the component model should be able to represent the behavior of the component regardless of the way it is used in a diagram.

solidThinking Activate provides facilities to create models including both blocks and components in a consistent way. For that, it uses the methodology originally developed for Scicos. Components and blocks can co-exist in the same diagram but block and component ports cannot be interconnected. The connection of the two worlds is done through Special blocks containing mixed port types.

Components in **solidThinking Activate** are called implicit blocks. The behavior of implicit blocks cannot be defined conveniently through explicit functions. In a regular block, a function may provide the value of the output as a function of the input, but in an implicit block since the inputs and outputs cannot be

specified, at best such a description would require multiple functions considering various input/output scenarios. But even then the resulting simulation code would be inefficient.

1.3 Modelica

In **solidThinking Activate**, the behavior of implicit blocks is specified through symbolic equations. Unlike regular blocks that may be regarded as black boxes, implicit blocks expose the implicit equations of the corresponding component. **solidThinking Activate** then can, through symbolic manipulation, use these equations to produce efficient code for models obtained from the interconnection of implicit blocks. The symbolic specification of the behavior of implicit blocks in **solidThinking Activate** is done using the MODELICA language (see www.modelica.org).

1.4 Application and Implementation in solidThinking Activate

As introduced above, **solidThinking Activate** formalism allows for mixing implicit and explicit blocks. The actual use of those components is possible through the use of a toolbox, called COSELICA which provides a compiler and a library of blocks.

The COSELICA library is described in chapter 2 and Use Cases are presented in the following chapters.

Chapter 2

COSELICA Toolbox for Physical Component Modeling in solidThinking Activate

2.1 Introduction

COSELICA is a library of *physical components* for **solidThinking Activate** covering various physical domains, such as *mechanics*, *electrics*, *electronics*, and *thermodynamics*.

These components are provided as **solidThinking Activate** blocks and they can be found in hierarchical palettes named `Coselica` under the Toolbox category, provided the COSELICA add-on is installed.

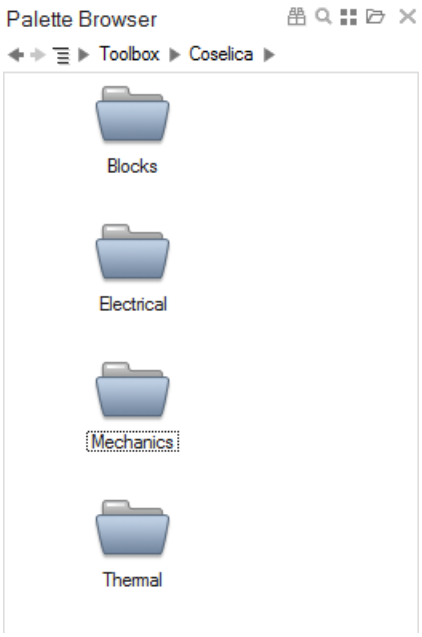


Figure 2.1: Palette Browser in **solidThinking Activate**

When dealing with physical components, all blocks have special connector ports and connecting them

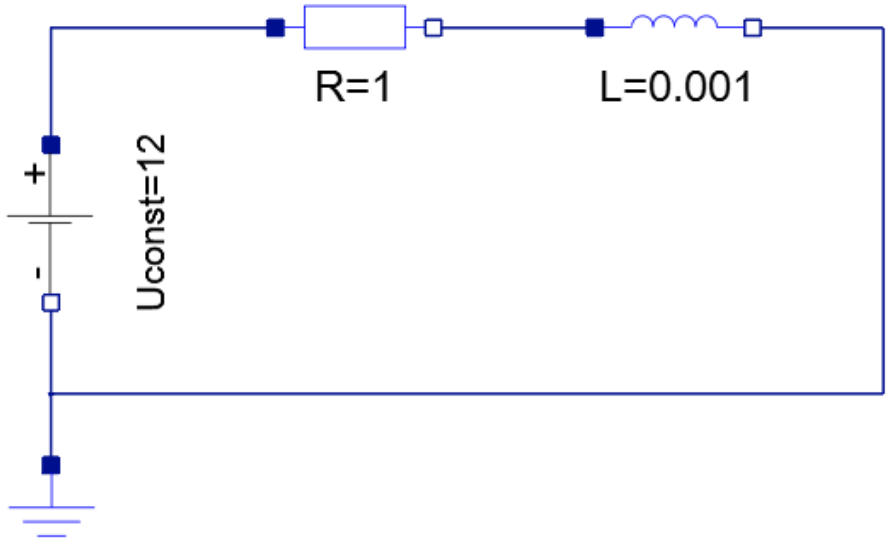


Figure 2.2: First Simple Model (`ElectricalCoil`)

is like assembling a physical network. Such a physical network, assembled with COSELICA, is shown in fig. 2.2. It represents the model of an electrical coil (`Resistor` and `Inductor` in series) driven by a constant voltage source (`Uconst=12`).

In the model, there are two types of special connector ports: a filled blue square (positive pin) and an empty blue square (negative pin). Modeling this electrical model is just like drawing an ordinary schematic diagram, simply using all components needed from the `Electrical` ▶ `Analog` palette (cf. sec. 2.4.2).

2.2 Origin and Further Information

COSELICA library is internally implemented using the MODELICA language (`https://www.modelica.org`).

For the most part COSELICA is a *subset* of the freely available **Modelica Standard Library** (`https://www.modelica.org/libraries`), herein after referred to as MSL.

This subset had to be modified to make it work within the Physical Component library of **solidThinking Activate**, which currently supports only a subset of the MODELICA specifications 2.x (`https://www.modelica.org/documents`).

A certain fraction of COSELICA (the `Planar` palette; cf. sec. 2.4.3) is inspired by the `MultiBody` package of the MSL, adopted the graphical appearance and the user-visible structure. But it is a *significantly differing implementation*.

COSELICA contains moreover, a number of blocks, which are unique and do not originate from MSL. Please refer to sec. **??**, for a more detailed description of the content and basic usage of COSELICA.

2.3 Acausal and Causal Modeling

Conventional **solidThinking Activate** blocks have clearly defined *input* and *output* ports, where an input can be regarded as *cause* and an output as *effect*. Models built solely with these blocks are *causal* models. In contrast to that, most COSELICA blocks have neither identifiable input nor output ports. Instead, they have connector ports which propagate physical quantities (e.g. electrical voltage and current), and it is a priori undecided what quantity *causes* which *effect*. Models like that of fig. 2.2 are called *acausal*.

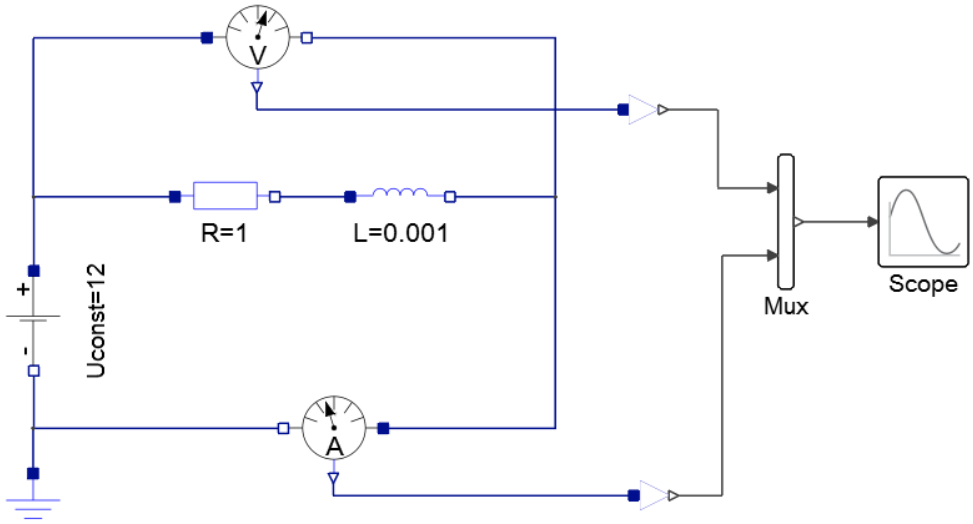


Figure 2.3: Simple Model for Simulation (`ElectricalCoilWithSensors`)

If we actually want to simulate and retrieve results from a purely acausal model, like that of fig. 2.2, then we have to introduce some causality into the model. In other words, we need to decide which quantities of the system are of interest for us. In COSELICA, this is usually achieved by adding appropriate sensors to the model, e.g. as shown in fig. 2.3. Here, we have added a `VoltageSensor` and a `CurrentSensor` in order to "measure" the voltage across and the current flowing through the coil. Both quantities are provided by the sensor blocks as *real signals*.

There are a number of COSELICA blocks (cf. sec. 2.4.1) available in order to deal with real signals. However, in most cases it is preferable to make use of the rich and powerful set of conventional **solidThinking Activate** blocks (like `Mux` and `Scope` in fig. 2.3) for further processing, saving, or plotting of results.

A simulation time up to 0.01 yields the results as plotted in fig.2.4. As one might expect, the voltage across the coil remains constantly at 12 (black curve). The current flowing through the coil (green curve) exhibits a first-order delay characteristic, due to the dynamic behavior of the `Inductor`, i.e. the current is initially zero and cannot follow instantaneously the voltage source step from 0 to 12 at t=0.

Please note, that direct connections between COSELICA and conventional **solidThinking Activate** blocks are not allowed. For this purpose, one has to use special interface blocks (see `Coselica` ▶ `Blocks` ▶ `Interfaces`).

In fig. 2.3, we have used `RealOutput` twice for a connection from COSELICA to **solidThinking Activate**. There is also a `RealInput` block for connections the other way around.

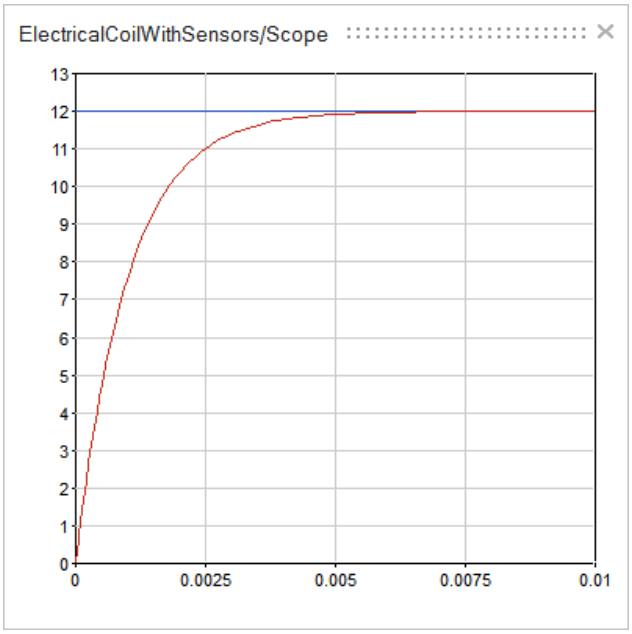


Figure 2.4: Simulation Results (`ElectricalCoilWithSensors`)

Advantages of Acausal Modeling

Acausal modeling is nowadays very popular in engineering for the following benefits:

- time needed for modeling is decreased significantly,
- acausal models are easier to interpret,
- acausal models are highly reusable and shareable.

We will now derive and implement a purely causal model in order to justify the claimed advantages of acausal modeling by example. This model shall be equivalent to that of fig. 2.3.

We denote the voltages across resistor and inductor by $u_R(t)$ and $u_L(t)$. The constant source voltage shall be $u_s(t)$ and the current flowing through the coil is denoted by $i(t)$. Consequently, we can describe the dynamic behavior of the electrical coil (cf. fig. 2.2) using three equations:

$$u_s(t) = u_R(t) + u_L(t)\,, \tag{2.1}$$

$$u_R(t) = R \cdot i(t)\,, \tag{2.2}$$

$$u_L(t) = L\frac{di(t)}{dt}\,. \tag{2.3}$$

We are only interested in $u_S(t)$ and $i(t)$, but not in $u_R(t)$ and $u_L(t)$. Thus, we have to derive equations for $u_s(t)$ and $i(t)$, which do not depend on u_R and u_L. $u_s(t)$ is a priori known to be

$$u_s(t) = 12\,. \tag{2.4}$$

By insertion of eq. 2 & 3 in eq. 1, we get an ordinary first order differential equation

$$\frac{di(t)}{dt} = \frac{1}{L}\left(u_s(t) - R \cdot i(t)\right)\,, \tag{2.5}$$

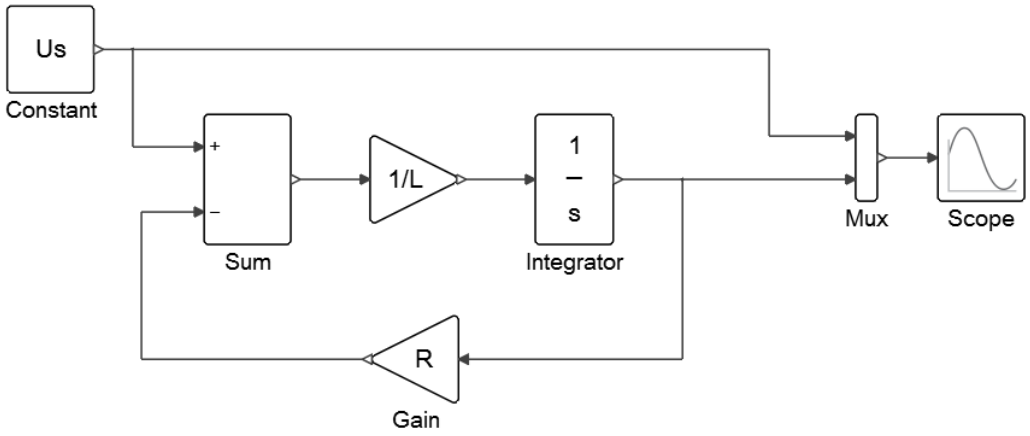


Figure 2.5: Purely Causal Model (`ElectricalCoilPurelyCausal`)

which can be implemented by using conventional **solidThinking Activate** blocks as shown in fig. 2.5. The interested reader may easily verify by himself, that simulation of fig. 2.5 (with initial condition $i(0) = 0$) yields the very same results as shown in fig. 2.4.

2.4 Available Domains

A technical system is more often than not a physical multi-domain system (e.g. a DC motor consists of an electrical *and* a mechanical part) and thus requires a modeling environment covering multiple physical domains and interconnections between them. The *available domains* within COSELICA are described in the following and their usage is illustrated by rather simple introductory examples.

2.4.1 `Blocks` (Real Signals)

All blocks in the `Blocks` palette are causal, i.e. they have input and outputs ports only. They are commonly used to generate and process *real signals* and resemble a subset of the `Modelica.Blocks` package of the MSL. Almost all blocks of this palette are equivalent or very similar to their counterparts found in the MSL.

Unique to COSELICA

The following blocks do not originate from the MSL:

- `Interfaces` ▶ `RealInput` and `Interfaces` ▶ `RealInput`
- `Routing` ▶ `DeMultplexVector2` and `Routing` ▶ `MultplexVector2`
- `Math` ▶ `Vectors`▶ ... (except `Sum` and `MatrixGain`)
- `Math` ▶ `Nonlinear` ▶ `Hysteresis` and `Math` ▶ `Nonlinear` ▶ `RateLimiter`

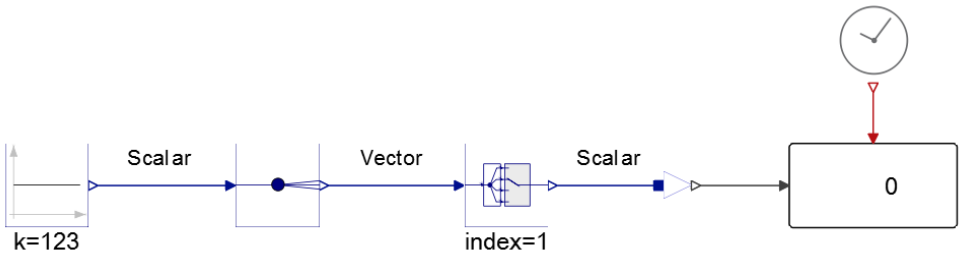


Figure 2.6: Type Coercion between Scalars and Vectors (`ReplicatorExtractor`)

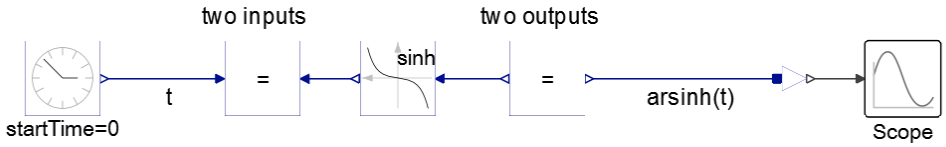


Figure 2.7: Reversal of Causality (`ReversalCausality`)

Vector-Valued Real Signals

Most blocks work with scalar real signals only, but working with vector-valued real signals is possible as well (see `Routing` and `Math` ► `Vectors`).

Please note, there is no automatic *type coercion* between scalar real signals and real signal vectors of length one. However, such type coercions can be implemented using `Routing` ► `Replicator` and `Routing` ► `Extractor` (e.g. in fig. 2.6).

Reversal of Causality

Due to the nature of the underlying MODELICA language, it is possible to change an input into an output port and vice versa. This can be done with `Math` ► `TwoOutputs` and `Math` ► `TwoInputs`. For example, it is possible to implement the missing inverse hyperbolic sine function by reverting the causality of the `Sinh` block as shown in fig. 2.7.

Example: Generation of a PWM Signal

PWM (pulse width modulation) signals are widely used for electrical power control (e.g. electrical heating, light dimmers, dc motors). Fig. 2.8 shows how a PWM signal can be generated. Its frequency and duty cycle can be adjusted by setting the parameter k of `Constant (Duty Cycle)` and the parameter $period$ of `SawTooth (Frequency)`.

The generated PWM signal, for k=30 (%) and $period$=0.0005, is shown in fig. 2.9.

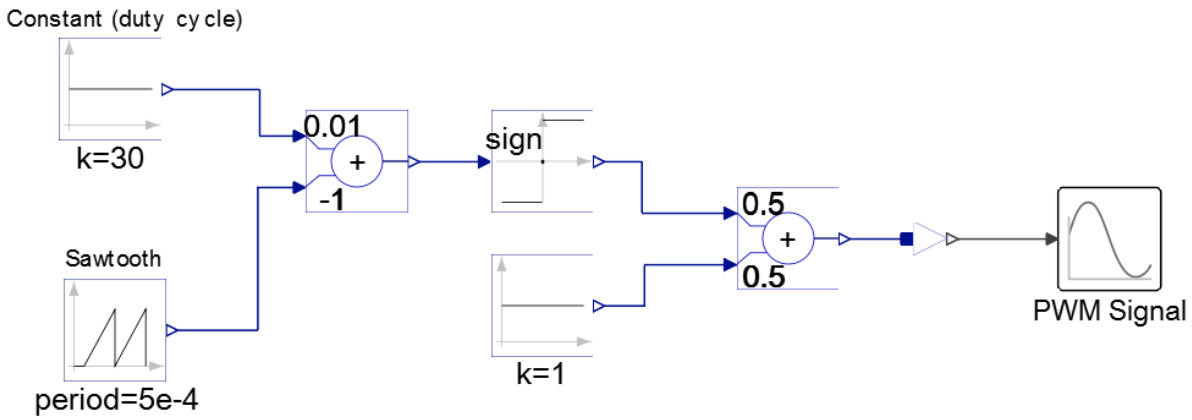
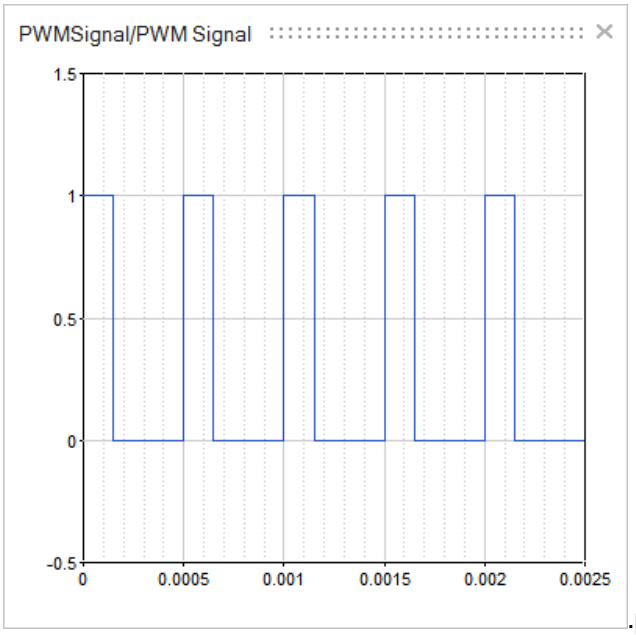


Figure 2.8: Generation of a PWM Signal (`PWMSignal`)



.png

Figure 2.9: PWM Signal ($k = 30$; $period = 0.0005$)

2.4.2 Electrical ► Analog

The Electrical ► Analog palette contains various analog electric and electronic components, such as resistor, capacitor, transformer, diode, transistor, switches, sources, sensors, and others. It resembles a subset of the Modelica.Electrical.Analog package of the MSL. All blocks of this palette are equivalent or very similar to their counterparts found in the MSL.

Links to other Domains

Blocks (Real Signals). Some blocks of the Sources palette (SignalVoltage and SignalCurrent), all blocks of the Sensors palette, some blocks of the Basic palette (VariableResistor, VariableCapacitor, VariableInductor), and some of the Ideal palette (IdealOpeningSwitch, IdealClosingSwitch) provide connections to the domain of real signals (cf. sec. 2.4.1).

Mechanics ► Rotational. There a two blocks, Basic ► EMF and Basic ► EMF0, which describe the transformation of electrical energy into rotational mechanical energy (cf. sec. 2.4.3)

Mechanics ► Translational. There a two blocks, Basic ► TranslationalEMF and Basic ► TranslationalEMF0, which describe the transformation of electrical energy into translational mechanical energy (cf. sec. 2.4.3).

Thermal ► HeatTransfer. The block Basic ► HeatingResistor provides a model of a temperature dependent resistor and hereby a connection to the domain of thermal heat transfer (cf. sec. 2.4.4).

Example: DC Motor driven by PWM 1-Quadrant controller

Fig. 2.10 shows a model of simple DC motor, which is driven by a PWM 1-quadrant converter. The very basic electrical component of a DC motor is the electrical anchor coil (Resistor and Inductor in series). The relation between the electrical current (flowing through the coil) and the mechanical torque exerted at the motor shaft is modeled by the electric/mechanic transformer block EMF0. This block is a link between the electrical and rotational mechanics domain, i.e. the right port of EMF0 resembles the motor shaft propagating mechanical quantities (angle and torque). There is a constant supply voltage Uconst=12. It is controlled by a 1-quadrant controller, consisting of a Diode and an IdealClosingSwitch. The switch is triggered by a PWM signal (cf. example sec. 2.4.1). In reality the IdealClosingSwitch would be a semiconductor switch (e.g. a MOSFET).

For testing purposes, we generate a PWM signal whose duty cycle rises within 0.015s from 0% to 100% (PWM Duty Cycle) and we “measure” the exerted torque at the motor shaft using two blocks (TorqueSensor and Fixed) from the Mechanics ► Rotational palette (cf. sec. 2.4.3).

Please note, we do not care about mechanical inertia of the rotor, because right now everything mechanical is kept fixed. We will have a look at the mechanical part of the motor later, in sec. 2.4.3.

