Technische Universität München
TUM School of Engineering and Design

Stable and efficient simulation of manipulation and assembly processes in object-oriented modeling languages

Robert Michael Reiser

Vollständiger Abdruck der von der TUM School of Engineering and Design der Technischen Universität München zur Erlangung eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz: Prof. Dr.-Ing. Hans-Georg Herzog

Prüfende der Dissertation:

1. Hon.-Prof. Dr.-Ing. Martin Otter
2. Prof. Dr. rer. nat. Martin Arnold

Die Dissertation wurde am 09.01.2026 bei der Technischen Universität München eingereicht und durch die TUM School of Engineering and Design am 16.06.2026 angenommen.

Vorwort

Diese Dissertation entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Systemdynamik und Regelungstechnik sowie am Institut für Robotik und Mechatronik des Deutschen Zentrums für Luft- und Raumfahrt (DLR).

Mein besonderer Dank gilt Herrn Prof. Dr.-Ing. Martin Otter für die Betreuung der Arbeit. Insbesondere bedanke ich mich für die fachlichen Diskussionen und wertvollen Anregungen. Weiterhin danke ich Herrn Prof. Dr.-Ing. Johann Bals für die großzügige Förderung am Institut, die Unterstützung bei der Präzisierung des Themas und den gewährten Freiraum bei der Erstellung der Arbeit. Herrn Prof. Dr.-Ing. Alin Albu-Schäffer danke ich ebenfalls für den gewährten Freiraum für die Fertigstellung der Arbeit am Institut. Herrn Prof. Dr. rer. nat. Martin Arnold danke ich für die Übernahme des zweiten Gutachtens. Außerdem danke ich Herrn Prof. Dr.-Ing. Hans-Georg Herzog für die Übernahme des Prüfungsvorsitzes.

Außerdem bedanke ich mich herzlich bei meinen Kolleginnen und Kollegen für die vielen fruchtbaren Diskussionen, das stets konstruktive Feedback und die tolle Arbeitsatmosphäre am Institut. Mein besonderer Dank gilt Tobias Bellmann für die vielen bereichernden Diskussionen und die Unterstützung bei der Themenfindung. Hervorzuheben sind außerdem Bernhard Thiele, Fabian Buse und Antoine Pignède. Weiterhin bedanke ich mich bei Matthias Hellerer, Sebastian Kümper, Gerhard Hippmann, Carsten Oldemeyer, Andrea Neumayr, Dirk Zimmer, Raphael Gebhart und Rainer Krenn.

Abschließend bedanke ich mich bei meiner Familie und meinen Freunden. Danke euch, liebe Eltern, für die Unterstützung im Studium und die Ermutigung beim Verfassen dieser Arbeit. Danke dir, liebe Anja, für die Hilfe bei sprachlichen Fragen. Danke euch, liebe Freunde, für euren Rückhalt und dass ich auf euch zählen kann. Und schließlich gilt mein herzlicher Dank dir, liebe Kathi, für die Geduld und Ermutigung während des Schreibens.

Gilching, Januar 2026 Robert Reiser

Abstract

Manipulation and assembly processes are essential in many production areas and offer considerable potential for robotic automation. Their continuous improvement is driven by criteria such as time, quality, energy consumption, and cost. Simulation plays an important role in the optimization of these processes. Many tools are available for various tasks in robotic assembly simulation, but most are limited to the mechanical domain and do not support simulations of multi-domain models.
A detailed simulation of assembly processes requires consideration of multiple domains, such as a detailed simulation of drivetrains, including electric motors. This is even more relevant for automatic disassembly, where, for example, parts may need to be heated. Such tasks call for system simulation tools like Modelica environments or Simscape, which are based on object-oriented modeling languages and event-based solvers. In contrast, most robotic simulation tools use physics engines with non-smooth contact models and specialized solvers based, for example, on time-stepping methods. These approaches are not well suited for stiff multi-domain components as found, for example, in detailed drivetrain models. However, system simulation tools typically do not support dynamic structural changes of physical components during runtime as required in assembly and disassembly. In addition, they usually do not support the modeling of contacts between complex, non-convex rigid bodies.
In this work, a new approach for simulating assembly and disassembly processes in system simulation tools in a stable and efficient way is presented. It relies on compliant contact models, automatic generation and management of component pairs, and robust handling of hierarchical contacts. The object-oriented modeling language of a system simulation tool is extended with dedicated models for manipulation, assembly, and disassembly at two levels of detail: fast substitute models based on compliant forces that are a function of penetration depth and detailed contact models based on compliant, discretized contact surfaces. The former enables simulations in real-time for tasks like virtual commissioning.
Both model types are integrated into reusable component models that can be used to create models of assembly and disassembly processes. The created process models can be simulated at both levels of detail, significantly reducing the modeling effort. In addition, this extended modeling framework includes an intuitive description of assembly processes and assembly hierarchies, eliminating the need for modeling all potentially occurring contact pairs.
The developed solution is implemented with the multi-domain modeling language Modelica extended with specialized algorithms through an external environment. Several software packages are combined and optimized to minimize computation time. The new approach is demonstrated with assembly and disassembly use cases, including a furnace heating process.