Fig. 2.11 shows the simulation result of our simple test. The ramp of the increasing duty cycle (black curve), the corresponding PWM signal (green curve), which closes the switch (IdealClosingSwitch)

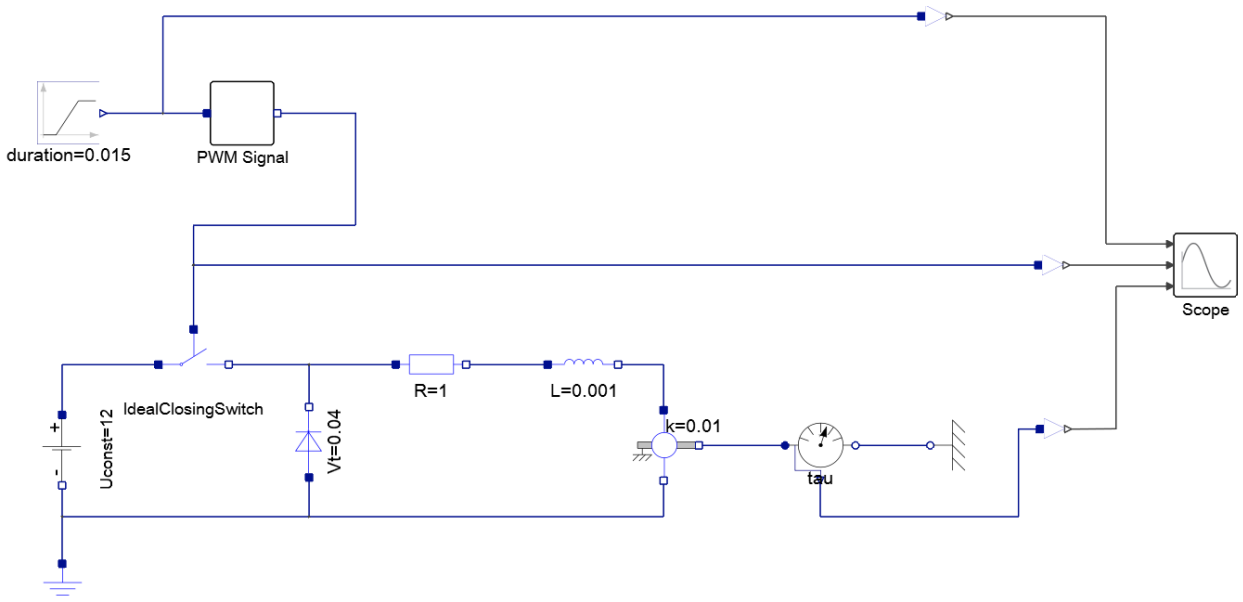


Figure 2.10: DC Motor driven by PWM Controller (`DCMotorWithoutInertia`)

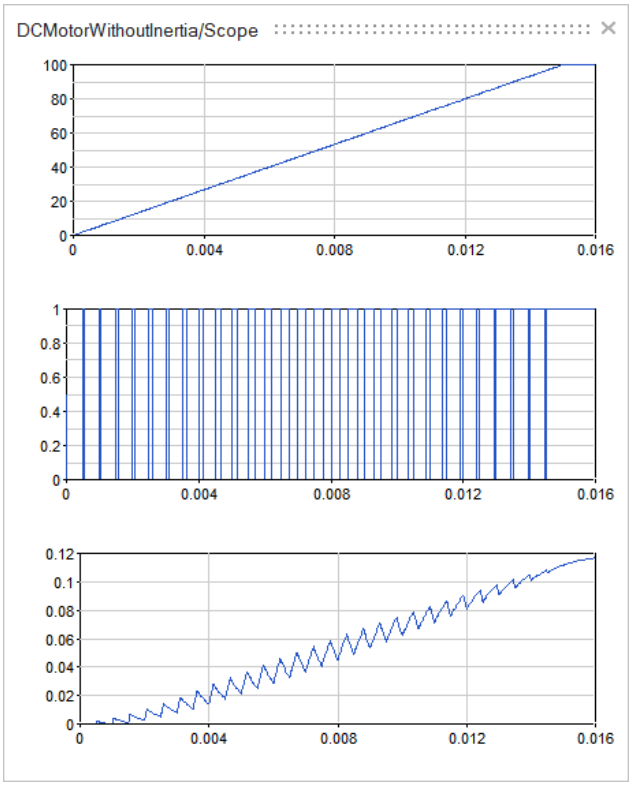


Figure 2.11: Duty Cycle, PWM Signal, and Torque (`DCMotorWithoutInertia`)

when positive, and the exerted mechanical torque (red curve) at the motor shaft. The torque exhibits ripple, due to the discontinuous excitation by the PWM signal. The torque ripple in our model is increased when the frequency of the PWM signal is reduced and vice versa.

2.4.3 Mechanics

The Mechanics palette contains components to model 1-dimensional (rotational and translational) and 2-dimensional (planar) mechanical systems. It is organized in three sub-palettes namely Rotational, Translational, and Planar. They are described in the following.

Rotational

The Rotational palette contains components to model 1-dimensional rotational mechanical systems. Different types of gearboxes, shafts with inertia, external torques, spring/damper elements, frictional and backlash elements, sensors to measure angle, angular velocity, angular acceleration and the cut-torque of a flange are included. It resembles a subset of the Modelica.Mechanics.Rotational package of the MSL. Almost all blocks of this palette are equivalent or very similar to their counterparts found in the MSL.

The direction of a rotation is indicated by the sign of a corresponding variable. How to interpret the sign of such a variable is described in the MSL[1].

Unique to COSELICA

The following blocks do not originate from the MSL:

- Components ▶ Free
- Components ▶ Freewheel
- Components ▶ IdealDifferential

Unconnected ports (propagating physical quantities) are not allowed. Unconnected ports can be avoided by connecting them to Components ▶ Free.

Links to other Domains

Blocks (Real Signals). Some blocks of the Sources palette (e.g. Position, Speed, Acceleration, Torque, ...), all blocks of the Sensors palette, and some of the Components palette (Brake, Clutch, OneWayClutch) provide connections to the domain of real signals (cf. sec. 2.4.1).

Mechanics ▶ Translational. The block Components ▶ IdealGearR2T describes the transformation of rotational motion into translational motion and provides therefore a connection to the translational mechanics domain (cf. sec. 2.4.3).

[1] Modelica.Mechanics.Rotational.UsersGuide.SignConventions

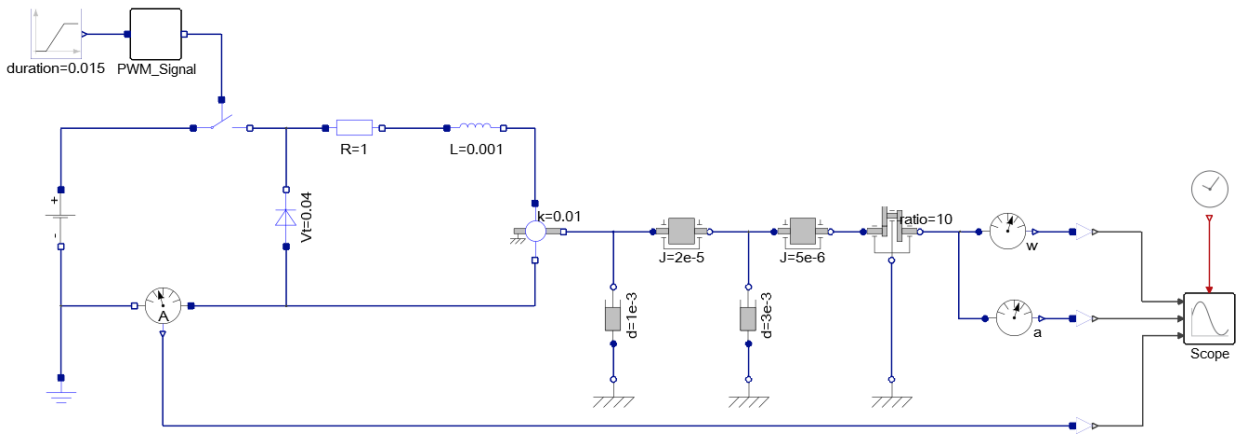


Figure 2.12: DC Motor with Speed Reducing Gear (`DCMotor`)

Example: DC Motor with Speed Reducing Gear

We have enhanced the model of fig. 2.10 in order to build a physical multi-domain (electrical/mechanical) model of a DC motor with a speed reducing gear as shown in fig. 2.12. The *rotor* of the DC motor is modeled by its `Inertia` and we take losses, due to bearing friction (`Damper`), into account. The friction is modeled here simply as viscous damping. However, it would be possible to use other friction models (such as Coulomb and Stribeck) by means of the block `BearingFriction`.

Attached to the rotor shaft is a speed reducing *gear* with ratio 10 (`IdealGear`). We take the gear `Inertia` and losses (again simply a viscous `Damper`), as seen from the gear input shaft, into account. We do not consider any backlash of the gear, but we could do so by using `ElastoBacklash`. Finally, we have a mechanical `SpeedSensor` and acceleration sensor (`AccSensor`) connected to the output shaft of the gear and we are interested in the electrical current drawn by the motor (`CurrentSensor`).

Running a simulation of the model reveals the startup behaviour of our *unloaded* DC motor as show in fig. 2.13. The output shaft of the gear reaches a rotational speed (black curve) of nearly 1 rpm in steady state. The corresponding acceleration (green curve) exhibits ripple during the PWM startup ramp, which is not surprising (cf. sec. 2.4.2; fig. 2.11). Furthermore, one can observe that the DC motor draws an electrical current (red curve) of nearly 12 in steady state.

Please note, that this DC motor model could actually be used to drive a mechanical system of choice just by connecting the system to be driven to the output shaft (right port) of the `IdealGear` block.

`Translational`

The `Translational` palette contains components to model 1-dimensional translational mechanical systems (sliding mass , mass with friction, spring, damper, and others).

It resembles a subset of the `Modelica.Mechanics.Translational` package of the MSL. Almost all blocks of this palette are equivalent or very similar to their counterparts found in the MSL.

The direction of a translation is indicated by the sign of a corresponding variable. How to interpret the

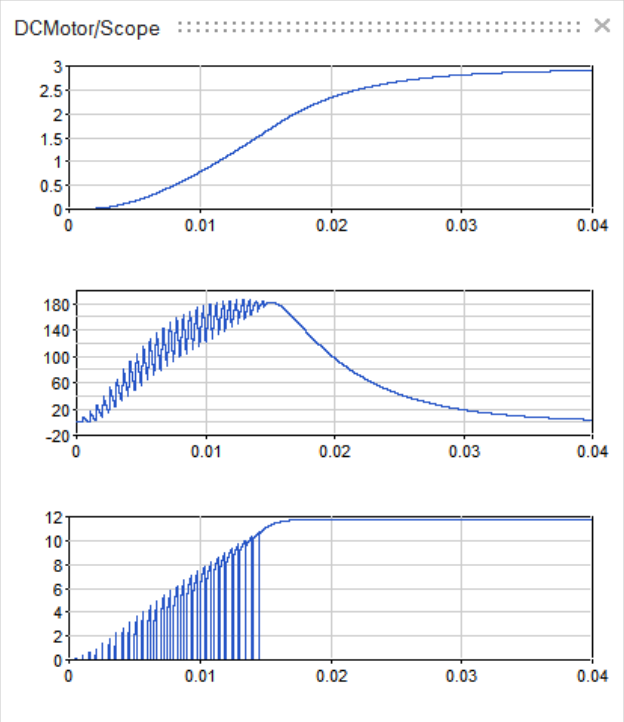


Figure 2.13: Speed, Acceleration, and Current (`DCMotor`)

sign of such a variable is described in the MSL[2].

Unique to COSELICA

The following blocks do not originate from the MSL:

- `Components` ▶ `Free`
- `Components` ▶ `MassWithWeight`
- `Components` ▶ `Pulley and Components` ▶ `ActuatedPulley`
- `Components` ▶ `Lever`

Unconnected ports (propagating physical quantities) are not allowed. Unconnected ports can be avoided by connecting them to `Components` ▶ `Free`.

Links to other Domains

Blocks (Real Signals). Some blocks of the `Sources` palette (e.g. `Position`, `Speed`, `Acceleration`, `Torque`, ...) and all blocks of the `Sensors` palette provide connections to the domain of real signals (cf. sec. 2.4.1).

[2] `Modelica.Mechanics.Translational.Examples.SignConvention`
`Modelica.Mechanics.Translational.Examples.WhyArrows`

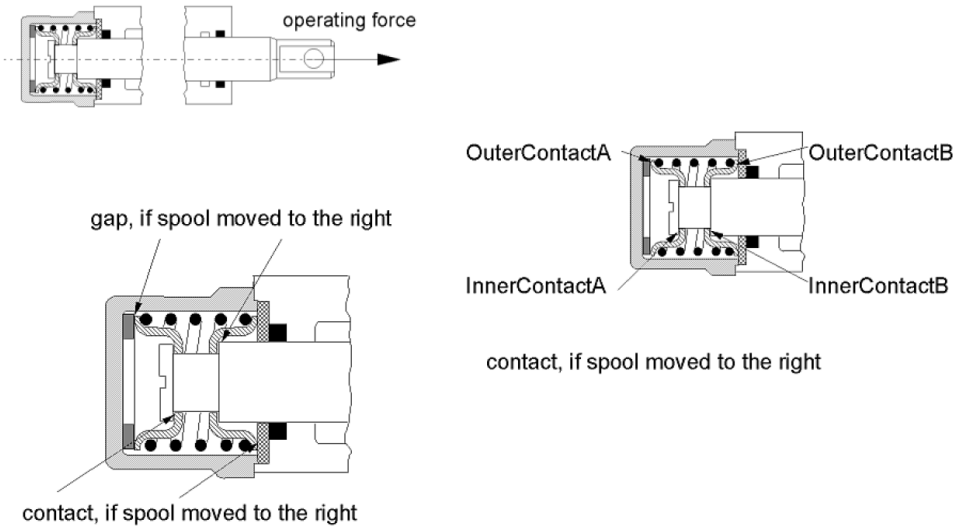


Figure 2.14: Spool Design using only one Spring for Preload (Source: MSL)

Mechanics ► Rotational. The block `Rotational ► Components ► IdealGearR2T` is not part of the `Translational` palette, but nevertheless provides a link between the translational and the rotational mechanics domain (cf. sec. 2.4.3).

Example: Preload of a Spool

Please note, that this example has been taken from the MSL[3]. It was slightly modified and rebuilt for COSELICA.

In force actuated hydraulic valves it is often necessary to hold the spool in a predefined (resting) position as long as the operating force is below a threshold value. If the force exceeds the threshold value a linear relation between force and position is desired. Some designs use only one mechanical spring to accomplish this behaviour. Relevant details of such a design are shown in fig. 2.14. At rest, the two *spring plates* are in contact with the *spool* and the *housing*. When the spool has moved, this is not true any more. One spring plate is still in contact to the other parts as before, but the other is not. There is one gap between this spring plate and the housing and another gap between the spring plate and the spool. This can happen at four places and depends on the direction of movement of the spool (cf. fig. 2.14).

The possible interchange of contact and gap (`OuterContactB`, `OuterContactB`, `InnerContactA`, `InnerContactB`) can be modeled with the `ElastoGap` block as shown in fig. 2.15. We are interested in the relation between actuation force and spool position. Thus, we are actuating the spool with a slowly ($freqHz$=0.01) varying `Sine (Operation Force)` and measuring the corresponding `Position` of the spool.

Simulation of the model yields the characteristic relation between force and position of the preloaded spool as shown in fig. 2.17. The characteristic looks expedient and exhibits a barely visible hysteresis behaviour. The existence of hysteresis might be surprising, but there is a high viscous friction present (see `Damper`).

[3] `Modelica.Mechanics.Translational.Examples.PreLoad`

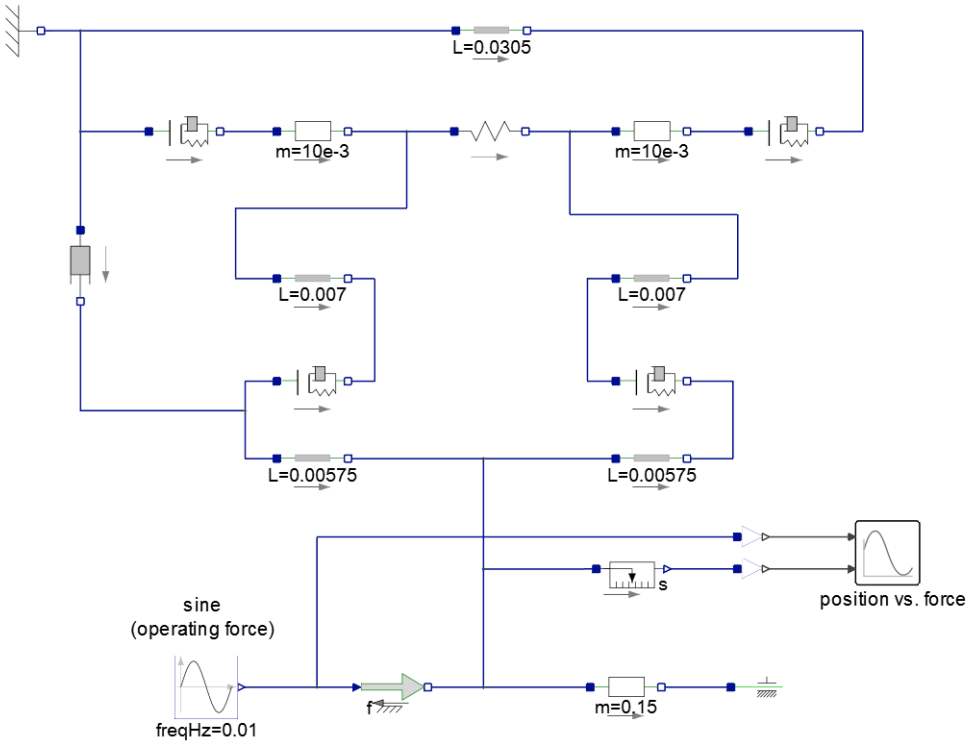


Figure 2.15: Model for Preload of a Spool (`SpoolPreLoad`)

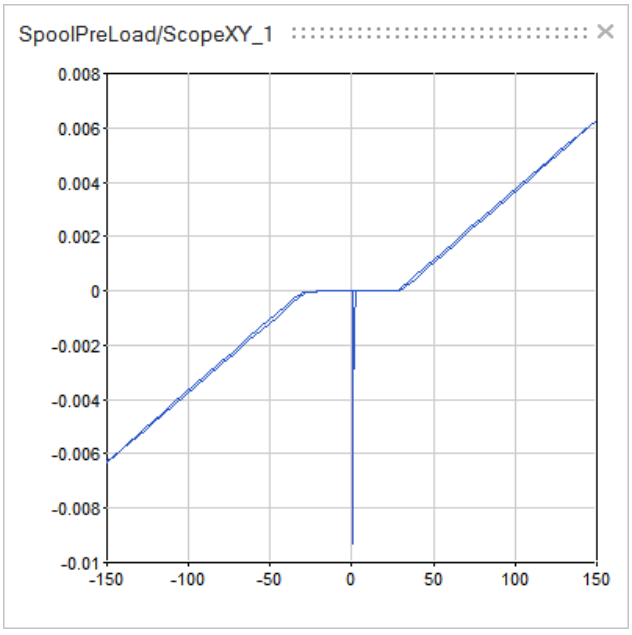


Figure 2.16: Spool Position vs. Operating Force at 0.01Hz (`SpoolPreLoad`)

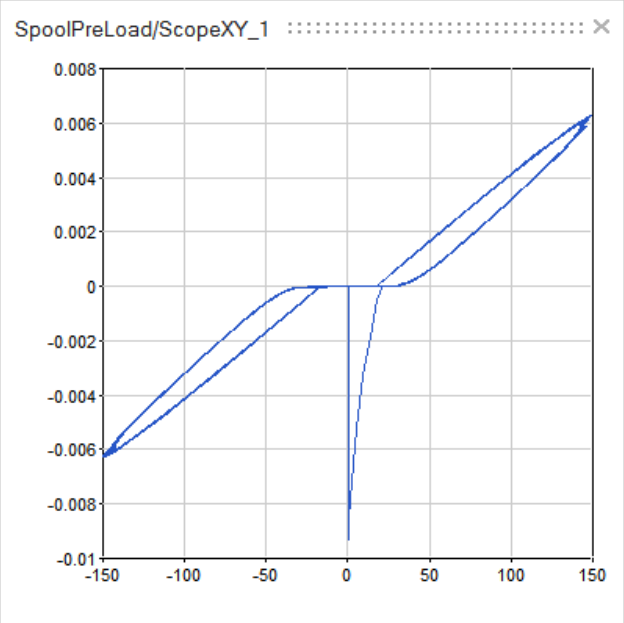


Figure 2.17: Spool Position vs. Operating Force at 0.1Hz (`SpoolPreLoad`)

So far, we have simulated the almost static behaviour of the system, by using a quite slowly varying sine wave actuation force. Furthermore, our model is truly suitable for an examination of system dynamics as well, because it takes relevant mechanical inertia (masses of spool and spring plates) into account. We might be interested now in the spool characteristic for a more quickly varying actuation force. Simulation with a frequency (parameter $freqHz$=0.1) ten times higher than before yields the characteristic as shown in fig. 2.17. It exhibits a clearly visible hysteresis behaviour.

`Planar`

The blocks of the `Planar` palette are used for modeling of rigid multi-body systems. This palette seems to be very similar to the `Modelica.Mechanics.Multibody` package of the MSL, due to its graphical appearance and user-visible structure. But in comparison to the MSL, it is based on a significantly differing implementation. The major differences are:

- restriction to 2-dimensional planar problems (2 translational, 1 rotational degree of freedom)
- special joint blocks have to be used in kinematic loops (see `LoopJoints` palette)
- connector ports propagate additionally velocities and accelerations (not user-visible)

Links to other Domains

Blocks (Real Signals). Some blocks of the `Forces` palette (`WorldForce`, `WorldTorque`, `FrameForce`) and all blocks of the `Sensors` palette provide connections to the domain of real signals (cf. sec. 2.4.1).

Mechanics ► Rotational. All actuated revolute joints (`Joints ► ActuatedRevolute` and `LoopJoints ► ActuatedRevolute`) and the `ActuatedRollingWheel` block support connections to the rotational mechanics domain (cf. sec. 2.4.3).

Mechanics ► Translational. Some blocks of the `Forces` palette (`LineForce` and `LineForceWithMass`) and all actuated prismatic joints (`Joints ► ActuatedPrismatic` and `LoopJoints ► ActuatedPrismatic`) support connections to the domain of translational mechanics (cf. sec. 2.4.3).

Usage of World and Fixed

Each interconnected network of components has to incorporate either exactly one `World` block and an arbitrary number of `Fixed` blocks or at least one `Fixed` block.

A `Fixed` block (with parameter r=[0,0]) instead of a `World` block may be used for systems without gravity.

`Fixed` blocks can be used to close kinematic loops, but this is not mandatory.

Kinematic Chains and Loops

It is possible to model kinematic chains and loops as well. For kinematic chains all prismatic and revolute joints have to be taken from palette `Joints`. However, every kinematic loop has to incorporate exactly three joints from the palette `LoopJoints` (see example below).

Usage of Sensors

In the planar mechanics domain we have 1 rotational and 2 translational degrees of freedom. Thus, some physical quantities are scalar (e.g. orientation angle, torque, ...) and others are vectors of length 2 (e.g. position, force, ...). All sensors with *round shape* provide a scalar output and all sensors with *rectangular shape* provide a vector output. The blocks of the `Blocks ► Routing` and the `Blocks ► Math ► Vectors` palettes are useful for processing the latter vector outputs.

The vector output of a sensor is basically resolved in the world frame (r=[0,0] with zero orientation angle). However, there are a number of sensors with an additional port marked “resolve”. Here, vector output is resolved in the “resolve” frame, i.e. in the frame that is connected to this additional port. Please note, that connections to “resolve” ports have no impact whatsoever on the mechanical properties of a system.

Example: Scotch Yoke Mechanism

The **Scotch Yoke**, depicted in fig. 2.18, is a well-known and simple planar mechanism for converting rotational motion into reciprocating translational motion and vice versa. A pin (dark red) is connected to a rotating part (light red). The pin is located in the slot of the yoke (light blue). The yoke is supported by bushings (dark blue) and allowed to travel horizontally back and forth driven by the rotating part.

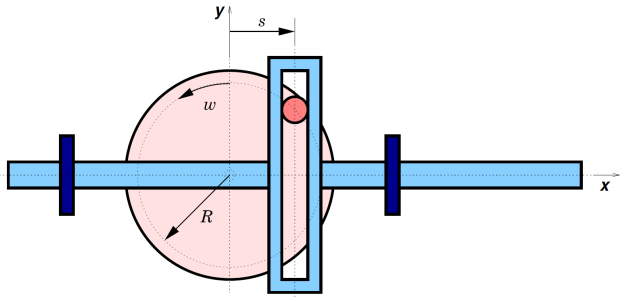


Figure 2.18: **Scotch Yoke** Mechanism

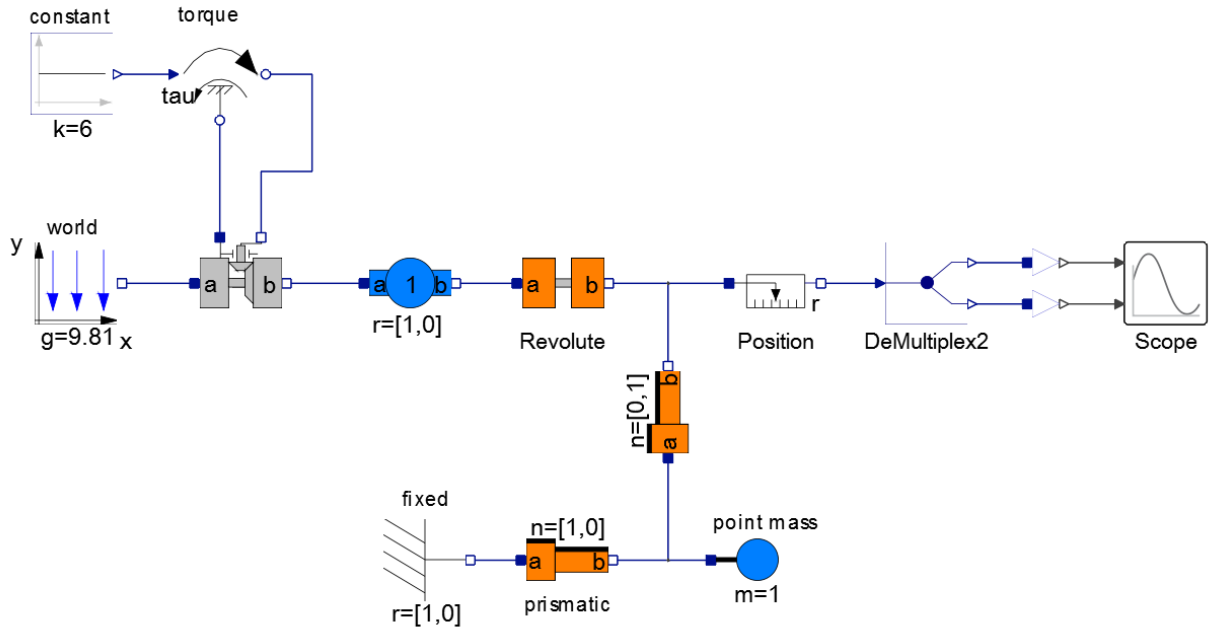


Figure 2.19: Model of **Scotch Yoke** Mechanism (`ScotchYoke`)

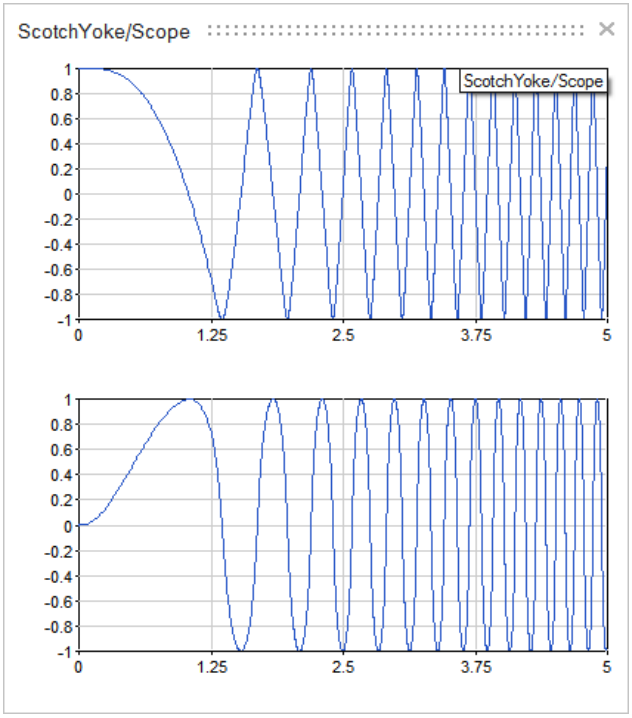


Figure 2.20: Simulated Pin Position of Scotch Yoke Mechanism (`ScotchYoke`)

A model of the **Scotch Yoke** is shown in fig. 2.19. Backlash and friction are neglected for the sake of simplicity. The effective radius R is assumed to be constant. Acceleration due to gravity is taken into account (`World`). An initial yoke position s=R=1 was presumed for the assembly of the model. The rotating part (`BodyShape`) bears mass and inertia, where the center of mass does not necessarily coincide with the center of rotation (`ActuatedRevolute`). The rotating part is driven by a constant `Torque`. The link between pin and yoke is modeled by a `Revolute` (connected to `BodyShape`) and a `Prismatic` joint in series. Finally, the motion of the yoke (`PointMass`) is restricted by another `Prismatic` joint which is connected to a `Fixed` frame, allowing the yoke to move horizontally only. The `Fixed` block closes a kinematic loop of four joints in series. As a general rule, one has to use exactly three `LoopJoints` (orange) in a kinematic loop.

The `Position` of the pin is "measured" and plotted as shown in fig. 2.20. The position vector is split into scalar values (`DeMultplex2`), i.e. x- (black curve) and y-coordinate (green curve). As indicated by the curves in fig. 2.20, the mechanism is traveling faster and faster with increasing time, this is due to our lossless modeling.

The model could be used for kinetostatic analysis of the mechanism as well. For this purpose, one would replace the `Torque` source by a `Position`, `Speed` or `Acceleration` source and add appropriate sensors (e.g. `CutForce` or `CutTorque`) in order to "measure" forces and torques of interest.

Furthermore, the mechanism could be driven by a more realistic source, e.g. by a electrical DC motor (cf. sec. 2.4.2).

2.4.4 Thermal ▶ HeatTransfer

The HeatTransfer palette contains components to model 1-dimensional heat transfer with lumped elements. It resembles a subset of the Modelica.Thermal.HeatTransfer package of the MSL. All blocks of this palette are equivalent or very similar to their counterparts found in the MSL.

Links to other Domains

Blocks (Real Signals). The block Components ▶ Convection, some blocks of the Sources palette (PrescribedTemperature and PrescribedHeatFlow), all blocks of the Sensors palette, and some blocks of the Celsius palette (ToKelvin, FromKelvin, PrescribedTemperature, TemperatureSensor) provide connections to the domain of real signals (cf. sec. 2.4.1).

Electrical ▶ Analog. The block Electrical ▶ Analog ▶ Basic ▶ HeatingResistor is not part of the HeatTransfer palette, but nevertheless provides a link between the thermal heat transfer and the electrical domain (cf. sec. 2.4.2).

Example: Insulated Steel Rod

How evolves the temperature at different locations on an insulated steel rod (initial temperature 20℃, length L, cross section area A, density ρ, specific heat capacity c, heat conductivity λ), when one of its ends gets connected to a heat source of higher temperature? The simulation of the model, shown in fig. 2.21, will give an answer to that question. The rod is modeled by three lumped HeatCapacitor blocks connected to their neighbors and the rod ends by ThermalConductor blocks. Please note, this model is just an approximation. However, its accuracy might be increased by using a higher number of lumped elements. For our purposes, we may expect to get reasonable approximations for the temperature at locations $\frac{L}{6}$, $\frac{3L}{6}$, and $\frac{5L}{6}$ on the rod.

Apart from the left rod end, perfect insulation is assumed. Thus, the right end is connected to a PrescribedHeatFlow of zero. The left end is not insulated and connected to something with a higher (200℃) temperature. Expressed more formally, we have here a boundary condition for the temperature at the left end, i.e. we have a PrescribedTemperature of 200℃ at this location.

We are "measuring" the temperature at different locations (right end, $\frac{L}{6}$, $\frac{3L}{6}$, and $\frac{5L}{6}$) using TemperaturSensor blocks. Additionally, we are interested in the amount of energy which is transferred to the rod via its left end. We determine this amount of energy by using a HeatFlowSensor, whose output is integrated (Integrator) and scaled (by factor 1e-4; just for nicer result plot).

Now, lets have a look at the simulation results shown in fig. 2.22. In the top diagram there are plots of the temperature at the left rod end (yellow curve) and at locations $\frac{L}{6}$ (red curve), $\frac{3L}{6}$ (green curve), and $\frac{5L}{6}$ (black curve) on the rod. Locations closer to the left end are heated up faster than others. In steady state, all will have the same temperature of 200℃. This could be verified by a prolonged simulation period. In the bottom diagram, we have the heat flow (blue curve), i.e. the power which is transmitted to the rod at every instant of time, and the additional accumulated energy (green curve) in the rod due to its heating-up. At the end of the simulation period this energy amounts to approx. 24.3×10^4 (J).

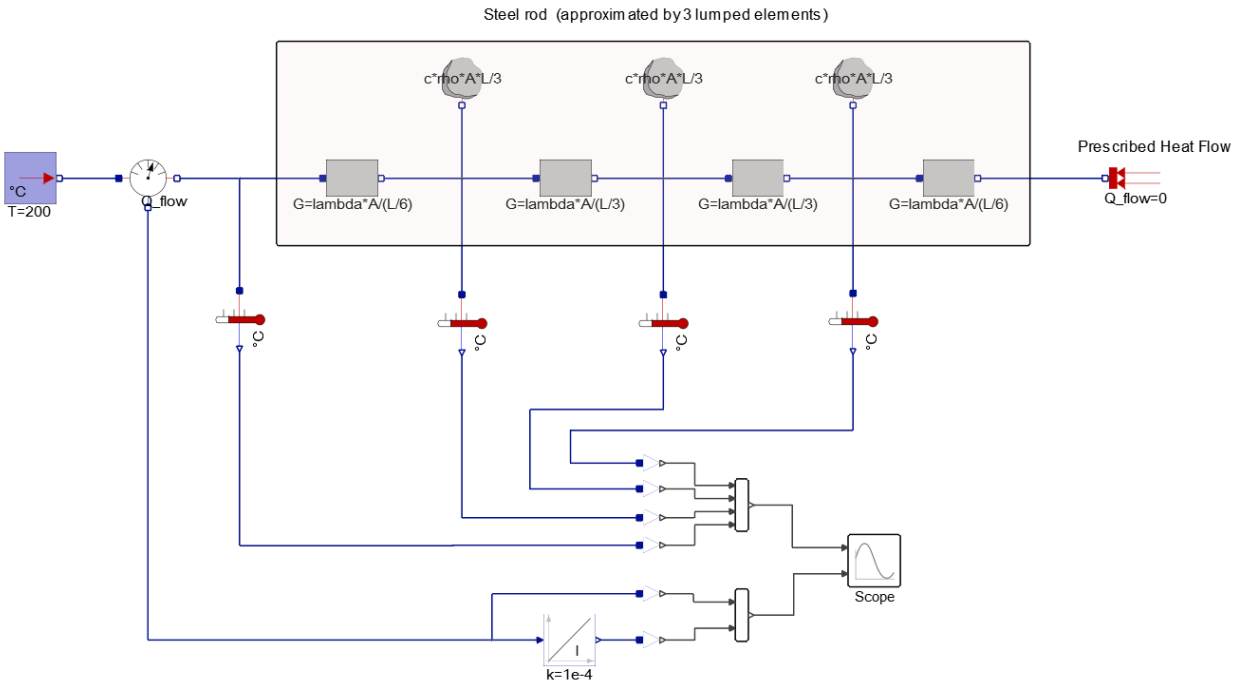


Figure 2.21: Insulated Steel Rod (`SteelRod`)

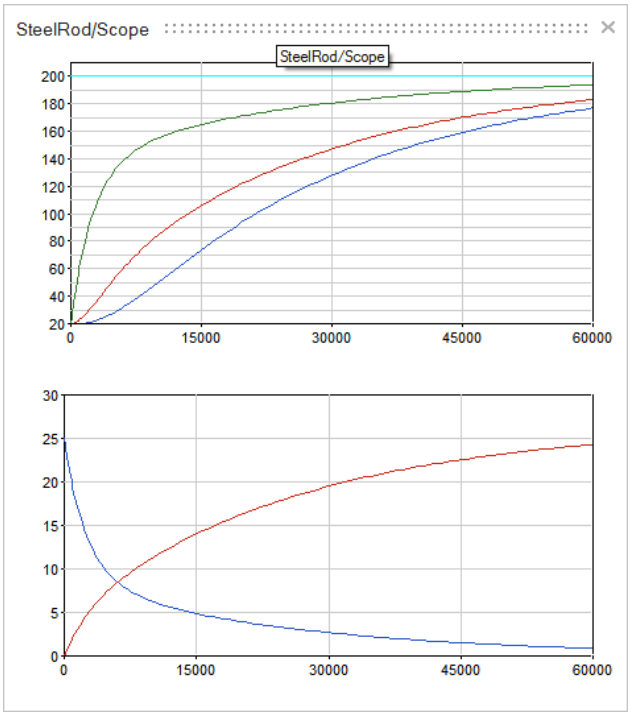


Figure 2.22: Temperatures, Power and Energy (`SteelRod`)

In this example, we have adequately modeled the heat transfer by thermal *conductance*. However, models incorporating heat transfer mechanisms like *radiation* and *convection* might be needed. In such cases, the blocks `BodyRadiation` and `Convection` can be used.

Chapter 3

Coselica Use Case 1 - Dynamic Extraction of a Pile

3.1 Scenario and Problem

A 23m long pile has been inserted 20m into the ground and shall be extracted by means of a *static* force S *and* a *dynamic* (harmonic oscillating) force V (cf. fig. 3.1). In order to achieve this, one has to overcome the shaft resistance force R of the pile. It is actually distributed over the whole shaft area which is in contact with the soil and depends on the relative movement of the shaft with respect to the soil, soil properties, and the pile geometry (size of shaft area).

In comparison to purely static extraction ($V = 0$) the use of a vibrator is advantageous, because the static force S needed here is usually significantly smaller. However, this implies the a priori choice of an appropriate vibrator.

3.1.1 Pile and Soil Properties

All relevant pile properties are given in tab. 3.1. Soil properties along the inserted length of the pile are known in terms of CPTParameters for a shaft resistance model (cf. sec. 3.2.1) have been derived based on these test results. Sometimes soil properties might be rather unknown and reasonable assumptions have to be made by geotechnical skilled persons.

Total Length L	23	m
Inserted Length	20	m
Young's Modulus E	2.1×10^{11}	$\frac{\mathrm{N}}{\mathrm{m}^2}$
Density ρ	7850	$\frac{\mathrm{kg}}{\mathrm{m}^3}$
Cross Section Area A	232×10^{-4}	m^2
Circumference	3.1	m

Table 3.1: Pile Properties

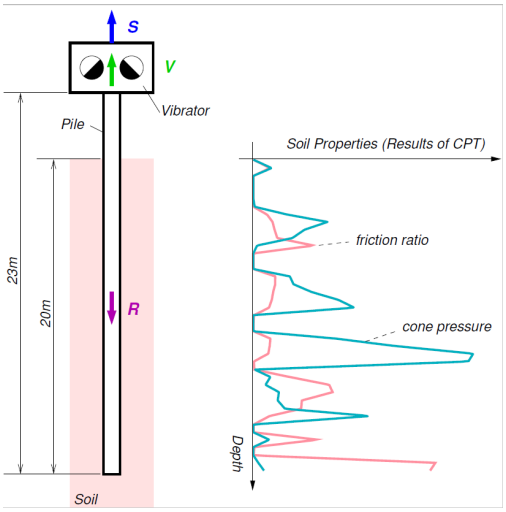


Figure 3.1: Dynamic Extraction of a Pile

Model	M_e [kgm]	f [Hz]	m_d [kg]
2310 VM	10	38.33	1450
2319 VM	19	38.33	2450
24 VM	24	38.33	4150
28 VM	28	38.33	3900
50 VM	50	33.33	4750

Table 3.2: Available Vibrators

3.1.2 Vibratory Pile Drivers

Inside a vibrator ex-centric counter-rotating masses are generating a vertical harmonic oscillating force

$$V = M_e\omega^2 = M_e(2\pi f)^2. \tag{3.1}$$

Where M_e is the ex-centric moment and f is the rotation frequency of the vibrator. Another important parameter is the dynamic (oscillating) mass m_d. We are looking for an appropriate vibrator from tab. 3.2.

3.1.3 Open Questions

In practice, one is interested in cost-efficient solutions and thus it is of high interest to know *a priori*:

- What is the smallest[1] vibrator, which is suitable for the job?

Furthermore, there is a kind of a mystic aura around vibratory pile driving and extraction. Even among civil engineering practitioners and experts there are continuing discussions regarding:

- Does the elasticity of the pile matter?

[1] with respect to the ex-centric moment M_e

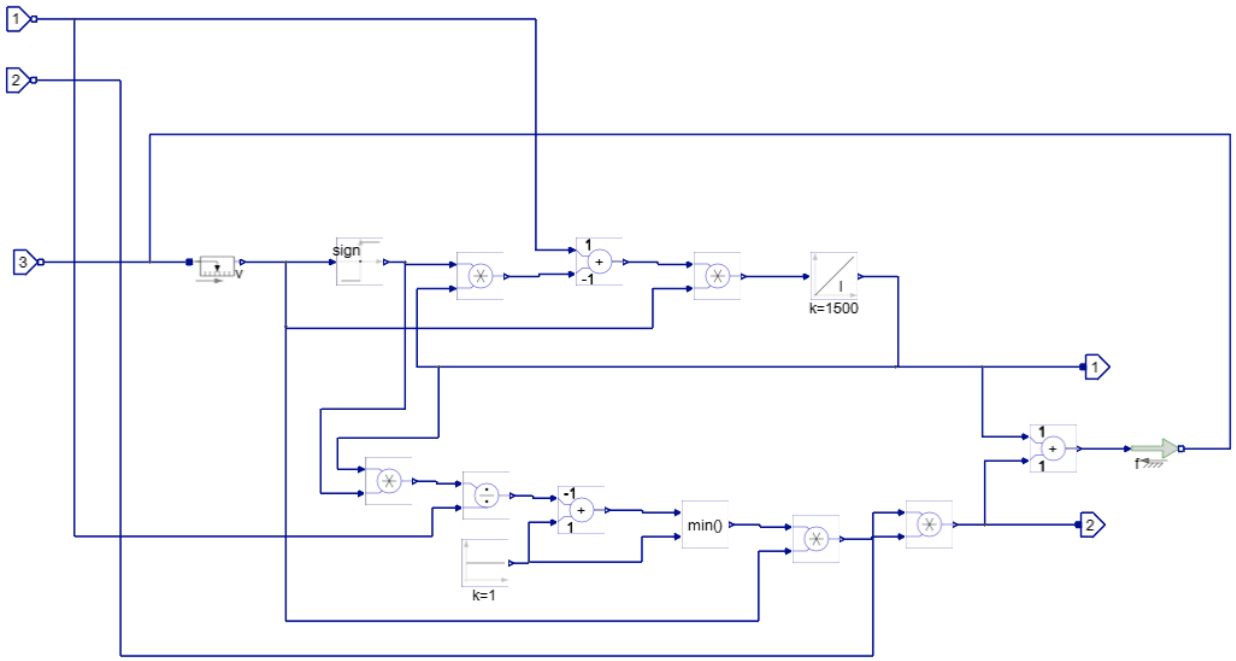


Figure 3.2: Shaft Resistance of Pile (`ShaftResistance`)

3.2 Modeling and Simulation

We are modeling and simulating the shaft resistance of the pile and the extraction process, assuming either a rigid or an elastic pile. For the sake of simplicity, we are taking gravity not into account and we are only interested, whether the fully (20m) inserted pile can be moved, thus we are not modeling the whole extraction process. It is pretty obvious, the ability to move the fully (20m) inserted pile allows the inference, that also every partial (<20m) inserted pile can be moved.

In a first step, we are investigating purely static pulling of the pile (cf. sec 3.2.2) in order to test our shaft resistance model (cf. sec. 3.2.1). Furthermore, this reveals the size of the static force S needed for extraction, albeit this is known beforehand.

In second step, we are investigating dynamic pulling of the pile (cf. sec. 3.2.3). We will see, whether we can pull the pile using one of the vibrators of tab. 3.2, assuming there is only a rather small static force $S = 10^5$N available.

Please note, in the following we have adopted in every model the same sign convention: a movement upwards is taken into account with negative sign. Furthermore, all movements are modeled as relative movements with respect to the position of the (initially) fully inserted pile. This implies that pulling (upwards) forces have negative sign.

3.2.1 Shaft Resistance of Pile

The shaft resistance R of the pile (cf. fig. 3.1) can be split into two parts: a *friction* force M and a *viscous damping* force D. In the following, we describe and use a simplified model, which has been

		Elastic Pile		
Parameter	Rigid Pile	Head	Center	Foot
D_{max} $[\frac{\text{Ns}}{\text{m}}]$	8.1×10^3	0.5×10^3	5.1×10^3	2.5×10^3
M_{max} [N]	4.1×10^6	0.01×10^6	2.59×10^6	1.50×10^6

Table 3.3: Shaft Resistance Model Parameters (for Rigid & Elastic Pile)

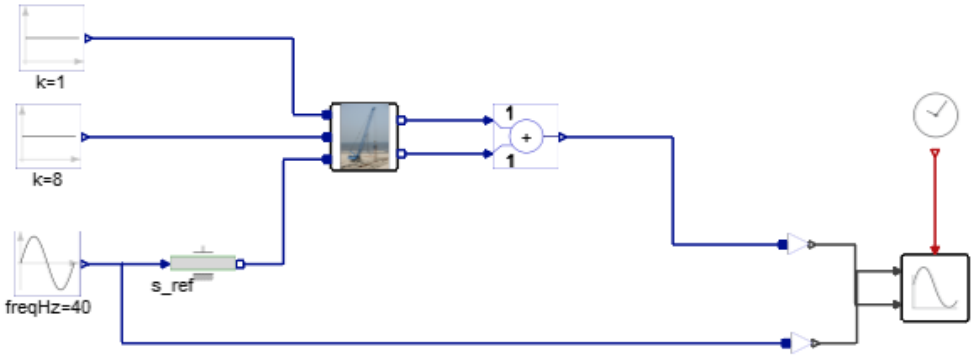


Figure 3.3: Testing of `ShaftResistance` super block (`ShaftResistance2`)

derived from literature[2].

The friction force M depends on the displacement of the pile after a reversal of motion and is given by

$$\frac{dM(t)}{dt} = \alpha \dot{z} \left(M_{max} - \text{sign}(\dot{z}) M(t) \right). \tag{3.2}$$

For sufficiently large displacements, a maximum friction force M_{max} will be mobilized. The parameters of eq. 3.2 are α (we will use a constant value of 1500) and M_{max}.

The viscous damping force D depends on the grade of mobilization of the friction force M and is given by

$$D(t) = D_{max} \dot{z} \min \left(1, 1 - \text{sign}(\dot{z}) \frac{M(t)}{M_{max}} \right). \tag{3.3}$$

Initially, after a reversal of the pile motion we have an effective maximum damping constant D_{max}. After a short period the mobilization of the friction force takes place and the effective damping constant becomes less than D_{max} and may be later for a full mobilization of friction ($|M| \approx M_{max}$) there will be no viscous damping at all. The parameters of eq. 3.3 are M_{max} and D_{max}.

For the described shaft resistance model we need to know the parameters M_{max} and D_{max}. They are depending on the soil properties (cf. sec. 3.1.1). Explicit values for them are given in tab. 3.3, they are used later for modeling the shaft resistance of a rigid and an elastic pile (cf. sec. 3.2.2 and sec. 3.2.3). The very details of their derivation are not within the scope of this document.

The above described model, i.e. eqs. 3.2 & 3.3, has been implemented as shown in fig. 3.2 as a super block. Herein, the parameters M_{max} and D_{max} have to be provided via the implicit input ports `Mmax` and `Dmax`. Port `z` is a translational *mechanical flange*, on which the shaft resistance `Force` (i.e. the

[2] GUILLERMO DIERSSEN: *Ein bodenmechanisches Modell zur Beschreibung des Vibrationsrammens in körnigen Böden*, Dissertation, University of Karlsruhe, Germany, 1994.