Kurzfassung

Manipulations- und Montageprozesse sind in vielen Bereichen der Produktion unverzichtbar und bieten ein erhebliches Potenzial für die robotische Automatisierung. Ihre kontinuierliche Verbesserung wird durch Kriterien wie Zeit, Qualität, Energieverbrauch und Kosten vorangetrieben. Dabei spielen Simulationen eine wichtige Rolle. Die meisten Tools sind jedoch auf die mechanische Domäne beschränkt und unterstützen keine Multidomänenmodelle.
Für eine detaillierte Simulation von Montageprozessen müssen mehrere Domänen berücksichtigt werden, wie etwa eine detaillierte Simulation von Antriebssträngen einschließlich Elektromotoren. Dies gilt besonders für die automatische Demontage, bei der beispielsweise Bauteile erwärmt werden müssen. Solche Aufgaben erfordern Systemsimulationstools wie Modelica-Umgebungen oder Simscape, die auf objektorientierten Modellierungssprachen und ereignisbasierten Lösern basieren. Im Gegensatz dazu verwenden die meisten Robotersimulationstools Physik-Engines mit nicht-glatten Kontaktmodellen und speziellen Lösern, die beispielsweise auf Zeitschrittmethoden basieren. Diese Methoden eignen sich nicht gut für steife Multidomänenkomponenten, wie sie etwa in detaillierten Antriebsstrangmodellen zu finden sind. Systemsimulationstools unterstützen jedoch meist keine dynamischen strukturellen Änderungen physikalischer Komponenten während der Laufzeit, wie sie bei der Montage und Demontage erforderlich sind. Außerdem unterstützen sie in der Regel nicht die Modellierung von Kontakten zwischen komplexen, nicht konvexen, starren Körpern.
In dieser Arbeit wird ein neuer Ansatz für die stabile und effiziente Simulation von Montageprozessen in Systemsimulationstools vorgestellt. Er basiert auf nachgiebigen Kontaktmodellen, der automatischen Generierung und Verwaltung von Komponentenpaaren und der robusten Handhabung hierarchischer Kontakte. Die objektorientierte Modellierungssprache eines Systemsimulationstools wird um dedizierte Modelle für Manipulation, Montage und Demontage in zwei Detailgraden erweitert: schnelle Ersatzmodelle auf der Grundlage von nachgiebigen Kräften, die eine Funktion der Eindringtiefe sind, und detaillierte Kontaktmodelle auf der Grundlage von nachgiebigen, diskretisierten Kontaktflächen. Ersteres ermöglicht Simulationen in Echtzeit für Aufgaben wie die virtuelle Inbetriebnahme.
Beide Modelltypen sind in wiederverwendbare Komponentenmodelle integriert, mit denen Modelle für Montage- und Demontageprozesse erstellt werden können. Die erstellten Prozessmodelle können in beiden Detailgraden simuliert werden, was den Modellierungsaufwand erheblich reduziert. Außerdem umfasst dieses erweiterte Modellierungsframework eine intuitive Beschreibung von Montageprozessen und -hierarchien, sodass nicht mehr alle potenziell auftretenden Kontaktpaare modelliert werden müssen.
Die entwickelte Lösung wird mit der domänenübergreifenden Modellierungssprache Modelica implementiert, die durch eine externe Umgebung um spezielle Algorithmen erweitert wird. Mehrere Softwarepakete werden kombiniert und optimiert, um die Rechenzeit zu minimieren. Der neue Ansatz wird anhand von Montage- und Demontage-Anwendungsfällen demonstriert, darunter ein Aufwärmvorgang in einem Ofen.

Acronyms

AABB Axis-Aligned Bounding Box. 8, 19, 21, 25, 62, 64, 68

ACI Algorithm for Component Interaction. 5, 33, 34, 59, 60, 64, 104

AGPSA Assembly Graph Partner Search Algorithm. 5, 34, 45, 49–54, 60, 66, 77, 81, 82, 104, 105

BEV Battery Electric Vehicle. 22, 24, 93, 105

BV Bounding Volume. 8, 25, 34, 62

BVH Bounding Volume Hierarchy. 8, 9, 14, 15, 17, 19, 105

CAD Computer-Aided Design. 24, 106

DLR German Aerospace Center (Deutsches Zentrum für Luft- und Raumfahrt). 1, 87, 93, 123, III