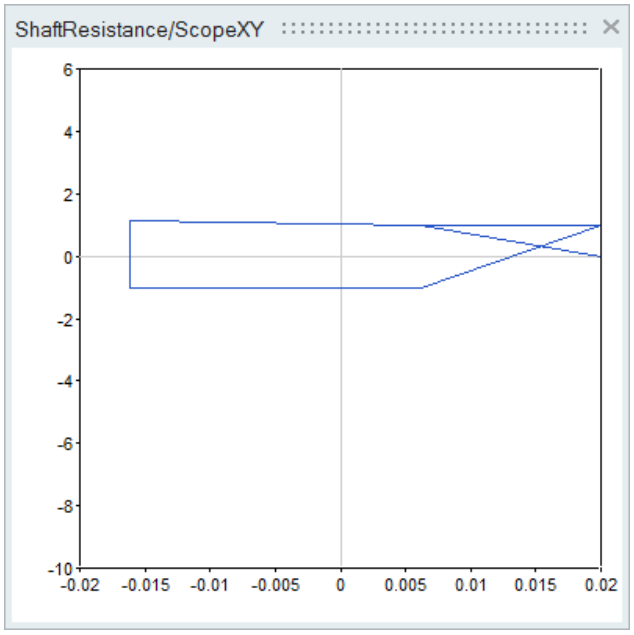


Figure 3.4: Shaft Resistance Force vs. Pile Movement (`ShaftResistance2`)

force $R = M + D$) is acting. It will later be connected with the pile, thus the `SpeedSensor` is measuring the speed $\dot{z}$ of the pile. Based on this measurement we are calculating M and D, according to eqs. 3.2 & 3.3, and exert a resistance `Force` $M + D$ on the pile (via port `z`).

The right-hand sides of eqs. 3.2 & 3.3 are implemented rather straightforward using a number of simple mathematical blocks (e.g. `Add`, `Product`, `Division`, `Sign`, ...). Eq. 3.2 is an ODE, thus we have to integrate its right-hand side via an `Integrator` block in order to get M. For convenience, we have embedded here the factor α (=1500) by choosing an integrator gain `k`=1500. In contrast, Eq. 3.3 is just an algebraic equation with respect to $\dot{z}$ and M. Please note, that `Add` blocks take two parameters `k1` and `k2`, and can be used to calculate arbitrary linear combinations $k_1 u_1 + k_2 u_2$ of their inputs u_1 and u_2.

The output ports `M` and `D` are not crucial, but might be handy for illustrating and *testing purposes* as shown fig. 3.3. For our test we are using just some simple parameter values (M_{max}=1 and D_{max}=8) for the shaft resistance and we are prescribing a oscillating pile `Movement (+-2mm)`. A `Sine` block generates a sinusoidal signal with frequency 40Hz, an amplitude of 0.002 (2mm) and a phase of $\frac{\pi}{2}$. Because of the phase $\frac{\pi}{2}$ this signal is actually a cosine signal, thus our prescribed movement will start at position +2mm. Finally, this signal can be used to prescribe the position `s_ref` (via a `Position0` block) of a translational *mechanical flange*, i.e. the position of the pile. For two movement cycles, the shaft resistance force $M + D$ versus the pile position z, is shown in fig. 3.4. The plot exhibits, that our shaft resistance model works as expected.

3.2.2 Static Pulling

We are exerting a slowly increasing (quasi-static) `Pulling Force` on the pile and measuring the pile movement. We expect, that the pile should start moving when this force becomes $\geq M_{max}$ (cf. tab. 3.3). Corresponding models and results for a rigid and an elastic pile are described in the following.

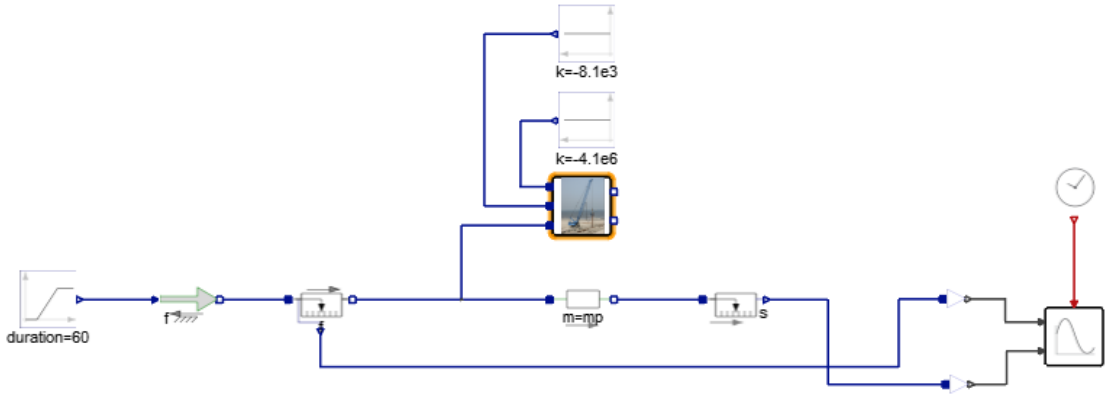


Figure 3.5: Static Pulling of a Rigid Pile (`StaticPullingRigidPile`)

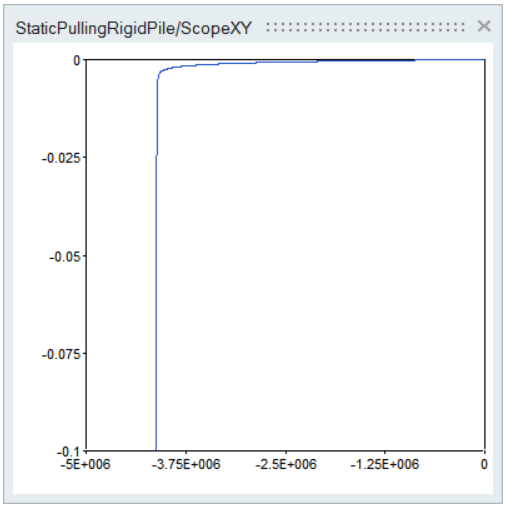


Figure 3.6: Simulation Results (`StaticPullingRigidPile`)

Rigid Pile

The `Pile` is modeled here as rigid body with mass `mp` (via a `Mass` block) as shown in fig. 3.5. There are two forces acting on the pile: the `Pulling Force 0...4.2MN` and the `Shaft Resistance` (cf. sec. 3.2.1). The `Pulling Force 0...4.2MN` signal is generated by a `Ramp` block (parameters `height`=-4.2e6, `duration`=60) and prescribed as force acting on a *mechanical flange* via a `Force0` block.

The maximum friction force $M_{max} = 4.1 \times 10^6 \mathrm{N}$ and the maximum damping force $D_{max} = 8.1 \times 10^3 \mathrm{N}$ (cf. tab. 3.3) are provided by two `Constant` blocks. Furthermore, we are measuring the force `f` acting on the `Pile` and its position `s`.

Simulation results, i.e. pulling force versus pile movement (upwards direction is negative), are shown in fig. 3.6. As expected, a significant upwards movement of the pile happens when the pulling force becomes $\geq M_{max}$.

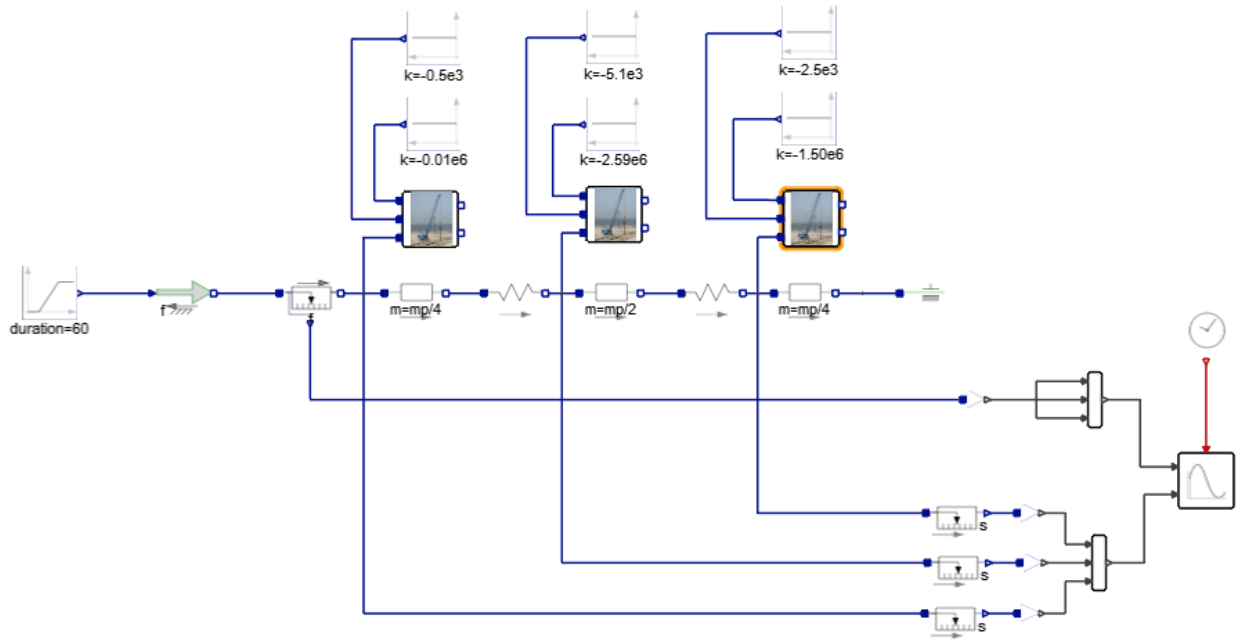


Figure 3.7: Static Pulling of an Elastic Pile (`StaticPullingElasticPile`)

Elastic Pile

The pile is modeled here approximately as elastic body by three lumped masses `mp/4`, `mp/2`, and `mp/4` (`Head`, `Center`, and `Foot`). They resemble three pile elements of length $\frac{L}{4}, \frac{L}{2}$, and $\frac{L}{4}$ and are connect via springs as shown in fig. 3.7. A different `Shaft Resistance` is acting on each of them. The right flange of the Foot (very lower end of the pile; tip of the pile) is connected to a `Free` block, because we are assuming there is no interaction between pile tip and soil.

The needed parameters D_{max} and M_{max} for each pile element (`Head`, `Center`, and `Foot`) are given in tab. 3.3. The elasticity $\frac{AE}{L}$ of the pile is modeled by the two springs in series, where each of them has a spring constant $2\frac{EA}{L}$. The relative movement of each pile element (`Head`, `Center`, and `Foot`) is measured.

Please note, that all component lengths (parameter `L` of all `Mass` blocks and parameter `s_rel0` of all `Spring` blocks) are set to zero, this is just fine, because, within the scope of this use case, we are only interested in relative positions of the pile elements with respect to their initial positions (for convenience set to zero as well).

Simulation results, i.e. pulling force versus pile movements (`Head`, `Center`, and `Foot`), are shown in fig. 3.8. As expected, a significant upwards movement of the whole pile happens when the pulling force becomes $\geq M_{max,\texttt{Head}} + M_{max,\texttt{Center}} + M_{max,\texttt{Foot}} = 4.1 \times 10^6$. Smaller pulling forces are just stretching but not significantly moving the pile as a whole.

3.2.3 Dynamic Pulling

We are replacing the quasi-static, really high, `Pulling Force 0...4.2MN` in the models of the previous section (cf. fig. 3.5 and fig. 3.7) by a rather small `Static Force` (10^5N are available) *and*

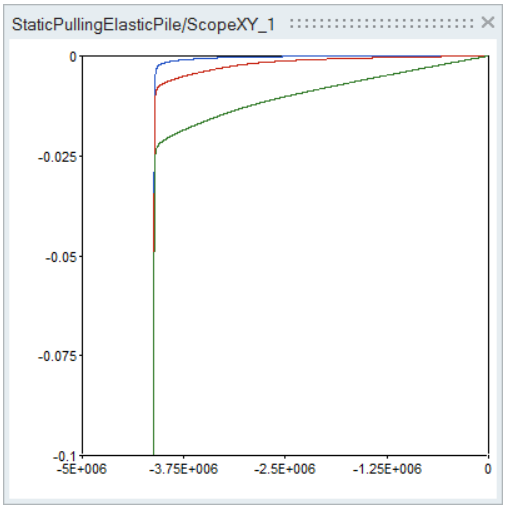


Figure 3.8: Simulation Results (`StaticPullingElasticPile`)

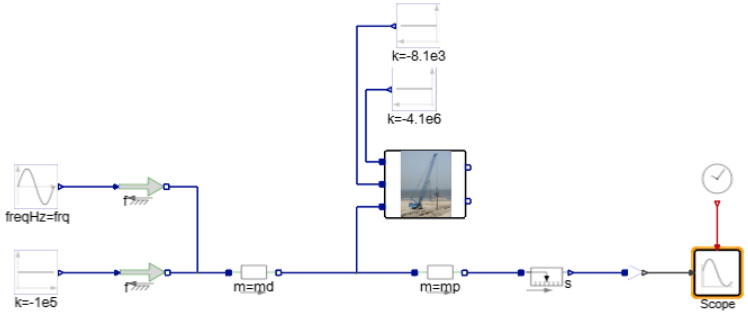


Figure 3.9: Dynamic Pulling of a Rigid Pile (`DynamicPullingRigidPile`)

a `Dynamic Force`, which is physically generated inside a `Vibrator` of mass `md` (cf. sec. 3.1.2) as shown in fig. 3.9 and fig. 3.10. The `Vibrator` is rigidly connected (in the real world using hydraulic clamps) to the `Pile` head. The `Dynamic Force` in the models is prescribed using a `Sine` block with parameters `amplitude`=V and `freqHz`=f (cf. eq. 3.1 and tab. 3.2). Please note, that the modeling of the rigid and the elastic pile remains the very same as before in sec. 3.2.2.

We are running a simulation for each of the available vibrators (cf. tab. 3.2) for a small period of time (2 seconds). If the pile moves significantly within this simulation period, then the corresponding vibrator might be regarded as suitable for the pile extraction job. Please note, that we use here a much smaller static pulling force of just 10^5N in comparison to 4.1×10^6N in the above case of purely static pulling (cf. sec. 3.2.2). Thus, an appropriate vibrator would made it possible to pull the pile using a pulling force of just 10^5N (apart from the gravity forces due to m_d and m_p).

Simulation results for dynamic pulling of a rigid and an elastic pile are described in the following.

Rigid Pile

The `Pile` is modeled as rigid body and connected to the `Vibrator` which generates a `Dynamic Force` as shown in fig. 3.9.

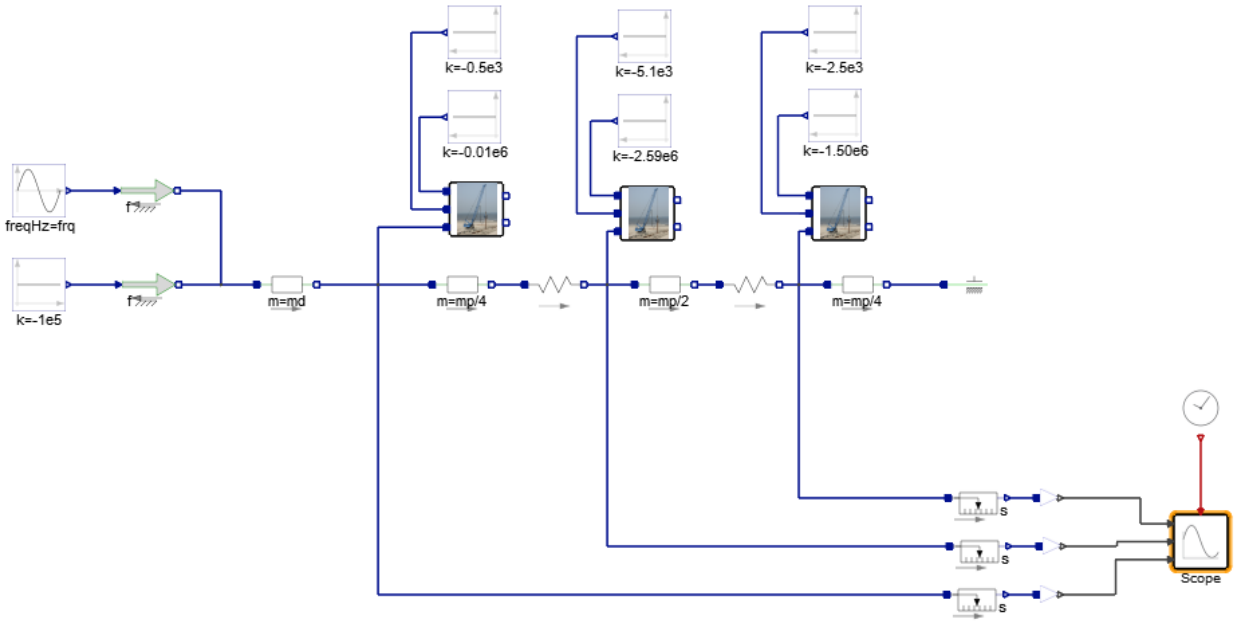


Figure 3.10: Dynamic Pulling of an Elastic Pile (`DynamicPullingElasticPile`)

Simulation results, i.e. pile movement versus time, for each available vibrator (cf. tab. 3.2) are shown in fig. 3.11. It seems, that none of the vibrators is capable to make the pile move significantly, not even the biggest one (*50 VM*).

These results are clearly unexpected and in contradiction to our experiences from practice! But, they are giving us a strong motivation for treating the pile as a somehow elastic body.

Elastic Pile

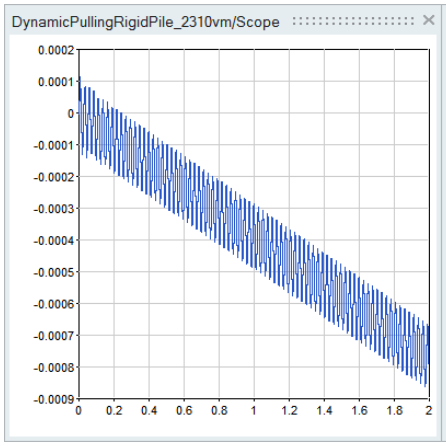
The pile is modeled as elastic body and the relative pile movement is measured at three locations (`Head`, `Center`, and `Foot`) as shown in 3.10. Due to the oscillating `Dynamic Force` the pile as a whole will experience stretch and compression at the same time, i.e. the relative movements of `Head`, `Center`, and `Foot` are in general differing.

Simulation results, i.e. the movements of the pile head (red), center (green), and foot (blue) versus time, for each available vibrator (cf. tab. 3.2) are shown in fig. 3.12. The pile as a whole is not moving significantly using the vibrators *2310 VM* and *2319 VM* (cf. fig. 3.12a and 3.12b). But with the bigger machines *24 VM*, *28 VM*, and *50 VM* the pile as a whole is moving significantly upwards (cf. fig. 3.12c, 3.12d, and fig. 3.12e). So what is the difference between success and failure? A close look reveals, that the *2310 VM* and *2319 VM* are not powerful enough to make the pile foot oscillating, such that the averaged (over time) friction force becomes small, whereas the others make the pile foot clearly oscillating. All in all, in practice a good (cost-efficient) choice for the pile extraction job seems to be the *24 VM*!

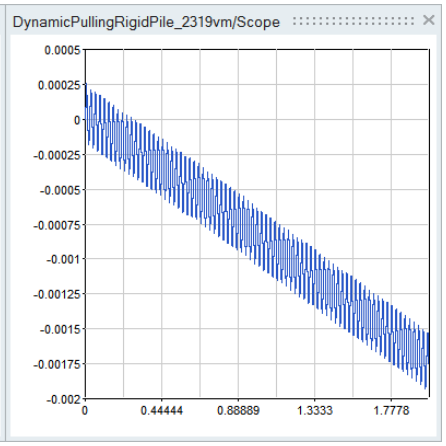
3.3 Major Results

Based on the above carried out simulation study, we can summarize the following results:

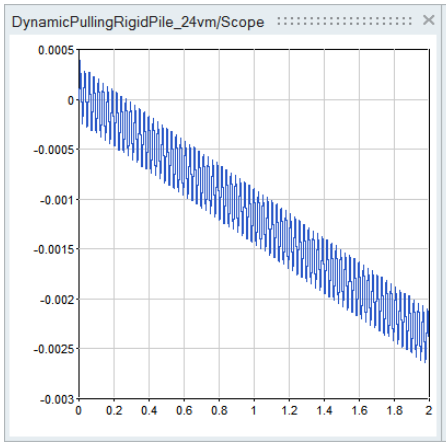
- Vibrator *24 VM* seems to be a good choice for the described pile extraction job.
- If a vibrator is used for pile extraction, then pile elasticity does matter. This result is twofold:
  - If pile elasticity is present, then it should be taken into account.
  - It easier to pull an elastic pile than a rigid pile.



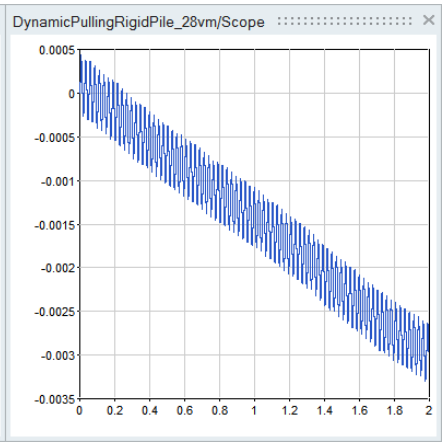
(a) Vibrator *2310 VM*



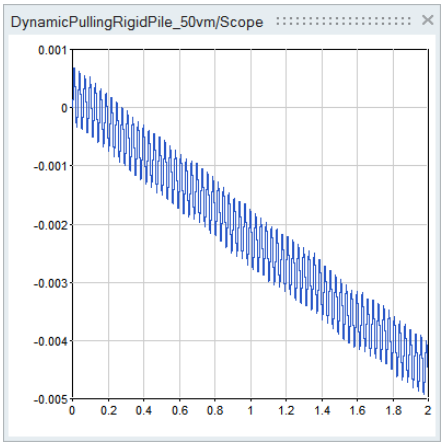
(b) Vibrator *2319 VM*



(c) Vibrator *24 VM*

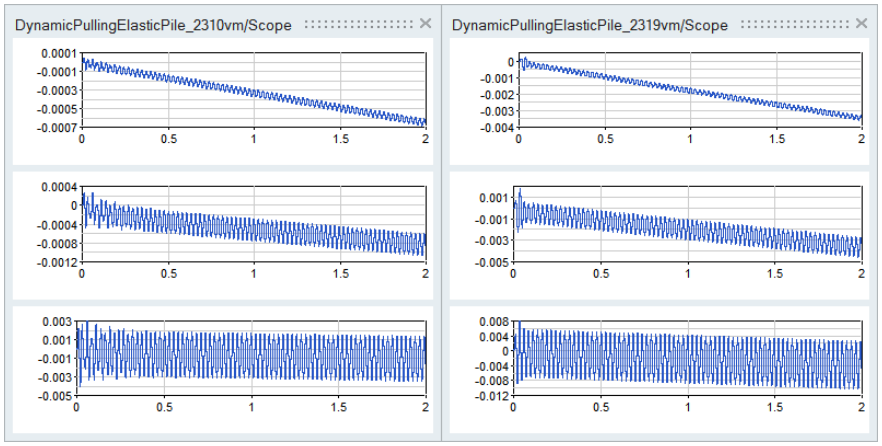


(d) Vibrator *28 VM*

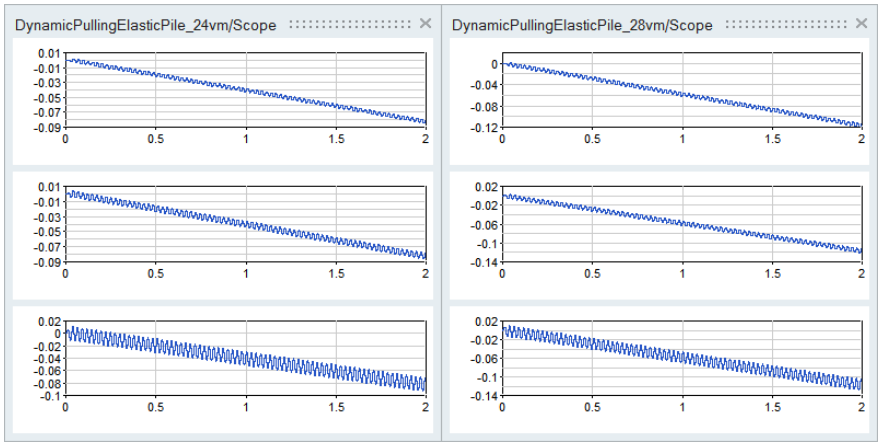


(e) Vibrator *50 VM*

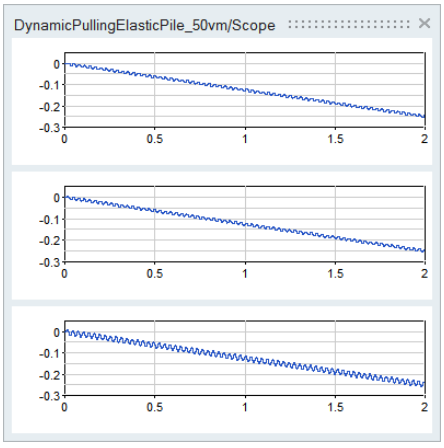
Figure 3.11: Simulation Results (`DynamicPullingRigidPile`)



(a) Vibrator *2310 VM* (b) Vibrator *2319 VM*



(c) Vibrator *24 VM* (d) Vibrator *28 VM*



(e) Vibrator *50 VM*

Figure 3.12: Simulation Results (`DynamicPullingElasticPile`)

Chapter 4

Coselica Use Case 2 - Controlled Heating-Up of a Small Reflow Oven

4.1 Scenario and Problem

Surface mounted devices[1] (SMDs) have to be mounted on the surface of *printed circuit boards* (PCBs) as shown on the titlepage. This can be done by *reflow soldering*[2], where the electronic components are placed on the contact pads, coated with solder paste, of a PCB. This assembly is heated-up in a *reflow oven*, such that the solder is melting and thus creating permanent electrical connections. The *heating-up* process should follow a prescribed *temperature profile* in order to avoid damaging the electronical components.