EE External Environment. 3, 5, 59–61, 64–67, 83, 84, 104

EFM Elastic Foundation Model. 13, 14, 16, 21, 35

EPA Expanding Polytope Algorithm. 8, 9, 21

FMI Functional Mockup Interface. 24, 37, 122

FoF Factory of the Future. 1, 87

GJK Gilbert-Johnson-Keerthi. 8, 9, 21, 25, 63

HiL Hardware in the Loop. 19, 23, 24, 26, 74, 123

LCP Linear Complementary Problem. 12, 20, 37

LoD Level of Detail. 2–5, 7, 19, 22–26, 55–58, 67, 71–87, 91, 92, 100–105

MPR Minkowski Portal Refinement. 9, 21, 25, 59, 60, 62–64, 66–68, 83, 89, 105, 106

MSL Modelica Standard Library. 19, 21, 69, 70, 88, 95, 96, 123

OBB Oriented Bounding Box. 8, 25

ODE Ordinary Differential Equation. 12, 37, 61

PCM Polygonal Contact Model. 9, 14, 15, 25, 36, 38, 40, 48, 59, 60, 63, 64, 66–68, 71, 73, 79, 83, 84, 89, 90, 98, 104–106

PFC Pressure Field Contact. 16, 17, 36, 38, 106

PVS Path Vertices Sequence. 46, 47, 49, 50, 52, 53, 120, 121

SDF Signed Distance Field. 9, 16

SFA Substitute Force Algorithm. 5, 34, 39, 41–45, 53, 55, 60, 66, 68, 70, 77, 81, 82, 88, 89, 98, 104, 105

SiL Software in the Loop. 19, 23, 26, 56, 122

VC Virtual Commissioning. 2, 4, 7, 19, 23, 24, 26, 32, 48, 55–57, 72, 73, 104

VPA Voxelmap Pointshell Algorithm. 9, 16, 25, 36, 38

VSS Variable-Structure Systems. 4, 7, 17, 22, 26

Contents

1 Introduction

This chapter begins with an overview of the topic and its relevance. In addition, the objectives and main contributions of this work are presented. Then, assumptions are made and boundary conditions are specified. In addition, the structure of the work is given.

1.1 Overview and motivation

Manipulation and assembly processes are widespread in many production areas, including aerospace, automotive, and mechanical engineering. Since assembly often accounts for a large portion of production costs, it offers significant potential for automation (Lotter and Wiendahl 2012). In addition, the trend towards individualized products requires greater flexibility in production systems, especially for robotic assembly cells. Furthermore, robotic disassembly processes will be increasingly necessary for recycling tasks in the future. These factors result in a continuous demand to improve these processes in terms of factors such as process time, quality, energy consumption, and cost.
Simulation plays an important role in the optimization of robotic manipulation, assembly, and disassembly processes. Figure 1.1 shows the simulation of an assembly process involving detailed contact forces between complex, non-convex rigid bodies. In this example, an electric motor is assembled into the housing of a chainsaw.

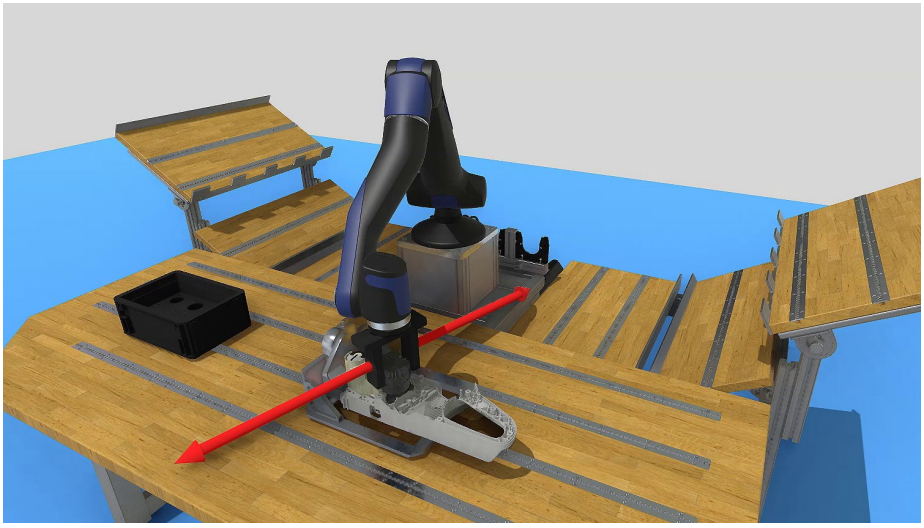

Figure 1.1: A simulation of the assembly process of an electric motor into the housing of an electric chainsaw in the DLR project *Factory of the Future*. The model provides detailed contact forces between the components (exemplarily represented by red arrows for the gripper jaws) based on a compliant contact force model.