4.1.1 Reflow Soldering (Temperature) Profile

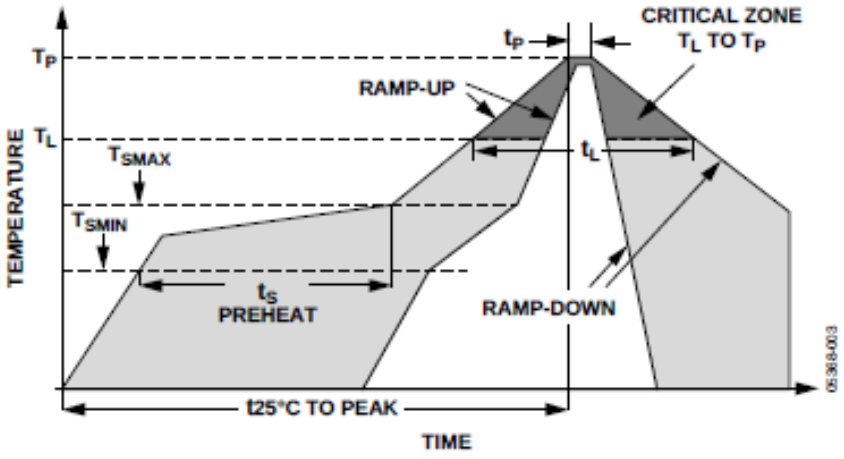


Figure 4.1: Recommended Soldering Profile (Source: Datasheet ADXL78)

[1] http://en.wikipedia.org/wiki/Surface-mount_technology
[2] http://en.wikipedia.org/wiki/Reflow_soldering

Profile Feature	Sn63/Pb37	Pb-Free
AVERAGE RAMP RATE (T_L TO T_P)	3°C/s max	3°C/s max
PREHEAT		
Minimum Temperature (T_{SMIN})	100°C	150°C
Maximum Temperature (T_{SMAX})	150°C	200°C
TIME (T_{SMIN} TO T_{SMAX}), t_S	60 s – 120 s	60 s – 150 s
T_{SMAX} TO T_L		
Ramp-Up Rate	3°C/s	3°C/s
TIME MAINTAINED ABOVE LIQUIDOUS (T_L)		
Liquidous Temperature (T_L)	183°C	217°C
Time (t_L)	60 s – 150 s	60 s – 150 s
PEAK TEMPERATURE (T_P)	240°C + 0°C/–5°C	260°C + 0°C/–5°C
TIME WITHIN 5°C OF ACTUAL PEAK TEMPERATURE (t_P)	10 s – 30 s	20 s – 40 s
RAMP-DOWN RATE	6°C/s max	6°C/s max
TIME 25°C TO PEAK TEMPERATURE	6 min max	8 min max

Table 4.1: Recommended Soldering Profile (Source: Datasheet ADXL78)

Manufacturers of electronic components are recommending soldering profiles to be used when mounting their components. A typical example is depicted in fig. 4.1. The peak temperature `Tp` is reached after three consecutive phases: *heating up to* `PREHEAT`, `PREHEAT`, and `RAMP-UP`. The peak temperature has to be reached in a way, such that certain boundary conditions, given in tab. 4.1, are fulfilled. Please note, we intend soldering PCBs compliant to the RoHS[3], thus only the `Pb-Free` case is of interest for us. The cooling-down phase is much less critical (cf. bottom of tab. 4.1) and not subject of this use case.

4.1.2 Open Questions

A small electrical oven (AC 230V@50Hz, Power 1500W) shall be used for reflow soldering of PCBs. In order to avoid time-consuming and costly blackbox experiments, we would like to investigate by simulation the following:

- Is it feasible to generate a compliant temperature profile, due to the rather limited power of the oven?
- Can we control the oven using conventional PID control, such that a satisfactory heating-up profile is generated?

4.2 Modeling and Simulation

We are modeling and simulating an automatic controlled electrical oven. The oven is described by a physical multi-doman model, i.e. it has an interconnected electrical and thermodynamical part. The heating-up phase of the oven will be set under automatic control, such that the oven temperature follows a prescribed temperature profile (reflow soldering profile).

In a first step, we are using a simple relay controller to check, whether the oven is powerful enough to follow a temperature profile suitable for reflow soldering.

In a second step, we are switching to PID control in order to improve control performance. Having real application in mind, we are adding an “Anti-Windup” feature and testing the controller with respect to

[3]http://en.wikipedia.org/wiki/Restriction_of_Hazardous_Substances_Directive

disturbances like varying ambiant temperature and noisy (temperature) measurements.

4.2.1 Electrical Oven

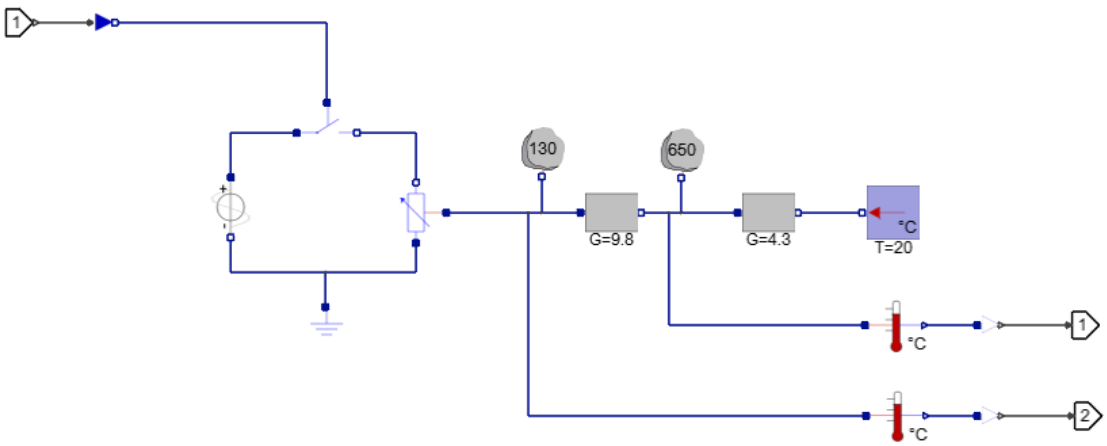


Figure 4.2: Model of Electrical Oven (`Reflow Oven`)

The electrical oven is modeled as a super block, whose content is shown in fig. 4.2. Both electrical and thermodynamical behaviour is modeled. In the electrical domain a heating resistor resembles the `Heating Element` of the oven. It is powered by an AC voltage source `220V @ 50Hz` and converts electrical into thermal energy (red port). Thus, it is link between the electrical and the thermodynamical modeling domain. The heating can be switched *on* and *off* by means of a `Switch`, which is triggered by a real signal (`Power`). The thermodynamical part of the model considers two lumped heating capacities `Heating Element` and `Oven` and thermal `Conductance` between them and the environment (`Ambient Temperature`). We are measuring the `Oven Temperature` and the `Heating Element Temperature` and providing them via output ports `T_Oven` and `T_Heating`. In this model the heating capcaties of PCBs located inside the oven are considered to be comparatively small and thus neglected.

So far, we can switch the oven *on* (0% heating power) and *off* (100% heating power) only. We intend to use later a PID controller whose output will prescribe arbitrary values (0%...100%) for the heating power. Thus, we need a `PWM Converter` as shown in fig. 4.3, which translates a continuous `DutyCycle(%)` signal into an `On/Off` (0/1) signal which can be used to trigger the `Switch` (cf. fig. 4.2). The output `On/Off` is switched from 0 to 1 and vice versa based on a comparison of the `DutyCycle(%)` and a `Triangular Wave`.

A simple test of our oven model is shown in fig. 4.4. Here, we can prescribe an arbitrary constant `Heating Power` and observe the corresponding `On/Off` switching of the heating and the evolution of the `Oven Temperature`.

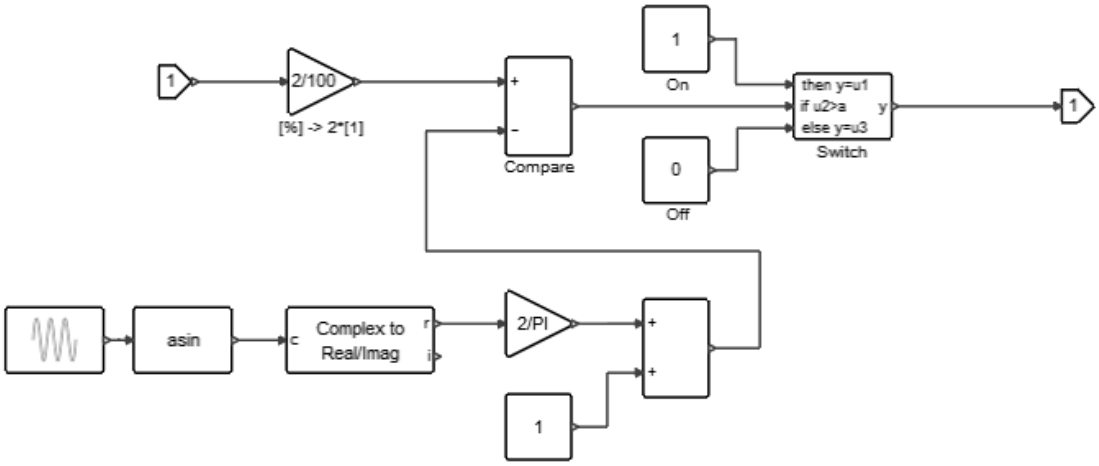


Figure 4.3: Conversion of Percentage to On/Off-Signal (`PWM Converter`)

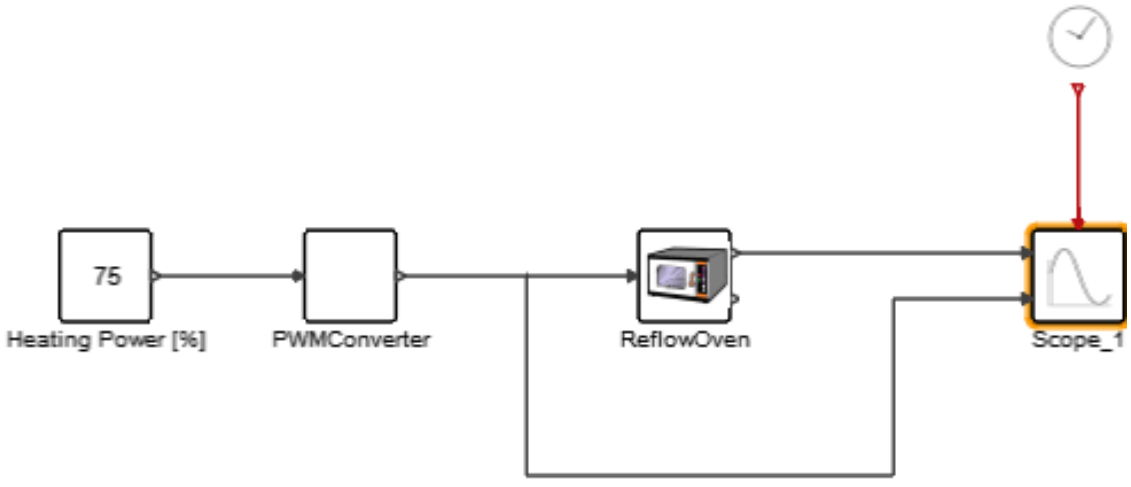


Figure 4.4: Test of Oven Model (`ReflowOvenCharacteristics`)

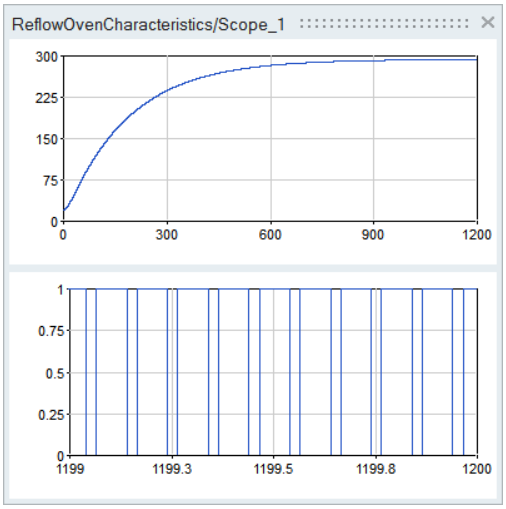


Figure 4.5: Test Results

Simulation results shown in fig. 4.5 meet our expectations. For a heating power of 75% the oven reaches a final – steady state – temperature of approx. 295°C (cf. fig. 4.5). The corresponding switching signal is shown in fig. 4.5 and reveals that within one period the heating is switched on in 75% of the time, which was expected. Please note, that these results (cf. fig. 4.5) might be compared with data of an experiment carried out with a real oven in order to validate our model.

4.2.2 Simple Relay Control

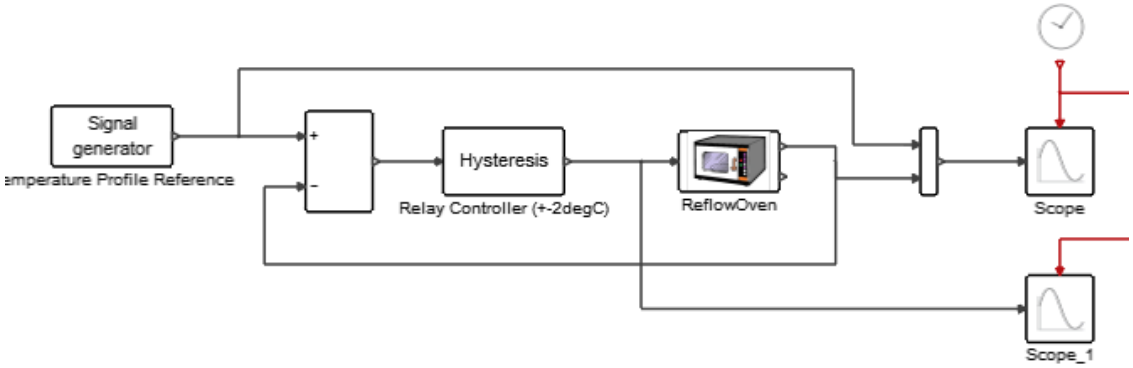
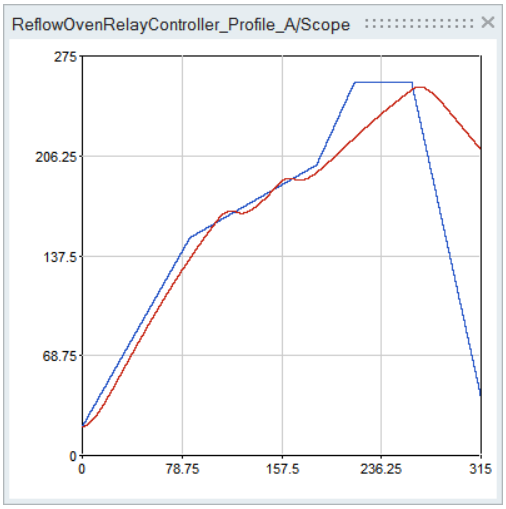


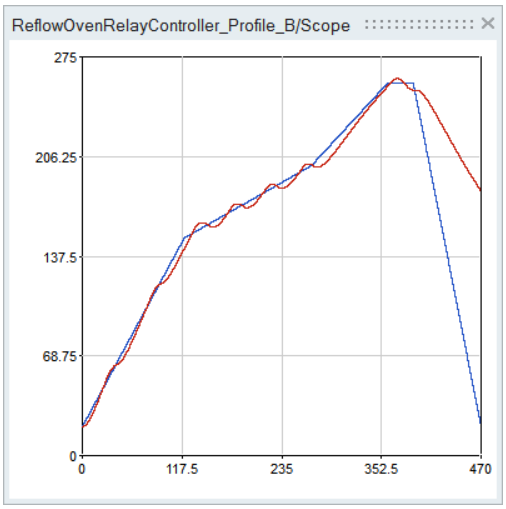
Figure 4.6: Reflow Oven with Relay Controller (`ReflowOvenRelayController`)

The oven is now *relay controlled* as shown in fig. 4.6. We do not need a PWM Converter here, because the output of the `Relay Controller (+-2degC)`, i.e. a hysteresis block, switches from 0 to 1 and vice versa, whenever the difference between actual (`T_Oven`) and desired (`Temperature Profile Reference`) temperature exceeds ± 2°C. A time-varying `Temperature Profile Reference` is implemented using a *lookup* block.

We have set up a temperature profile, called *A* (cf. blue curve in fig. 4.7a), which is compliant to the reflow soldering recommendations discussed above (cf. sec. 4.1.1). Is the oven powerful enough to



(a) Temperature Profile A



(b) Temperature Profile B

Figure 4.7: Relay-Controlled Oven Temperature

generate, at least approximately, that profile? Simulation reveals (cf. red curve in fig. 4.7a), that the oven is *not* powerful enough. Beginning at $t \approx 200$s the oven temperature is loosing track.

Are we forced to rule out our specific oven at this point? No, we might give it another try, using a more moderate – but still compliant (cf. sec. 4.1.1) – temperature profile, called *B* (cf. blue curve in fig. 4.7a). Simulation exhibits now, that the oven is capable of following the track during the whole heating-up phase. Thus, the oven seems to be suitable for reflow soldering purposes, as long as we stick to a temperature profile as shown in fig. 4.7b (blue). We are now moving on to PID control, in order to decrease the control error (difference between actual and desired temperature).

4.2.3 PID Control

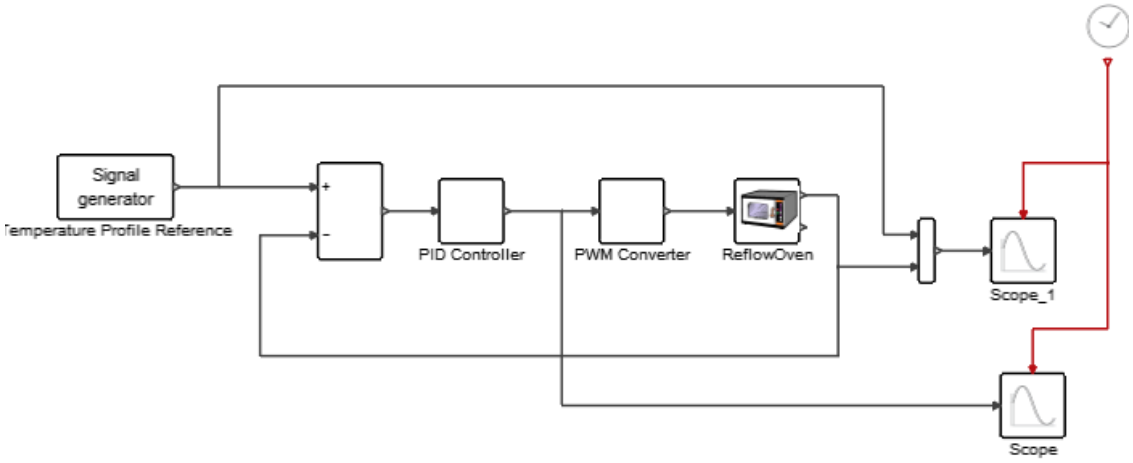


Figure 4.8: Reflow Oven with PID Controller (`ReflowOvenControllerPid`)

We have replaced the relay controller from above by a `PID Controller` in series with a `PWM Converter` (discussed above; cf. sec4.2.1) as shown in fig. 4.8. The controller takes the error e as input and gives a continuous output value u, which is interpreted a heating power in % to control the oven.

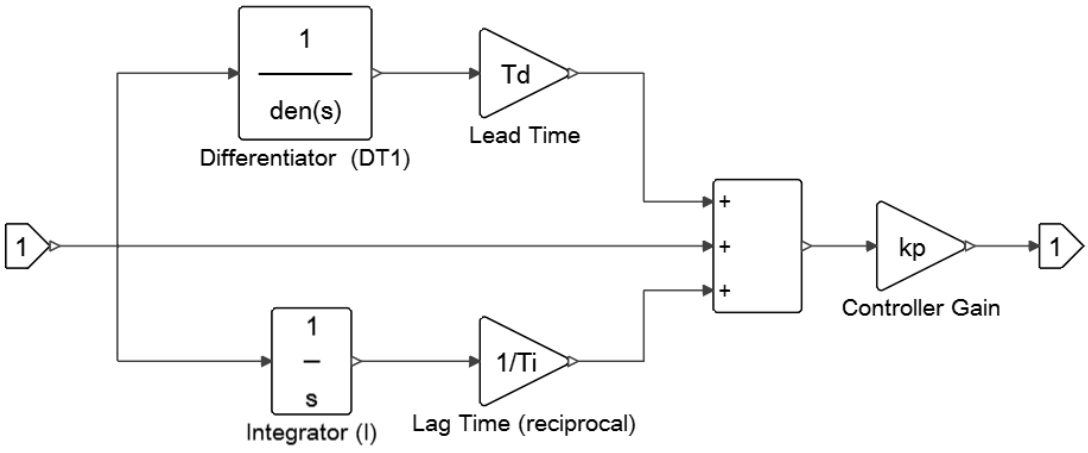


Figure 4.9: PID Controller (`PID Controller`)

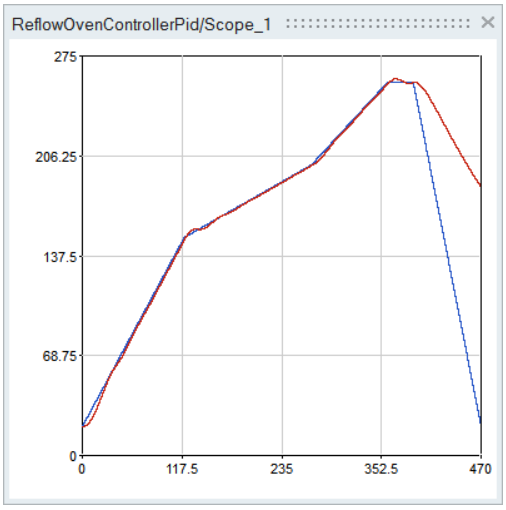
Inside the super block `PID Controller` the dynamic relation between e and u is implemented (cf. fig. 4.9). In the LAPLACE domain it can be written as

$$U(s) = k_p(1 + \frac{1}{T_i s} + \frac{T_d s}{\frac{Td}{Nd}s + 1})E(s). \tag{4.1}$$

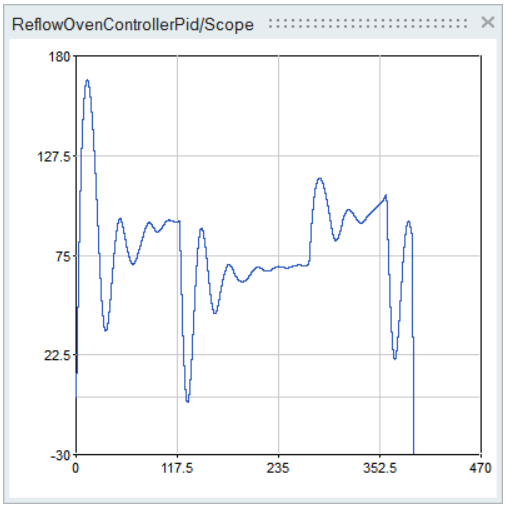
Please note, that instead of $T_d s$ we are using $\frac{T_d s}{\frac{T_d}{N_d}s+1}$ for the differentiating part of the control law in order to make the controller realizable, i.e. causal.