In addition to optimization, simulations are used for tasks such as Virtual Commissioning. This method is used to test robot programs beforehand to detect errors in an early stage (Wünsch 2008). For some types of Virtual Commissioning, simulations that are real-time capable are required. However, the cost and effort required to create models is a major obstacle to the utilization of Virtual Commissioning (Striffler and Voigt 2023, p. 678).
In recent years, there has been an increasing trend towards collaborative robots. Industrial robots are typically expected to generate sufficient torque to reach a specific position (Thulesen 2016, p. 4). In contrast, for collaborative robots, it is important to have sufficient knowledge about torque limitations in order to push their limits. As a result, detailed models of robot dynamics, including models for drivetrains and motor torques, are becoming more important. This is especially true for collaborative robots.
There are many simulation tools available for various tasks in robotic assembly simulation. However, most are limited to the mechanical domain and do not support simulations based on multi-domain models. For the detailed simulation of assembly processes, multiple domains must be considered such as detailed models of the robot drivetrains, including the electric motors and their energy consumption. This is particularly relevant for automatic disassembly, as in some cases, parts need to be heated in order to be disassembled. This type of simulation requires system simulation tools such as *Modelica* environments or *Simscape*. These are based on object-oriented modeling languages and event-based solvers.
In contrast, most robotic simulation tools such as *CoppeliaSim* or *Gazebo* use physics engines with non-smooth contact models and specialized solvers based, for example, on time-stepping methods. These approaches are not well suited for stiff, multi-domain components, such as those found in detailed drivetrain models. However, system simulation tools are typically structure-invariant and do not support dynamic structural changes of physical components during runtime, such as adding or removing a connection between components. This capability is required for manipulation, assembly, and disassembly processes, though. In addition, these tools usually do not support the modeling of contacts between complex, non-convex, rigid bodies.
In this work, a new solution for the stable and efficient simulation of assembly and disassembly processes in system simulation tools is presented (see Chapter 3). It is based on compliant contact models and the automatic generation and management of component pairs (see Section 3.2.5). The solution extends the object-oriented modeling language of a system simulation tool with specialized models for manipulation, assembly, and disassembly at two Levels of Detail: fast substitute models and detailed contact force models.
The former models are based on compliant contact forces that are a function of penetration depth. These models enable dynamic structural changes in structure-invariant system simulation environments, such as the connection and disconnection of physical components during runtime (see Section 3.4), and enable real-time simulations for tasks such as Virtual Commissioning. In addition, these substitute models are extended to robustly handle hierarchical contacts, as found in assemblies (see Section 3.5). The latter models are based on compliant, discretized contact surfaces (see Section 3.3) and are used for simulations involving detailed contact forces for tasks such as process optimization.

Both the substitute models and the detailed models are integrated into reusable component models. These component models can be used to create process models of specific manipulation, assembly, and disassembly processes. The so created models of specific processes can then be simulated at both Levels of Detail (see Section 3.6). Therefore, the modeling effort is significantly reduced because a specific process only needs to be modeled once. Furthermore, this extended modeling framework provides an intuitive description of assembly processes and assembly hierarchies. This eliminates the need to manually model all potentially occurring contact pairs, as is usually the case with existing system simulation tools.
The developed solution is realized by using the object-oriented, multi-domain modeling language *Modelica* in the system simulation environment *Dymola* which is extended by multiple algorithms combined in an External Environment. Several software packages are being integrated and various optimizations are being made to improve the computation time. The capabilities of the new approach are demonstrated through multiple assembly and disassembly use cases, including a multi-domain simulation that combines a robotic disassembly process and a furnace heating process to estimate the overall power consumption.

1.2 Objectives and contributions of this work

The overall objective of this work is to develop a solution for detailed, multi-domain system simulations of manipulation, assembly, and disassembly processes using object-oriented modeling languages. This solution must be stable, efficient, highly flexible, and capable of handling complex, non-convex geometries while minimizing the modeling effort. This results in the following **main contributions** of this work:

- The development of an **algorithm for the interaction between components** in manipulation, assembly, and disassembly processes. This algorithm extends the object-oriented modeling languages of system simulation environments, including the automatic generation and management of component pairs (see Section 3.2.5).
- The **selection and integration of a contact force model** for the simulation of manipulation, assembly, and disassembly processes in system simulation environments based on compliant, discretized contact surfaces (see Section 3.3).
- The development of an **algorithm for dynamic structural changes** in order to connect and disconnect physical components during runtime in structure-invariant system simulation environments based on compliant contact forces (see Section 3.4).
- The development of an **algorithm for the robust handling of hierarchical contacts** for the stable and efficient simulation of assemblies (see Section 3.5), including an intuitive description of assembly processes and assembly hierarchies.

- The definition of **Levels of Detail** and the development of **component models** containing both the substitute models and the contact force models aiming to reduce the modeling effort (see Section 3.6).