The controller parameters k_p, T_i, T_d and N_d have to be tuned somehow. In this use case this was done just heuristically and further improvements might be possible. Simulation results of the PID controlled heating-up of the oven are shown in fig. 4.10. The oven temperature (red) is following our prescribed temperature profile (blue) reasonably well during the heating-up phase.

Now, lets have a look at the corresponding controller output (cf. fig. 4.10b). It clearly exceeds the applicable limits 0% and 100% several times, e.g. at times $t \approx 20s$, $t \approx 130s$, and $t \approx 280s$. This may raise the question: Are our simulation results valid? Yes, they are! Because an implicit limitation to 0%. . .100% is performed within the `PWM Converter`. Nevertheless, in practise one has to guarantee, that saturation of the controller output doesn't persist. Otherwise, there will be no active feedback anymore and this may render the control loop useless or even worse may provoke instability. In the following section we are improving our controller with respect to that issue.



(a) Oven Temperature



(b) Controller Output (Heating Power [%])

Figure 4.10: PID Controlled Oven Temperature

Anti-Windup

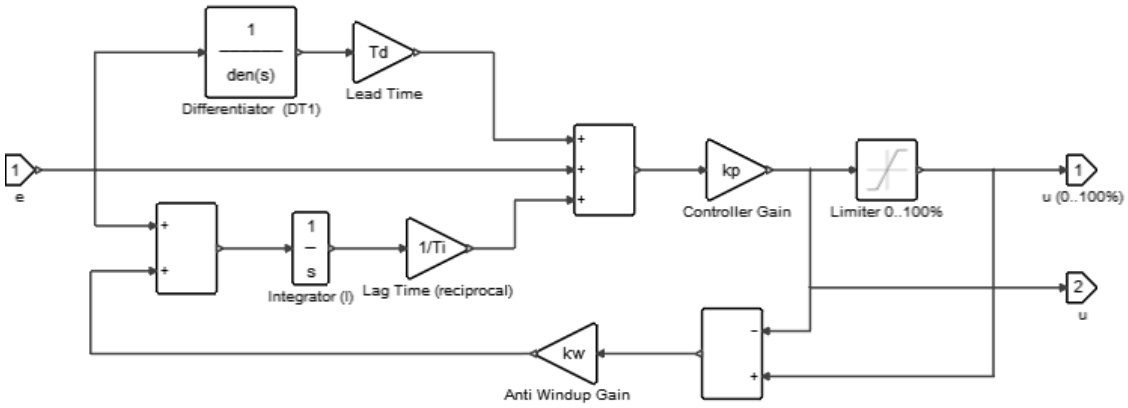


Figure 4.11: PID Controller with Anti-Windup (`PID Controller`)

We are adding a so-called “Anti-Windup” feature to our `PID controller` in order to avoid persistent saturation of the controller output. The improved controller is shown in fig. 4.11. The difference of limited (`u(0..10%)`) and unlimited output (`u`) is amplified (`Anti Windup Gain`) and additionally fed into the `Integrator`. In case of excess, this will automatically drive the output back within its limits. Please note, that just for observation purposes the controller has now an unlimited *and* a limited output.

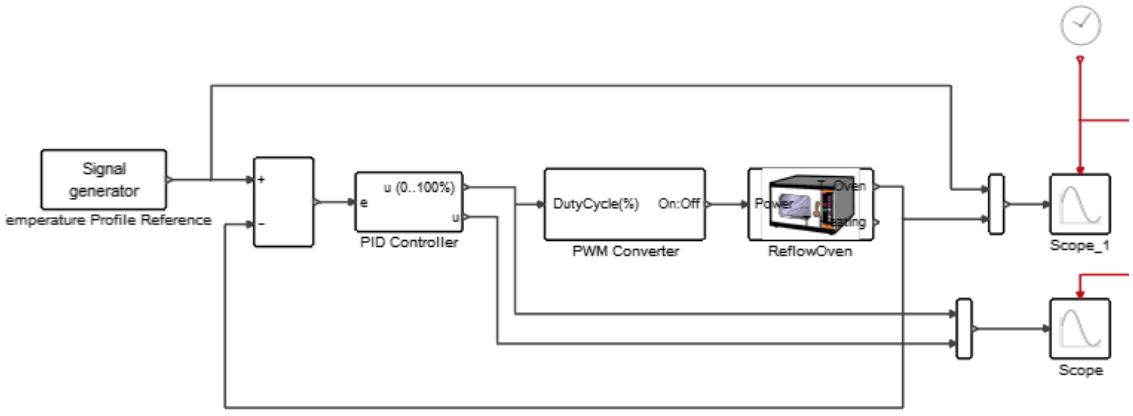
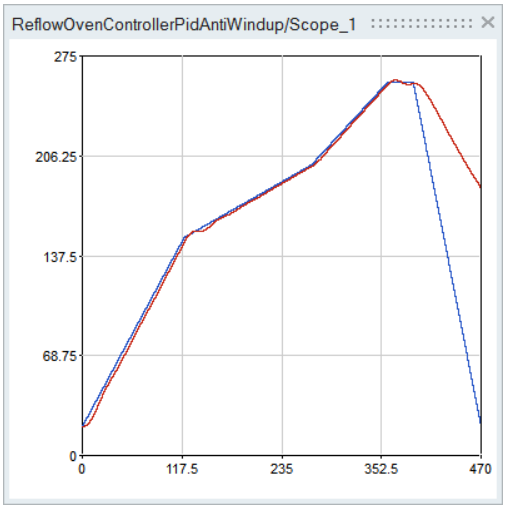


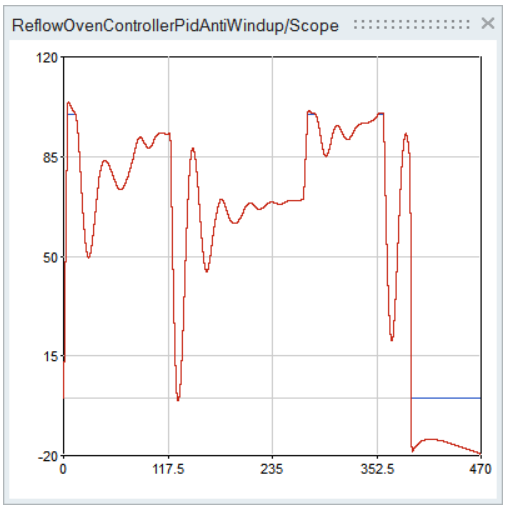
Figure 4.12: Reflow Oven with PID Anti-Windup Controller
(`ReflowOvenControllerPidAntiWindup`)

The PID controller with anti-windup (cf. fig. 4.12) was used and simulation results are shown in fig. 4.13. The tracking performance seems to be a bit degraded in comparison to the previous results (cf. fig. 4.10a), but it is still good enough. However, it is more interesting to look at the limited (blue) and unlimited (red) controller output shown in fig. 4.13b. The anti-windup feature is working as expected and its effect is clearly visible in comparison to fig. 4.10b.

At this point we may conclude, that a reasonably well working PID controller was found with respect to its tracking performance. In the real world the controller most likely has furthermore to cope disturbances



(a) Oven Temperature

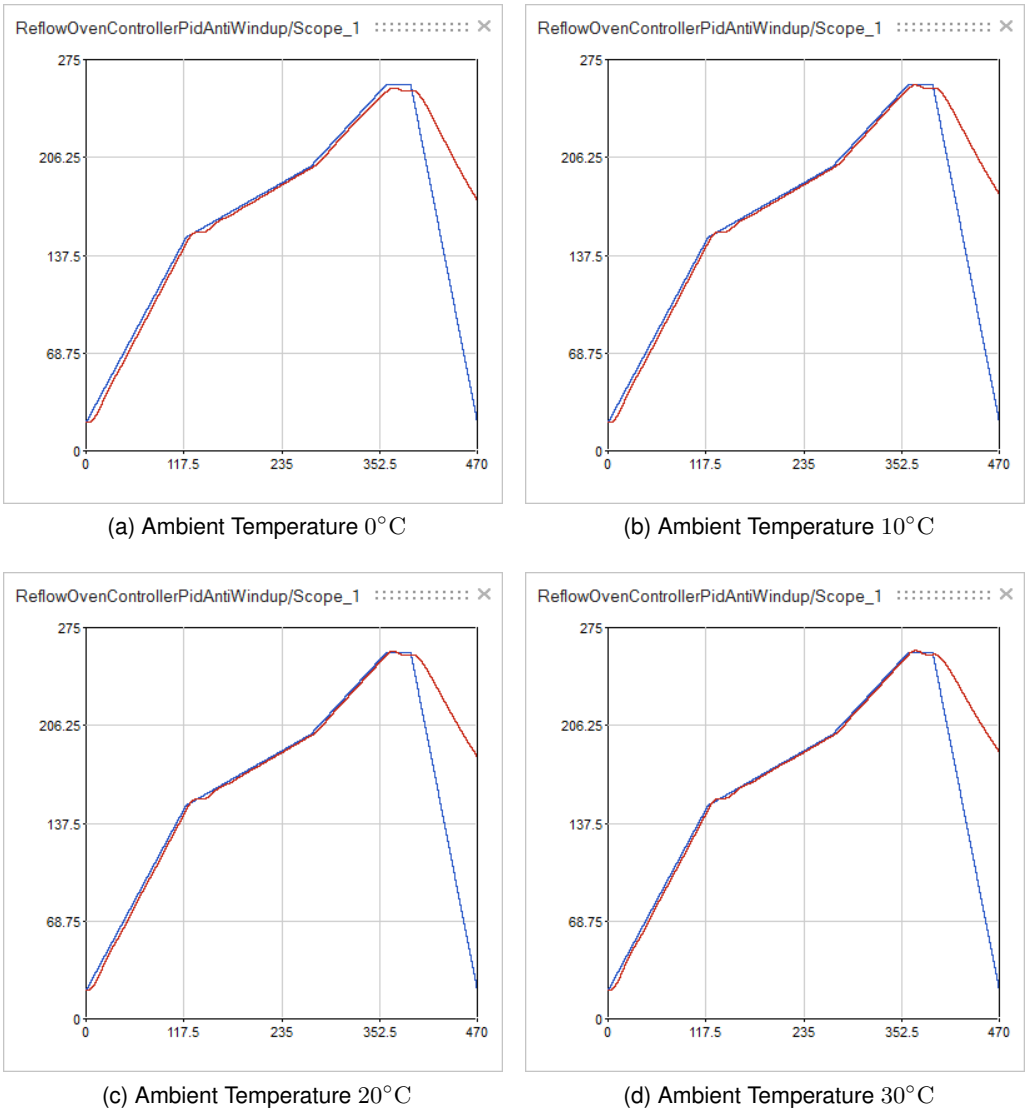


(b) Controller Output (Heating Power [%])

Figure 4.13: PID Anti-Windup Controlled Oven Temperature

like varying ambient conditions and measurement noise.

Varying Ambient Temperature



(a) Ambient Temperature $0°\mathrm{C}$ (b) Ambient Temperature $10°\mathrm{C}$

(c) Ambient Temperature $20°\mathrm{C}$ (d) Ambient Temperature $30°\mathrm{C}$

Figure 4.14: Oven Temperature at Different Ambient Temperatures

So far, we have assumed a constant `Ambient Temperature` (cf. fig. 4.2) of 20°C. We know that it somehow influences the thermal losses of the oven. Does the oven still perform reasonably well for ambient temperatures of $0°$C, $10°$, $20°$C and $30°$C? Simulation results for different ambient temperatures are shown in fig. 4.14. For $20°$C and $30°$C we get, as expected, satisfying results (cf. fig. 4.14c & d). For $0°$C and $10°$C, we can clearly see the negative influence of the increased thermal losses induced by the lower ambient temperature. A possible remedy in these cases could be improving the insulation of the oven or selecting a more powerful oven. Both aspects will not be pursued further in this use case.

Based on the above, we may assume that our oven is working well for a constant ambient temperature

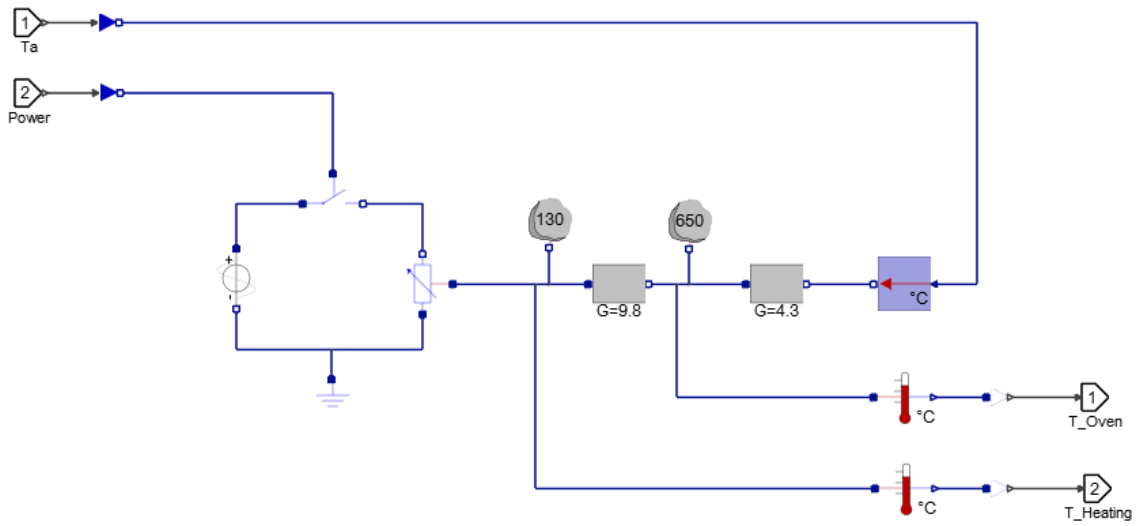


Figure 4.15: Reflow Oven with Variable Ambient Temperature (`Reflow Oven`)

between 15°C and 25°C. What happens, when the ambient temperature is not constant any more and rather fast varying? In order to investigate that, we have enhanced the oven model by an additional input port `T_a` (cf. fig. 4.15), which allows the prescription of an arbitrary time-varying ambient temperature. We use this model in the control loop (cf. fig. 4.16) and prescribe a sinusoidally, rather fast, varying ambient temperature of $20° \pm 5°$(`Ambient Temperature (+-5degC)`; cf. fig. 4.16 and 4.17).

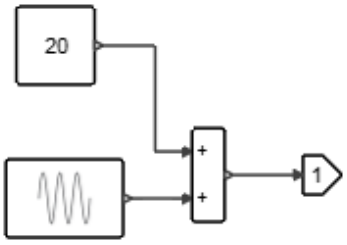


Figure 4.17: Ambient Temperature Signal (`Ambient Temperature (+-5degC)`)

Simulation results in fig. 4.18 are indicating, that the controller is able to compensate the disturbance of a rather fast varying ambient temperature (magenta curve) very well.

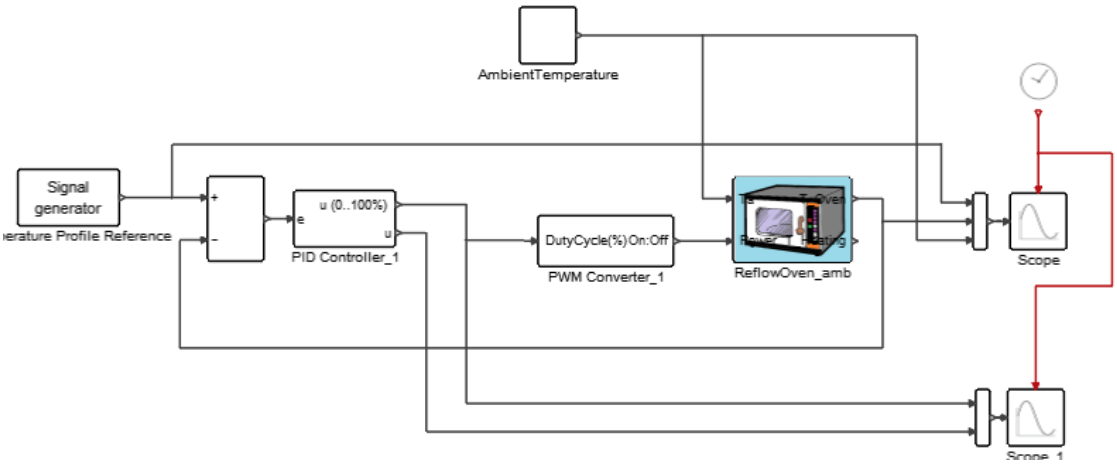


Figure 4.16: Reflow Oven Control with Varying Ambient Temperature (`ReflowOvenControllerPidAntiWindupVaryingAmbiance`)

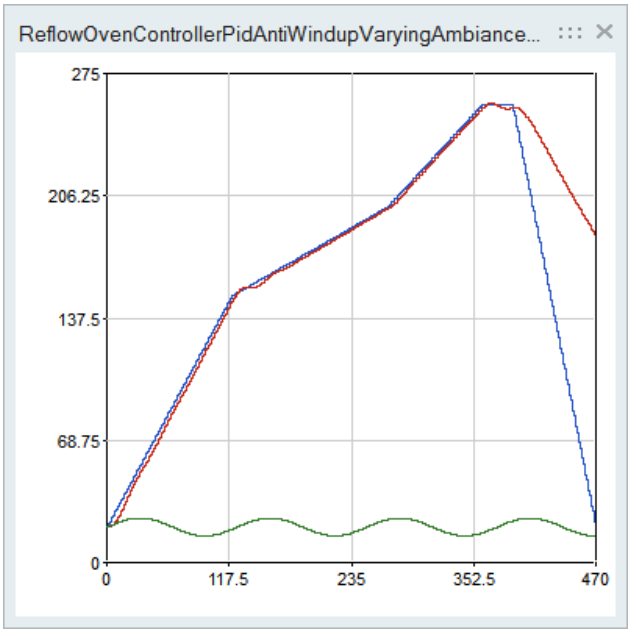


Figure 4.18: Oven Temperature and Ambient Temperature

Sensor Noise

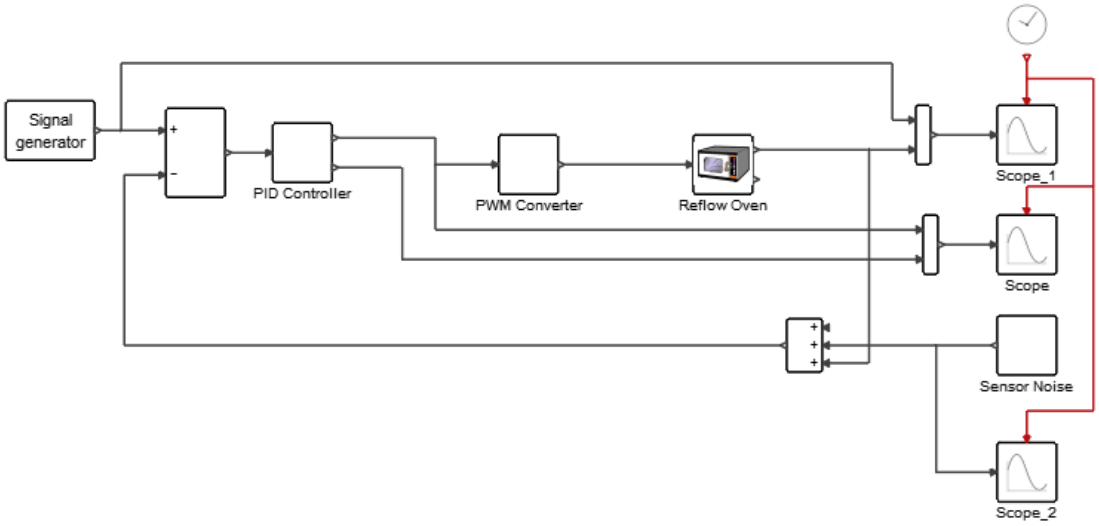


Figure 4.19: Reflow Oven Control with Sensor Noise
(`ReflowOvenControllerPidAntiWindupSensorNoise`)

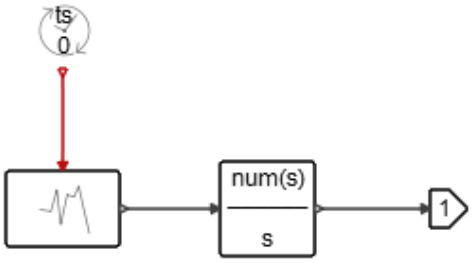


Figure 4.20: Generation of Coloured Sensor Noise

In this section, we will have a look at the impact of measurement noise on the performance of our control loop. We are assuming that our measurements of the oven temperature (`T_Oven`) are tainted with `Sensor Noise` as shown in fig. 4.19. The noise signal `n` is generated inside the super block `Sensor Noise` by applying a simple filter (`Coloured Noise`) to a zero-mean, Gaussian `White Noise`. This setup generates a coloured noise signal as shown in fig. 4.21.

Simulation reveals, that the introduced sensor noise has no visible impact on the oven temperature (cf. fig. 4.22a). Thus, the controller is good at compensating the sensor noise. Please note, the noise is propagated to the controller output, which is clearly visible in fig. 4.22b. But, this is not a problem, because the oven itself is a second order delay system (low pass) and suppresses the noise very well.