1.3 Boundary conditions and assumptions

The following boundary conditions and assumptions apply to this work:

- The primary boundary condition of this work is the use of a structure-invariant system simulation environment based on an object-oriented modeling language. This means that no structural changes are possible during runtime. For example, the number of states cannot be changed and impulses cannot occur, because this is not supported by object-oriented modeling languages. Physical components cannot be connected or disconnected without the use of additional models.
- In addition, only polygonal representations of bodies are considered. As a result, bodies are only approximated in many cases (for example spheres and cylinders).
- The contact surfaces of bodies in contact are assumed to be rigid. However, deformable bodies can also be modeled in combination with other libraries. In this case, the surfaces of the bodies are split into segments.
- The reference for the validation of the models for connecting and disconnecting components during runtime in structure-invariant environments using substitute contact forces is the detailed contact force model (no hardware validation is performed).
- For this work, the term "real-time" is defined as a statistical adherence to time constraints, also known as *firm real-time* or *soft real-time*.

1.4 Structure of this work

This work outlines the conceptual design, development, realization, and application of a solution for the simulation of manipulation and assembly processes in structure-invariant environments. It is structured as follows:

- In **Chapter 2**, an analysis of the state of the art is presented and open issues are discussed. This includes collision detection and response, simulation environments, simulation of grasping processes, body contacts in system simulation tools, Virtual Commissioning, Variable-Structure Systems, and models with multiple Levels of Detail.
- The novel solution for the stable and efficient simulation of manipulation and assembly processes in structure-invariant environments with object-oriented modeling languages is presented in **Chapter 3**. The *Block Scenario* example is introduced and an

overview of the concept is given. Based on this example, algorithms are described in the following sections. First, the algorithm for the interaction between components in manipulation, assembly, and disassembly processes is presented. It extends the object-oriented modeling languages of system simulation tools. Then, a compliant contact force algorithm suitable for manipulation and assembly is selected. Next, an algorithm for dynamic structural changes during runtime in structure-invariant system simulation tools based on compliant contact forces is described. An additional algorithm for the robust handling of hierarchical contacts for the stable and efficient simulation of assemblies is presented. Finally, the topic of switching between contact models based on component models with multiple Levels of Detail is discussed. The main algorithms and models (representing the contributions) are:

  - The **Algorithm for Component Interaction (ACI)** is shown in Section 3.2.5. It handles the interactions between components in manipulation and assembly processes for both the substitute models and the contact force models.
  - The **Substitute Force Algorithm (SFA)** introduced in Section 3.4 allows components to be connected or disconnected during runtime in structure-invariant environments. It is based on compliant contact forces that hold the components in place.
  - The **Assembly Graph Partner Search Algorithm (AGPSA)** is presented in Section 3.5. It extends the SFA for the robust handling of hierarchical contacts and allows the stable and efficient simulation of assemblies.
  - The switching between the contact force algorithm and the SFA is addressed in Section 3.6. Here, **component models** with multiple **Levels of Detail (LsoD)** are used to improve the flexibility and to reduce the modeling effort.

- In **Chapter 4**, the realization of the solution developed in Chapter 3 is presented. The object-oriented, multi-domain modeling language *Modelica* is used in the system simulation environment *Dymola* and extended by an External Environment developed in this work. The integration of additional software packages and the interaction between *Modelica* models and the External Environment during simulation are discussed. In addition, optimizations to improve the computation time are introduced.

- Applications and their results are shown in **Chapter 5**. First, the simulation of the *Block Scenario* assembly process is shown. Based on this application, an additional analysis is performed, including a comparison between the contact models and a time analysis. In addition, a disassembly process is shown for this example and it is simulated with different robots with detailed models of the drivetrain. Another example is the assembly process of an electric motor into the housing of a chainsaw (see Figure 1.1). Furthermore, the multi-domain simulation of the disassembly process of an electric motor, including a furnace heating process, is presented.

- Finally, in **Chapter 6**, a conclusion is given and future developments are discussed.

2 State of the art

A comprehensive analysis of the current state of the art is presented. First, the basics of collision detection and response are discussed and selected contact models are shown. Subsequently, simulation environments are analyzed. This includes tools in general, simulating grasping processes in particular, body contacts in structure-invariant environments, and approaches for Variable-Structure Systems. The topic of Virtual Commissioning is also explored, including detailed physics simulations and models with multiple Levels of Detail. Finally, a summary and critique of the state of the art is given.

2.1 Collision detection and response

This section addresses collision detection and response in multibody systems. For this work, only rigid bodies are considered (no deformable shapes). There are also different possibilities for the representation of the body: polygonal models such as polygon soups and non-polygonal models like constructive solid geometry or parametric surfaces (J. Bender et al. 2012, p. 28). For this work, only polygonal representations are considered.

2.1.1 Collision detection

The basis for rigid body contacts in multibody scenarios is finding the collisions between all bodies. This is achieved by collision detection. Checking all possible collision pairs can be time consuming, especially if there are many bodies in a model. Therefore, the procedure of collision detection is often split into two phases (Ericson 2004, p. 14) or three phases (J. Bender et al. 2014, p. 248). Figure 2.1 shows the different phases.