4.3 Major Results

Based on the above carried out simulation study, we can summarize the following results:

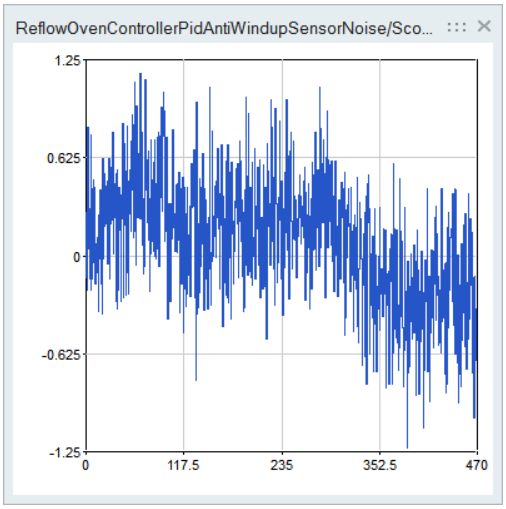
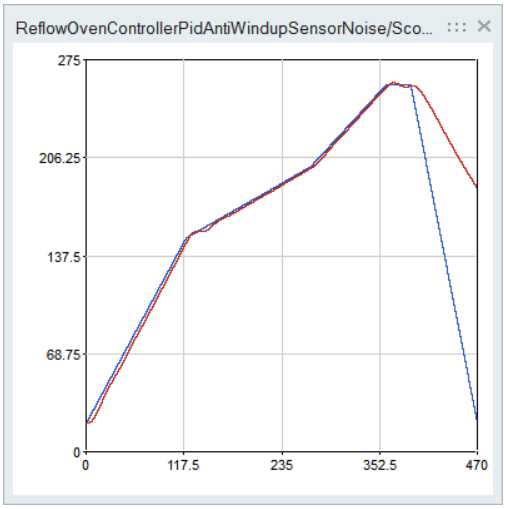
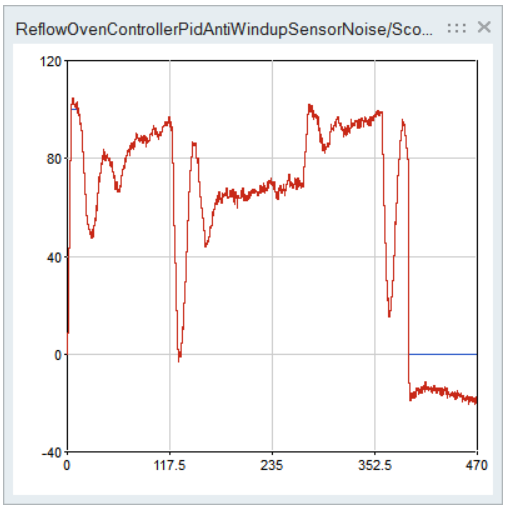


Figure 4.21: Sensor Noise

- It is possible to use a simple, rather power-limited (1500W) electrical oven for reflow soldering purposes.
- Conventional PID-control is suitable for automatic control of the oven temperature, the controller is able to
  - track compliant reflow soldering profiles adequately
  - compensate varying ambient temperatures
  - suppress noise introduced by a temperature sensor



(a) Oven Temperature



(b) Controller Output (Duty Cycle)

Figure 4.22: PID Anti-Windup Controlled Oven Temperature with Sensor Noise

Chapter 5

Coselica Use Case 3 - Electrically Actuated Hatch Mechanism

5.1 Scenario and Problem

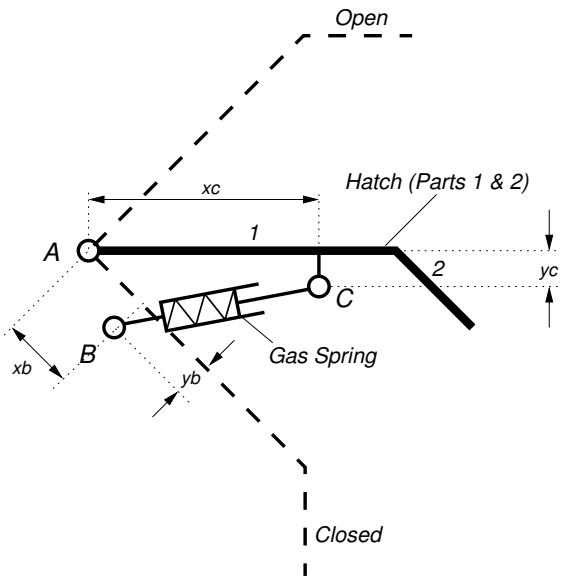


Figure 5.1: Hatch Mechanism with Gas Spring

Manual opening and closing of a hatch is usually be eased by a spring. A hatch mechanism with a spring is shown in fig. 5.1. The hatch (thick solid) – a rigid body with mass and inertia – is hinged to a fixed position `A`. Gravity acceleration in negative vertical direction shall be taken into account. The flanges of a *gas spring* are connected to a fixed position `B` (x_b, y_b) and a position `C` (x_c, y_c) located on the hatch.

The spring serves several purposes, it keeps the hatch in the rest positions `Closed` and `Open` (dashed lines), facilitates manual opening of the hatch, and avoids too hard bumping into the `Closed` position. However, this will work only for appropriately chosen positions `B` and `C`, and a spring with suitable parameters (extended length, stroke, force characteristic).

In this use case, we are especially interested in how to electrically actuate the hatch mechanism for automatic opening and closing. For this purpose, we will try to use a spindle, driven by a DC motor, to actuate the spring itself.

5.1.1 Gas Spring

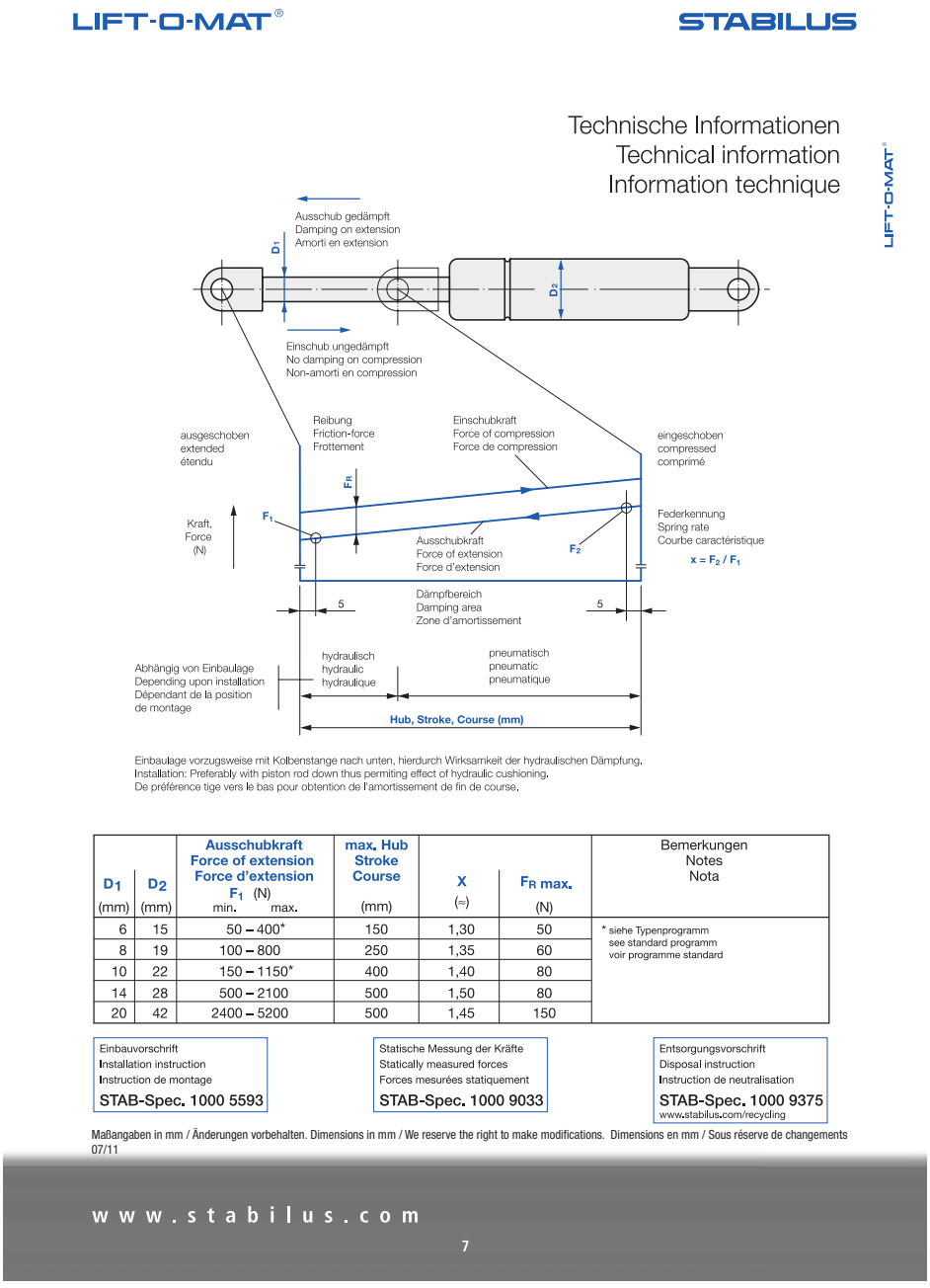
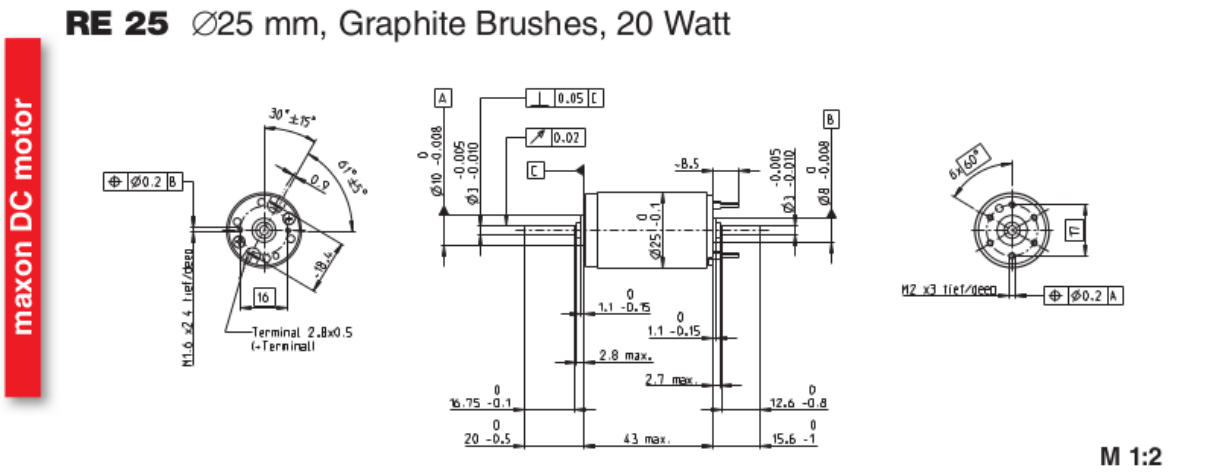


Figure 5.2: Gas Spring Parameters (Source: http://www.stabilus.com)

Parameters of some available standard gas springs are given in fig. 5.2. For the sake of simplicity, we are a priori restricting our choice of an appropriate gas spring as follows. We are taken only springs with a `Stroke A` of 180mm into account. In contrast to the data sheet, we are assuming an increased `Extended length B` of 505mm instead of 445mm (provides additional space for a future spindle drive). Furthermore, the `Friction-force` F_R shall be always zero.

5.1.2 DC Motor

We will use a standard DC Motor `RE 25` (`P/N 339150`), whose parameters are given in fig. 5.3.



Stock program
Standard program
Special program (on request)

			Part Numbers										
			302534	339149	339150	339151	339152	339153	339154	339155	339156	339157	339158
Motor Data													
	Values at nominal voltage												
1	Nominal voltage	V	7.2	9	12	18	24	30	36	48	48	48	48
2	No load speed	rpm	10500	9710	9620	10400	10900	9210	10100	9540	8450	6720	4650
3	No load current	mA	133	93.2	68.1	50.6	40.2	25	23.7	16.4	13.7	9.89	6
4	Nominal speed	rpm	8970	8260	8310	9190	9690	8010	8860	8360	7270	5530	3430
5	Nominal torque (max. continuous torque)	mNm	21.9	24.4	27.5	29.1	30.4	31.4	30.7	31.7	32.3	32.9	32.8
6	Nominal current (max. continuous current)	A	3.68	2.97	2.45	1.85	1.5	1.04	0.931	0.68	0.614	0.495	0.341
7	Stall torque	mNm	259	238	268	297	325	265	279	270	243	192	127
8	Starting current	A	42.1	28.1	23.2	18.4	15.6	8.61	8.24	5.67	4.51	2.84	1.3
9	Max. efficiency	%	79	81	84	86	88	88	88	89	89	88	86
	Characteristics												
10	Terminal resistance	Ω	0.171	0.32	0.517	0.98	1.53	3.49	4.37	8.47	10.6	16.9	36.8
11	Terminal inductance	mH	0.0163	0.0308	0.0573	0.112	0.186	0.407	0.493	0.979	1.25	1.97	4.11
12	Torque constant	mNm/A	6.15	8.46	11.5	16.1	20.8	30.8	33.8	47.7	53.8	67.7	97.6
13	Speed constant	rpm/V	1550	1130	828	591	460	311	282	200	177	141	97.8
14	Speed / torque gradient	rpm/mNm	43.2	42.8	37.1	35.9	34	35.2	36.5	35.6	35.1	35.2	36.9
15	Mechanical time constant	ms	6.52	6.06	5.62	5.36	5.24	5.17	5.16	5.13	5.12	5.12	5.14
16	Rotor inertia	gcm²	14.4	13.5	14.5	14.3	14.7	14	13.5	13.8	13.9	13.9	13.3

Figure 5.3: DC Motor Parameters (Source: http://www.maxonmotor.com)

5.1.3 Open Questions

We would like to investigate and answer by modeling and simulation the following questions:

- Where should a gas spring with stroke 180mm and an extended length of 505mm be mounted (positions `B` & `C`)?

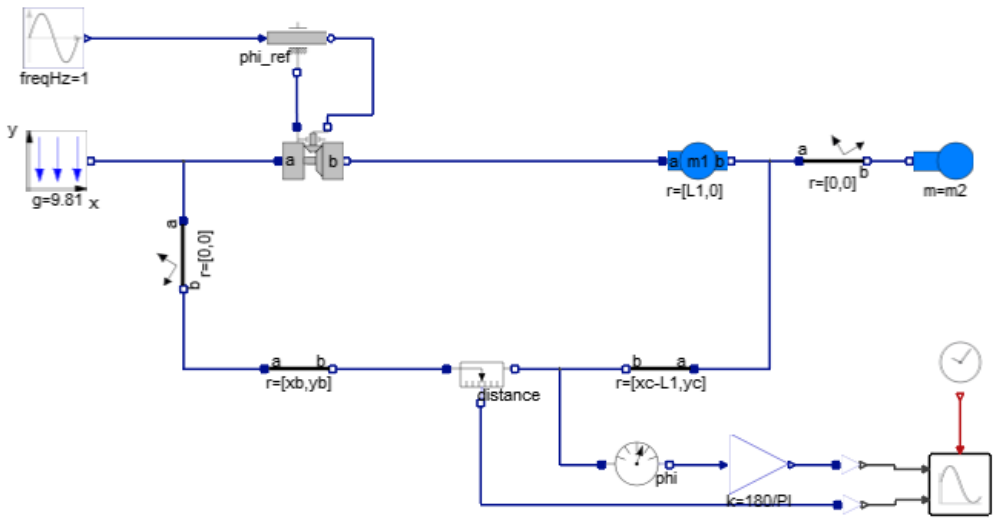


Figure 5.4: Kinematics of Hatch Mechanism (`HatchMechanismGeometry`)

- Which gas spring with respect to its `Force of extension` F_1 (cf. fig. 5.2) is suitable for a manual actuated hatch?
- Is it possible open and close the hatch automatically by means of an electrical spindle drive which changes the force characteristics of the spring?

5.2 Modeling and Simulation

In a first step, we are investigating the kinematics of the mechanism in order to find appropriate mounting positions for the gas spring and the dynamics in order to choose a suitable spring force characteristic.

In a second step, we are actuating the hatch mechanism for opening and closing it. We are modeling three different actuation methods:

- Exertion of manual pushes directly to the hatch
- Control of the spring preload
- Usage of an electrical spindle drive to control the spring preload

5.2.1 Gas Spring

We are modeling the hatch mechanism without and with gas spring. Simulations are carried out in order to find appropriate mounting positions for the spring and to select a suitable spring force characteristic.

Mounting Positions

We are trying to find appropriate mounting positions `B` & `C` (cf. fig. 5.1) for a preselected gas spring (extended length 505mm, stroke 180mm; cf. sec. 5.1.1). We are assuming, that only $y_b = y_c = 0.04$m are possible. Thus, we are looking for x_b=?, such that the spring is fully extended in position `Open` ($+45°$) and fully compressed in position `Closed` ($-45°$). It is obvious, that x_c depends on x_b and can be calculated as

$$x_c = x_b + (extended\,length - stroke) + 0.005m. \tag{5.1}$$

We have build a model as shown fig. 5.4 to be able to simulate the kinematics for an arbitrary position x_b. Herein, the hatch is modeled by two rigid bodies[1] (`Hatch Body(1)` and `Hatch Body(2)`). `Hatch Body(1)` is a body of length $L1$=0.6m and its center of mass is at position $L1/2$ (cf. block parameters `r` and `r_CM`). `Hatch Body(1)` is a body, whose center of mass located at $L2/2$ with respect to its left (only) flange (cf. block parameter `r_CM`).

The `Rotation` block between them, performs just a coordinate transformation (translation `r`=[0,0], rotation `angle`=-45°) and is a matter of convenience[2]. The hatch is hinged (via an actuated `Revolute Joint A`) at the origin of the `World Frame`. Position of `B` is specified by a `Rotation` of the `World Frame` and a consecutive `Translation to B`. Position `C` is specified by a `Translation to C` which starts at the right end of `Hatch Body (2)`. A `Position` (`Prescribe Angle`) block is used to actuate the revolute joint according to a real signal `phi_ref`. We are prescribing a `Hatch Angle (Closed...Open)`, varying from $-45° \ldots +45°$, and measuring the corresponding `Length of Gas Spring`, i.e. the distance between `B` and `C`.

The spring length versus hatch angle for different values of x_b is shown in fig. 5.5. Please note, that the damping areas of the gas spring(cf. fig. 5.2) are not taken into account, thus the allowable spring length is $330 \ldots 500$mm. For x_b=0.05m and x_b=0.10m, the stroke of the spring would not be fully used (cf. fig. 5.5a and 5.5b). For x_b=0.015m, the hatch could not be fully opened (cf. fig. 5.5c). Finally, a position x_b=0.012m may be regarded as appropriate within the scope of this use case (cf. fig. 5.5d).

We can state here, that we have found appropriate mounting positions for the gas spring.

Force of Extension

We are now trying to select a force characteristic, i.e. a suitable `Force of extension` (cf. fig. 5.2). For this purpose, we have changed the model (cf. fig. 5.6) in order to investigate the dynamics of the mechanism. We are using now an *unactuated* `Revolute Joint A` and exerting a `LineForce` on the mountings points `B` & `C`, which resembles the spring force.

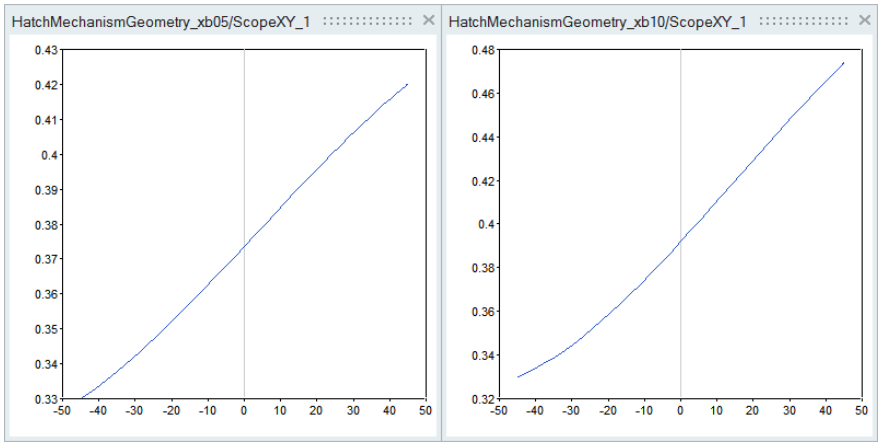
The `Gas Spring` (super block) itself is modeled in the translational mechanics domain as show in fig. 5.7. A `Spring` is `Preload`ed with the `Force of extension` F_1. The spring constant

$$c = \frac{F_1(x-1)}{A - 0.01\mathrm{m}}$$

is calculated according to the parameters given in fig. 5.2. Furthermore, we are assuming that the `Spring` bears viscous damping (we use $d = 1000\frac{\mathrm{Ns}}{\mathrm{m}}$). The stroke is restricted to stay within the `Closed Position` and the `Open Position`. In case of excess highly stiff and viscous `Damping`

[1] This assembly might be replaced by just one body block, if center of mass and inertia of the hatch as a whole is known

[2] The vector r_CM to the center of mass of `Hatch Body(2)` is easily specified in the rotated frame



(a) Position x_b =0.05m (b) Position x_b =0.10m

(c) Position x_b =0.15m (d) Position x_b =0.12m

Figure 5.5: Spring Length versus Hatch Angle

`Area`s will be activated (by means of `ElastoGap` blocks with parameters `c`=5e7N/m and `d`=1e6Ns/m). The spring `Force` and `Length` are provided as real signals for testing and plotting purposes.

We are simulating the hatch mechanism starting with an initial hatch angle $0°$ (cf. parameters `initType` and `phi_start` of `Revolute Joint A` block), i.e. the hatch is half-open at t=0.

Simulation shall reveal the smallest `Force of extension` F_1, which will drive the hatch autonomously into the `Open` ($+45°$) position and keep it there. Results for different F_1 are shown in fig. 5.8. A spring with $F_1 = 400$N is not capable to open the hatch (cf. fig. 5.8a). However, a spring with $F_1 = 500$N is strong enough to open the hatch and keep it open (cf. fig. 5.8b).

In summary, we have found an appropriate gas spring, which seems to be suitable to facilitate opening and closing of our hatch mechanism.

5.2.2 Actuated Hatch

The different methods of hatch actuation are modeled and investigated.

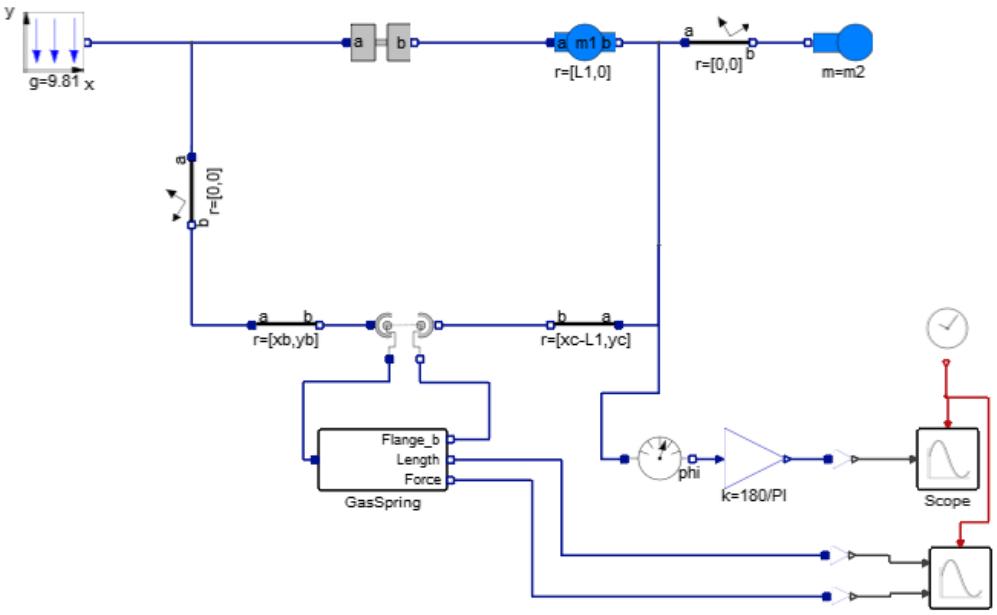


Figure 5.6: Hatch Mechanism with Gas Spring (`HatchMechanismGasSpring`)

Manually

We are testing our hatch mechanism with respect to its manual actuation and use the model shown in fig. 5.9. The hatch shall be driven from `Closed` to `Open` position and vice versa by a vertical push (`Operating Force`), which is applied to the very right rim of the hatch (right flange of `Hatch Body (2)`). The prescribed `Operating Force` (a `WorldForce` block) is resolved in the world frame. Thus, there is no, i.e. a zero, `Horizontal` force component and the `Vertical` push signal is generated by a `Trapezoid` block.

Simulation results, i.e. operating force and hatch angle, for opening the hatch are shown in fig. 5.10. A trapezoid shaped push force (blue) of different heights, 30N, 35N, and 40N (cf. parameter `amplitude` of `Vertical` block), is applied to the hatch and the hatch angle angle (red) was recorded. In case of 30N, there is no visible movement of the hatch (cf. fig. 5.10a). Pushing a bit stronger, with 35N, reveals that the hatch is opening a little, but traveling back to its Closed position. Pushing stronger, now with 40N, is strong enough to open the hatch fully (cf. fig. 5.10c). Please note, that at the end of the push ($t = 1s$) the hatch has not yet reached the `Open` position, but it is moving on autonomously.

Now, lets look a corresponding results for manual closing of the hatch. Please note, that our push period is here longer (2s) than before (1s; cf. parameters `rising`, `width`, and `falling` of `Vertical` block) and we have used 10N, 20N, and 30N downward (negative) pushes. Using 10N, we are not pushing strong enough, the hatch remains in `Open` position (cf. fig. 5.11a). With 20N, the hatch is traveling quite a bit towards the `Closed` position, but a the end of the push (2s) it starts moving back to the `Open` position (cf. fig. 5.11b). Finally, with 30N, the hatch travels to the fully `Closed` position and remains there (cf. fig. 5.11c).

We can resume, that our hatch mechanism works well and facilitates manual opening and closing of the hatch.

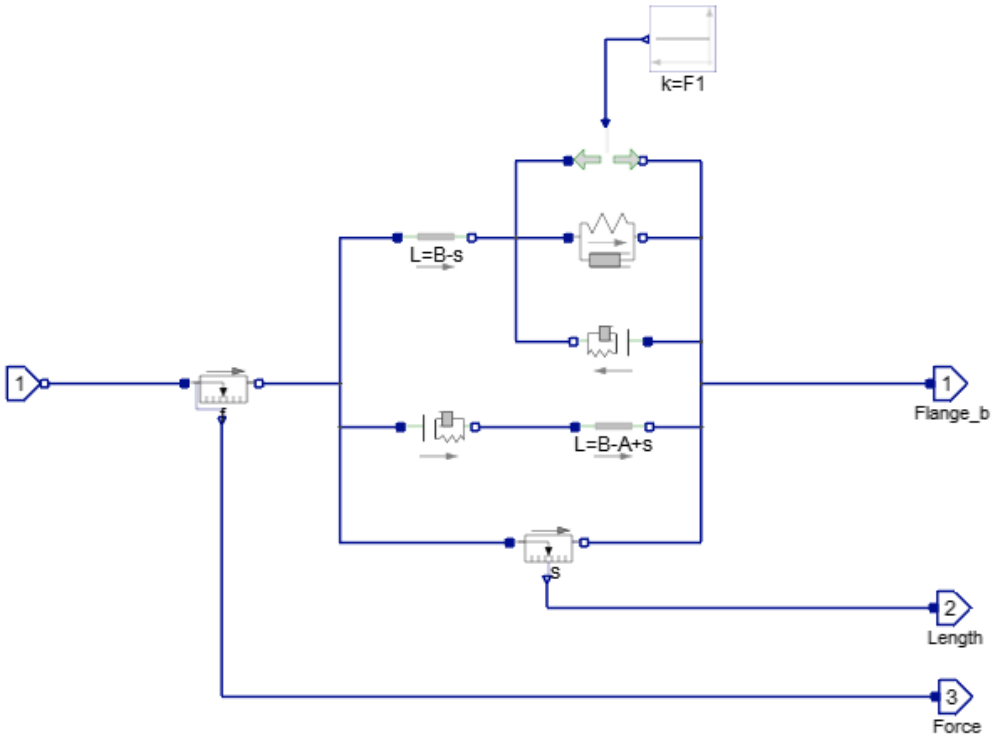


Figure 5.7: Model of Gas Spring (super block `Gas Spring`)

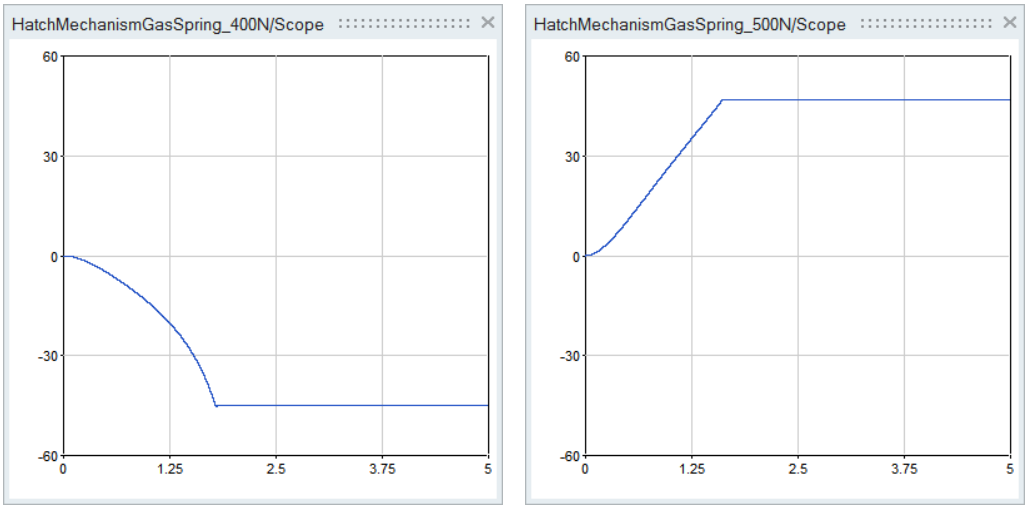
Spindle Driven

We modify the gas spring, such that it can be used as an actuator, which might later be driven by a electrical spindle drive (cf. sec. 5.2.2). Two additional spindle flanges (`Spindle_a` and `Spindle_b`) are added to the gas spring model (`Gas Spring w/ Spindle`) as shown in fig. 5.12. Furthermore, we use a somewhat stiffer force characteristic (`Force of extension` F_1=400N, `Spring rate` x=2; cf. fig. 5.2).

The model of this new gas spring is shown in fig. 5.13. Moving `Spindle_b` with respect to `Spindle_a` causes a change of the `Spring` preload. The relative position of `Spindle_b` with respect to `Spindle_a` shall be within ± 0.1m. Positive values can be interpreted as increase and negative values as decrease of the `Spring` preload. Please note, that the `Spring` and `Damping` are here modeled by separates block (instead of a single `SpringDamper` block as in fig. 5.7), because we are assuming that movements of the spindle flanges are not afflicted with damping.

Now, lets go back to our complete hatch mechanism in fig. 5.12, we are prescribing a profile for the in order to open and consecutively close the hatch. A `Spindle Position` signal (using a `Trapezoid` block) `s_ref` is generated and a translational `Position` (named `Spindle`) block is used to prescribe the spindle position accordingly.

Simulation results, spindle position (blue curve) and hatch angle (red curve), are shown in fig. 5.14. The spindle position starts at -0.1m (minimal spring preload) and the hatch is fully `Closed` ($-45°$). The spindle is driven to $+0.1$m (maximal spring preload) at t=2s and remains there. A visible opening movement of the hatch starts $t \approx 2$s and it travels to the fully `Open` ($+45°$) position. At t=6s the spindle position is moving towards -0.1m, which is reached at t=8s. The open hatch starts moving at $t \approx$7s



(a) Force of Extension $F_1 = 400N$ (b) Force of Extension $F_1 = 500N$

Figure 5.8: Hatch Angle versus Time

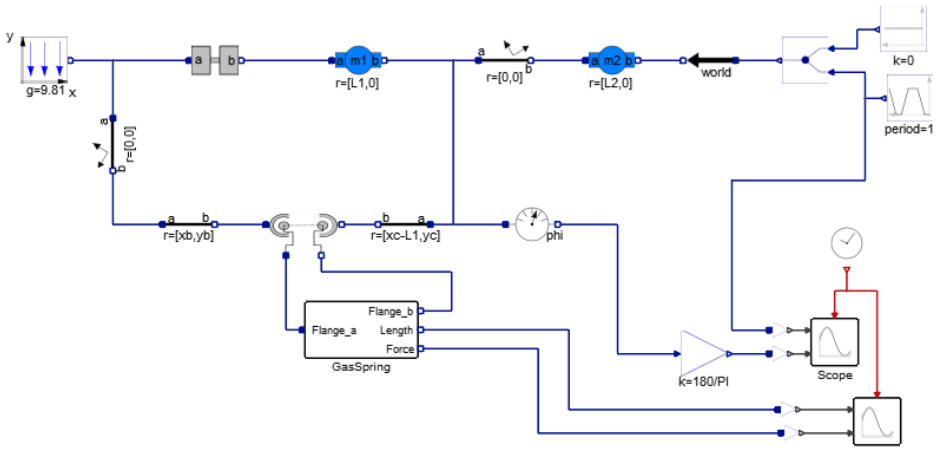
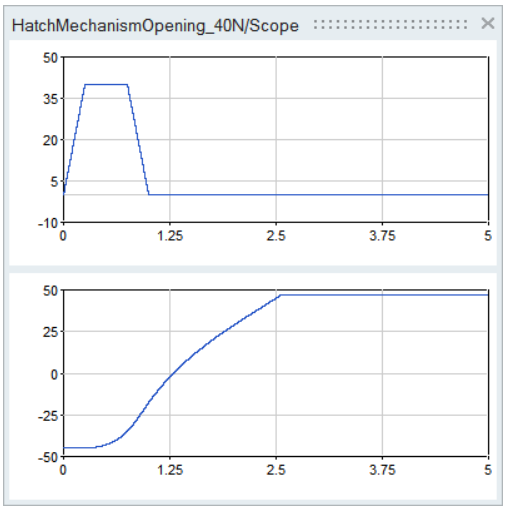


Figure 5.9: Manually Actuated Hatch Mechanism (`HatchMechanismOpening/Closing`)

(a) Opening Push 30N

(b) Opening Push35N



(c) Opening Push 40N

Figure 5.10: Operating Force and Hatch Angle

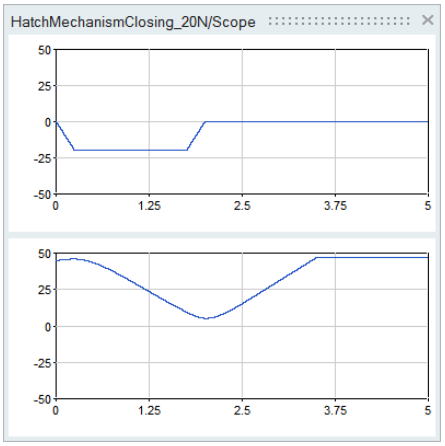
towards it closed position and at $t \approx$9s the hatch is again fully closed. Thus, we have proved, that the hatch can be actuated by controlling the preload of the gas spring.

Electrical Spindle Drive

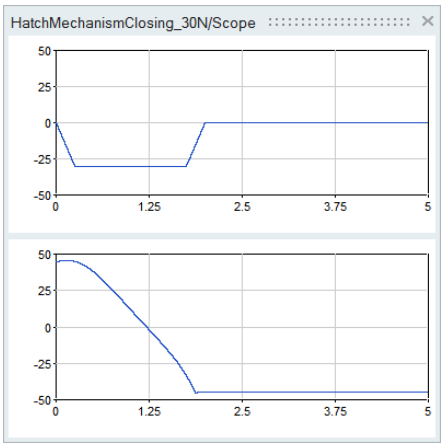
An `Electrical Spindle Drive` is used to actuate our hatch mechanism as shown in fig. 5.15. The spindle flanges of the gas spring (`Gas Spring w/ Spindle Flange`) are connected to an electrical drive (`Electrical Spindle Drive`). The spindle drive takes a reference spindle position (`Ref. Position`) as real input signal and provides the actual `Position` as real output signal.

The content of the super block `Electrical Spindle Drive` is shown in fig. 5.16. The `Spindle` (an `IdealGearR2T` block; assumed transmission ratio is 1 rotation per 0.00075mm) converts rotational into translational movement and is driven by a DC motor (`P/N 339150`; cf. fig. 5.3). The motor

(a) Closing Push 10N



(b) Closing Push20N



(c) Closing Push 30N

Figure 5.11: Operating Force and Hatch Angle

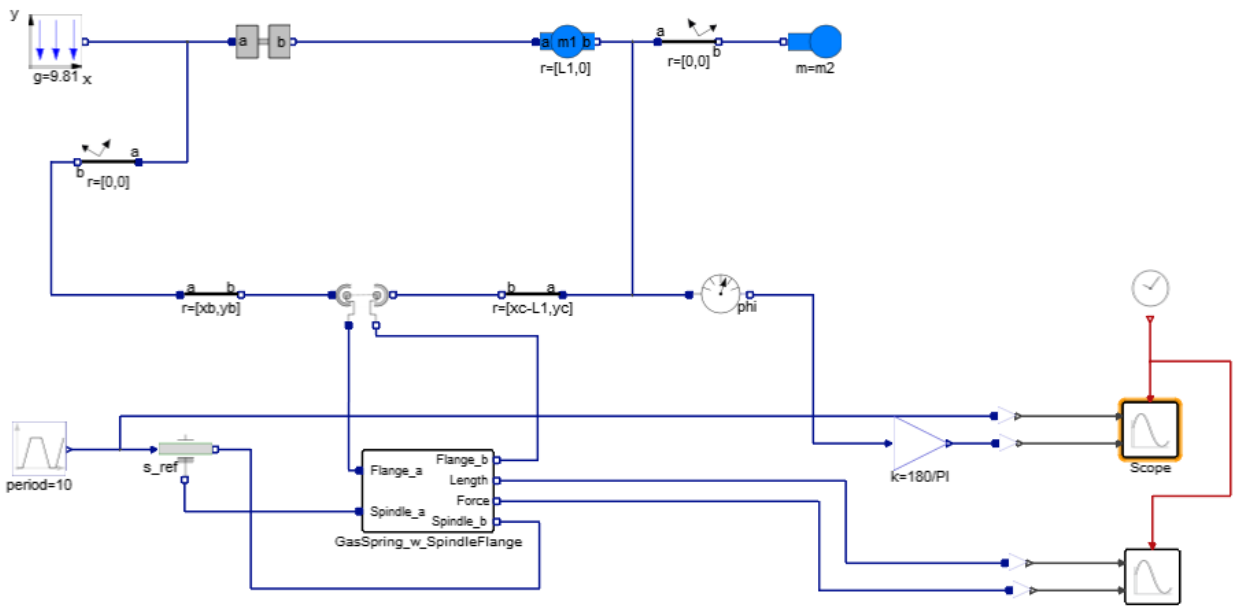


Figure 5.12: Spindle Driven Hatch Mechanism (`HatchMechanismActuatedSpindle`)

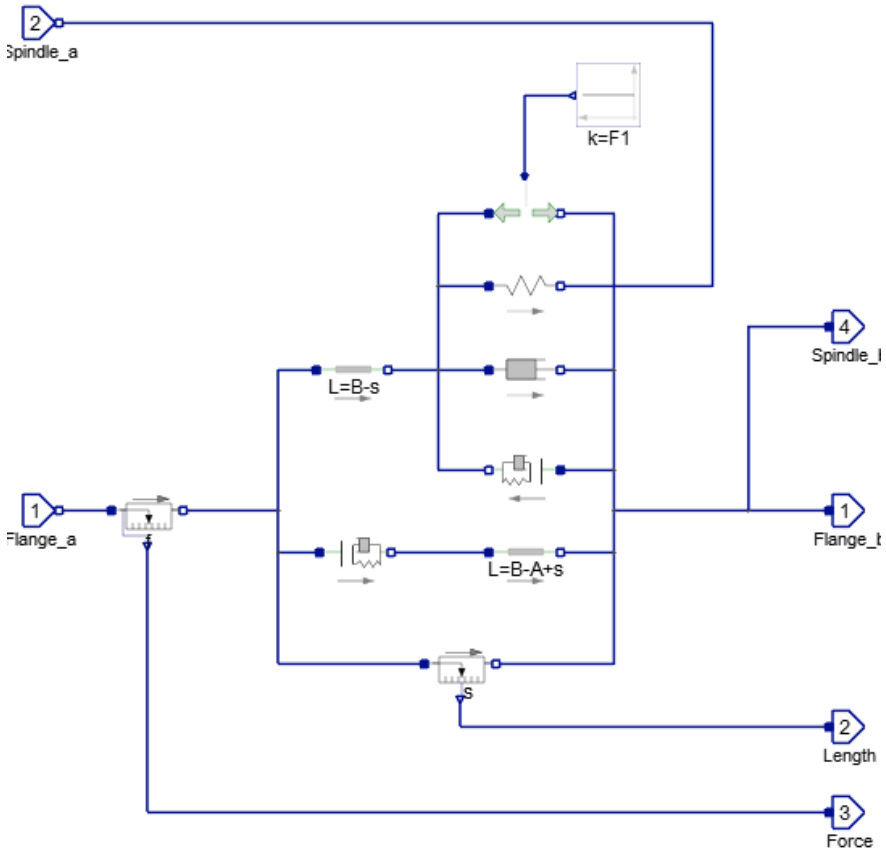


Figure 5.13: Model of Gas Spring with Spindle Flanges

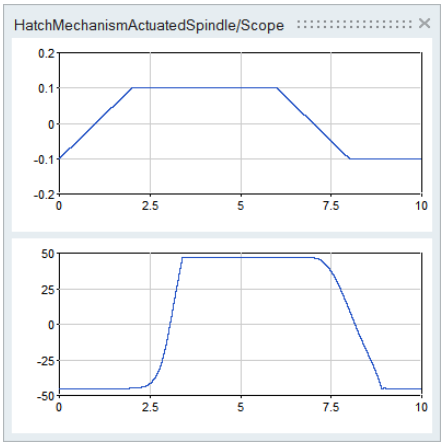


Figure 5.14: Spindle Position and Hatch Angle

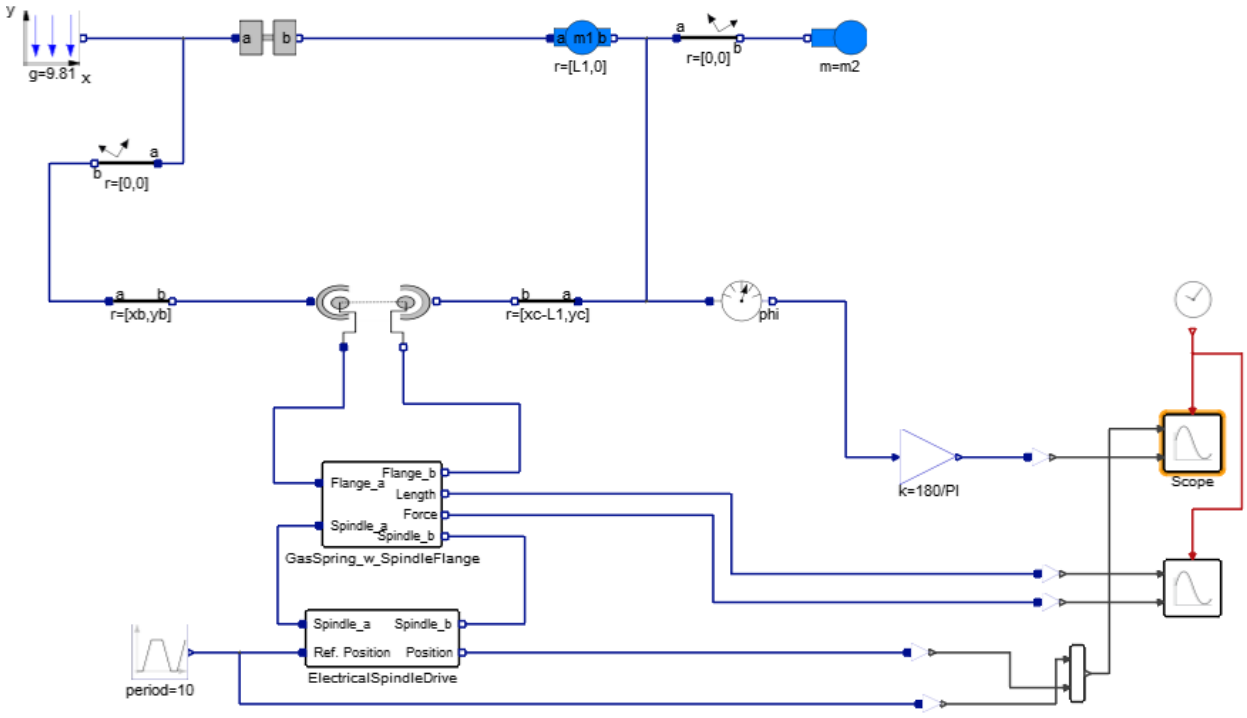


Figure 5.15: Electrically Spindle Driven Hatch Mechanism
(`HatchMechanismActuatedSpindleElectrical`)

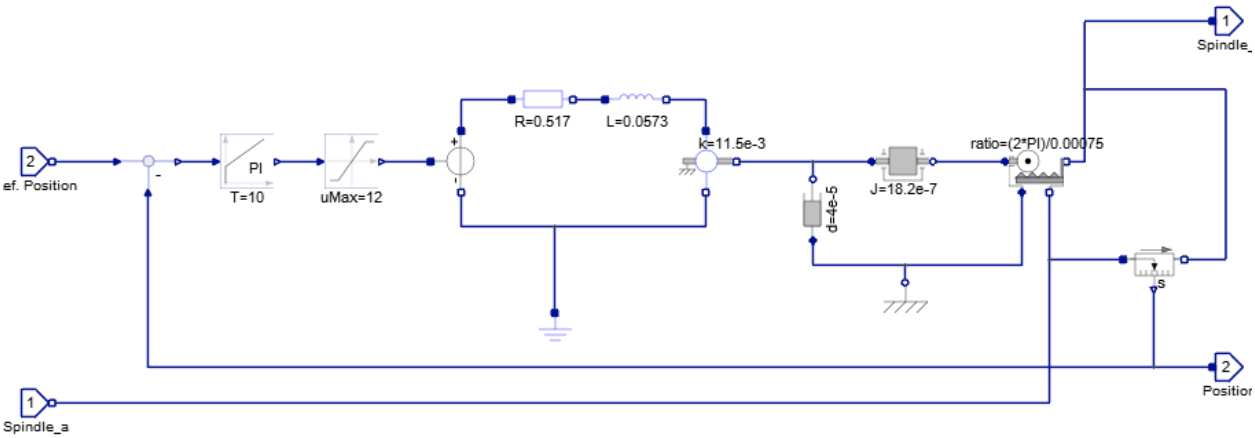


Figure 5.16: Model of Electrical Spindle Drive

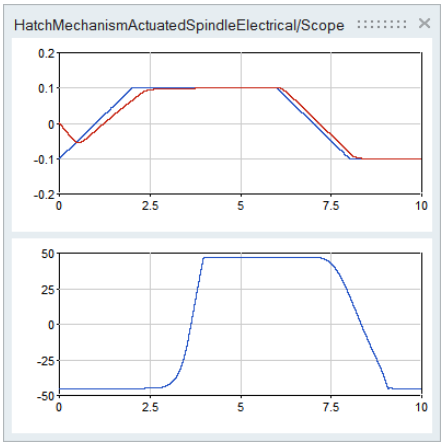


Figure 5.17: Spindle Position and Hatch Angle

has to overcome the inertia of its rotor and the inertia of the spindle (which is modeled by one `Rotor & Spindle` inertia block; parameter `J`=14.5e-7 (rotor) + 3.7e-7 (spindle) = 18.2e-7), and viscous damping `Losses` (which are roughly estimated as `d`=4e-5). Please note, that for the sake of simplicity, we are not taken COLUMB-like friction/stiction into account. In the electrical domain, the motor is modeled by a `Resistance`, an `Inductance`, and an electro-motoric force (`EMF`) block. The respective parameters are taken from the motor data sheet (cf. fig. 5.3). The motor is driven by a `Voltage Source`, which is limited to `-12V...+12V`. and controlled by a `PI Controller`.

The `PI Controller` is effectively used to control the measured `Position` of the spindle by feedback. Thus, it is compared with a desired reference position (`Ref. Position`) and the control error is provided as input to the controller.

Simulation results of the electrical actuated hatch mechanism are shown in fig. 5.17. We have used the same spindle position profile (blue) as in the previous section (cf. fig. 5.14). The actual position of the spindle (red curve in top diagram) starts at 0.0m and follows, due to the feedback control, sufficiently well the prescribed position profile (blue curve in top diagram). The hatch opens and closes as expected (cf. bottom diagram in fig. 5.17). Please note, that opening and closing is a little bit delayed in comparison to our previous results (cf. sec. 5.2.2, fig. 5.14). This is due to the tracking delay introduced by our electrical spindle drive (cf. fig. 5.17).

In summary, it is possible to actuate the hatch by by means of an electrical spindle drive which controls the preload of the gas spring.

5.3 Major Results

Based on the above carried out simulation study, we can summarize the following results:

- We did find appropriate mounting positions B & C for a given spring geometry
- We were able to select a spring force characteristic suitable for manual actuation of the hatch
- We have proved, that it is possible to open and close the hatch automatically by means of an electrical spindle drive which controls the preload of the spring