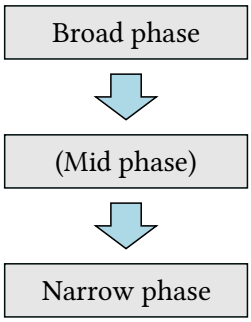

Figure 2.1: The collision detection process is typically divided into distinct phases, with a typical division comprising two or three phases (own representation based on J. Bender et al. (2014, p. 248)).

In the **broad phase**, the bodies are replaced with primitives (J. Bender et al. 2014, p. 248). For primitives, the distances can be calculated very fast. The aim is to exclude as many body pairs from the detailed collision detection as possible (Mainzer 2015, p. 18). These primitives are also known as Bounding Volumes (BVs) because they are chosen to completely encapsulate a body. Examples of BVs are Bounding Spheres, Oriented Bounding Boxes (OBBs) and Axis-Aligned Bounding Boxes (AABBs) (van den Bergen 1997, p. 131). The latter are one of the most commonly used (Ericson 2004, p. 77).
For complex bodies, Bounding Volume Hierarchies (BVHs) can be used to further reduce the number of computationally intensive collision calculations. This step is sometimes referred to as the **middle phase** (J. Bender et al. 2014, p. 248). The generation of a BVH starts with the creation of a BV for the entire body. This BV is then recursively divided into two smaller BVs. For example, if the BV is a box, it is halved in the direction of the longest edge. The procedure ends when the BVs encapsulate a predefined number of polygons (Mainzer 2015, p. 22). This can significantly reduce the number of collision calculations, especially for bodies consisting of many parts of which only a few are in collision.
In the **narrow phase**, detailed collision checks are performed based on the polygonal representation of the bodies (J. Bender et al. 2014, p. 248). Only the pairs that have passed the broad phase are checked for collision. When using BVHs, collision checks are only performed on those parts of a body whose BVs are in collision. The collision checks in the narrow phase are performed using collision detection algorithms.
There are several collision detection algorithms. They can be classified by the shape type as shown in Figure 2.2.

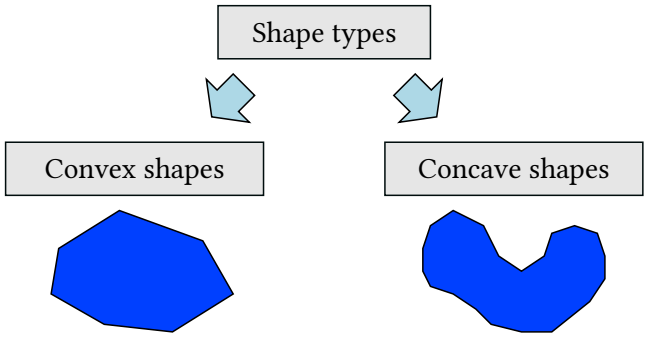

Figure 2.2: Classification of shape types for collision detection algorithms.

For **convex shapes**, there are two common algorithms.
The *Gilbert-Johnson-Keerthi (GJK)* distance algorithm (Gilbert et al. 1988) is widely used in collision detection, for example in computer graphics. It calculates the minimum distance between two convex shapes (J. Bender et al. 2012, p. 29). If the bodies intersect, the *Expanding Polytope Algorithm (EPA)* (van den Bergen 2004, p. 148) can be used to calculate the penetration depth (J. Bender et al. 2012, p. 29). In addition, the GJK algorithm can calculate the

related points for the shortest distance on both bodies if the bodies are not in contact.
The *Minkowski Portal Refinement (MPR)* algorithm (Snethen 2008) is similar to the GJK algorithm and also restricted to convex shapes. In constrast to GJK, MPR does not calculate the distance between two bodies if they do not intersect. However, MPR does provide the penetration depth. MPR is simpler and more robust than GJK and therefore easier to implement (Snethen 2008, p. 178). In Neumayr and Otter (2017), the MPR algorithm is improved so that the minimum distance between two non-colliding bodies can be calculated.
The open source project *libccd* provides an implementation of both the MPR algorithm and the GJK algorithm (together with the EPA) for collision detection (Fiser 2018).
Algorithms for collision detection between convex shapes, such as the GJK and MPR algorithms, can be used for non-convex shapes by using convex decomposition. Here, a concave shape is split into several convex shapes that replicate it as closely as possible. Figure 2.3 shows an example of a convex decomposition. The open source library *V-HACD* can be used to generate the convex decomposition of a concave shape (Mammou 2016). The aim of the library is not to generate the exact decomposition of a body, but an approximation. Collision detection using the GJK or MPR algorithm and the convex decomposition of a concave body is usually faster than using an algorithm for concave bodies.

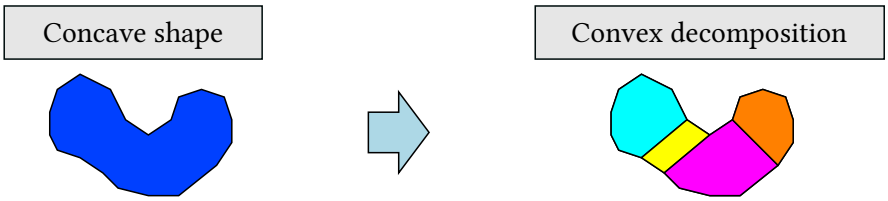

Figure 2.3: Convex decomposition of a concave shape. The shape is split into several convex shapes. This allows collision algorithms such as GJK or MPR to be used.

For **concave shapes**, there are also collision detection algorithms.
The *Polygonal Contact Model (PCM)* algorithm (Hippmann 2004b) is primarily used to calculate contact forces between rigid bodies. However, it contains a collision detection algorithm suitable for concave bodies. The surfaces of the bodies are represented by polygons. BVHs are used to reduce the number of computationally expensive intersection checks. In many applications, only parts of a bodies surface are in contact.
The improved *Voxelmap Pointshell Algorithm (VPA)* presented in Sagardia et al. (2014) can be used both for collision detection and contact force calculation. It is based on Signed Distance Fields (SDFs) and point-sphere trees. One body is discretized by a voxel grid. Each voxel contains the distance from the body surface. For the second body of the pair, the surface is represented with points in the same resolution as the voxel grid. The intersection check for a point against a voxel is very fast.

2.1.2 Collision response

The objective of collision response is to calculate the changes in motion of the bodies in question in order to prevent them from intersecting. In the work of Flores (2022, p. 136), two general approaches to address this problem are identified:

- Compliant (regularized, penalty-based)
- Non-smooth (constraint-based, complementary)

The **compliant approaches** are based on penetrations between rigid bodies. They are also referred to as "regularized" and "penalty-based" methods in literature. The bodies are considered rigid but have a deformable contact zone (Flores 2022, p. 136). Contact forces are used to separate the bodies. These contact forces "*are expressed as continuous functions of penetrations between contacting bodies*" (Flores and Lankarani 2016, p. 3).
Purely elastic models for the contact force such as Hertz (1882) are not suitable as they do not take energy loss into account (Machado et al. 2012, p. 107). A basic model considering energy dissipation is the Kelvin-Voigt contact force model (Flores 2022, p. 144). The normal force f_N is calculated using a spring-damper element:

$$f_N = k \cdot \delta + d \cdot \dot{\delta} \tag{2.1}$$

with δ being the penetration. The parameter k represents the spring stiffness and d the damping factor. Both are depending on the material parameters. Despite its simplicity, the Kelvin-Voigt model has been used in numerous academic works (Machado et al. 2012, p. 108). Multiple models have been developed since Kelvin-Voigt. A good overview of dissipative contact force models is given in Flores and Lankarani (2016, p. 44).
In addition to the normal force, a friction force occurs in tangential direction. A common friction model is Coulomb's law. It can be simplified as follows (Andersson et al. 2007, p. 3):

$$f_F = \mu \cdot f_N \cdot \text{sign}(v) \tag{2.2}$$

with μ being the coefficient of friction and v the tangential velocity.
Compliant approaches usually use a regularized friction model. The reason is that these models are differentiable (in comparison to the sign() function). One example is to use the tangens hyperbolicus (Andersson et al. 2007, p. 4):

$$f_F = \mu \cdot f_N \cdot \tanh(k_{tanh} \cdot v). \tag{2.3}$$

The parameter k_{tanh} determines the stiffness of the tangens hyperbolicus.
The advantages of the compliant methods are that they are easy to implement and efficient. The determination of appropriate parameters is a fundamental disadvantage of the method. In addition, stiff contacts in a model can lead to stiff differential equations and high-frequency vibrations, which in turn require small integration steps. This can significantly increase the computation time for a simulation (Flores and Lankarani 2016, p. 3).

In the **non-smooth approaches** (also referred to as "constraint-based" and "complementarity"), the bodies are considered rigid without elastic discretization in the contact zone (F. Pfeiffer 2008, p. 533). Here, unilateral constraints are used to prevent bodies from intersecting (Flores 2022, p. 136). Unlike in the compliant approaches, in the non-smooth approaches velocity jumps can occur, for example caused by impacts, and special solvers are required (Clauberg 2013, p. 7). A basis for this approach is the complementarity rule. *"It states that in contact dynamics either relative kinematic quantities are zero and the accompanying constraint forces [...] are not zero, or vice versa"* (F. Pfeiffer 2008, p. 533).
For a multibody system with unilateral contacts, the equations of motion can be described by (F. Pfeiffer 2008, p. 535) (F. Pfeiffer and Glocker 2004, p. 25):

$$\boldsymbol{M}(q,t)\ddot{\boldsymbol{q}} - \boldsymbol{h}(q,\dot{q},t) = \boldsymbol{0} \tag{2.4}$$

with the generalized coordinates $\boldsymbol{q}$, the mass matrix $\boldsymbol{M}$, and the vector of external forces $\boldsymbol{h}$.
The related relative velocities and accelerations are (F. Pfeiffer and Glocker 2004, p. 39-40):

$$\dot{g}_N = \boldsymbol{w}_N^{\mathrm{T}}\dot{\boldsymbol{q}} + \tilde{w}_N, \quad \ddot{g}_N = \boldsymbol{w}_N^{\mathrm{T}}\ddot{\boldsymbol{q}} + \bar{w}_N \tag{2.5}$$

$$\dot{g}_T = \boldsymbol{w}_T^{\mathrm{T}}\dot{\boldsymbol{q}} + \tilde{w}_T, \quad \ddot{g}_T = \boldsymbol{w}_T^{\mathrm{T}}\ddot{\boldsymbol{q}} + \bar{w}_T \tag{2.6}$$

and depend on the constraints from relative kinematics, defined by the constraint vectors $\boldsymbol{w}_{N/T}$ and their magnitudes $\tilde{w}_{N/T}$ (and the magnitudes of their deviations $\bar{w}_{N/T}$).
These relations must be completed with complementaries. For the normal direction, the complementaries are based on the normal force $\boldsymbol{\lambda}_N$ (F. Pfeiffer 2008, p. 535):

$$\ddot{\boldsymbol{g}}_N \geq 0, \quad \boldsymbol{\lambda}_N \geq 0, \quad \ddot{\boldsymbol{g}}_N^{\mathrm{T}}\boldsymbol{\lambda}_N \geq 0. \tag{2.7}$$

The following conditions represent friction based on the Coulomb friction law and apply to the tangential direction (Clauberg 2013, p. 12):

$$\ddot{\boldsymbol{g}}_T = 0 \Rightarrow |\boldsymbol{\lambda}_T| \leq \mu|\lambda_N| \tag{2.8}$$

$$\ddot{\boldsymbol{g}}_T \neq 0 \Rightarrow \boldsymbol{\lambda}_T = -\frac{\dot{\boldsymbol{g}}_T}{|\dot{\boldsymbol{g}}_T|}\mu|\lambda_N| \tag{2.9}$$

with μ being the coefficient of friction and $\boldsymbol{\lambda}_T$ the tangential force.
Taking into account the geometric constraints defined above, the equations of motion can be described by (Flores 2022, p. 147):

$$\boldsymbol{M}\dot{\boldsymbol{u}} - \boldsymbol{h} - \boldsymbol{w}_N\boldsymbol{\lambda}_N - \boldsymbol{w}_T\boldsymbol{\lambda}_T = \boldsymbol{0} \tag{2.10}$$

$$\dot{\boldsymbol{q}} = \boldsymbol{u} \tag{2.11}$$

at the acceleration level, with $\boldsymbol{q}$ being the generalized coordinates, $\boldsymbol{M}$ the mass matrix, $\boldsymbol{h}$ the vector of external forces, $\boldsymbol{w_N}$ and $\boldsymbol{w_T}$ the directions of the normal and tangential force, and $\boldsymbol{\lambda_N}$ and $\boldsymbol{\lambda_T}$ the normal and tangential force.

Now, the impacts between rigid bodies are considered. The impacts are assumed to be infinitesimally short in time and the position and orientation of the bodies does not change during the impact (F. Pfeiffer 2008, p. 536). Only the impulse forces can change during the impact (F. Pfeiffer 2008, p. 536). Hence, the equations of motion are expressed on the velocity level by (Flores 2022, p. 148) (F. Pfeiffer 2008, p. 536):

$$\boldsymbol{M}(\boldsymbol{u}^{+} - \boldsymbol{u}^{-}) - \boldsymbol{w}_N \boldsymbol{\Lambda}_N - \boldsymbol{w}_T \boldsymbol{\Lambda}_T = \boldsymbol{0} \tag{2.12}$$

with $\boldsymbol{u}^{+}$ and $\boldsymbol{u}^{-}$ being the velocities before and after the impact and $\boldsymbol{\Lambda}_N$ and $\boldsymbol{\Lambda}_T$ being the impulses in the normal and tangential direction. Both equations can be combined using $\mathrm{d}t$ as the Lebesgue measure for the continuous parts and $\mathrm{d}\eta$ as the Dirac point measure for the impacts (Flores 2022, p. 148-149) (F. Pfeiffer 2008, p. 538):

$$\boldsymbol{M}\mathrm{d}\boldsymbol{u} - \boldsymbol{h}\mathrm{d}t - \boldsymbol{w}_N \mathrm{d}\boldsymbol{P}_N - \boldsymbol{w}_T \mathrm{d}\boldsymbol{P}_T = \boldsymbol{0} \tag{2.13}$$

$$\mathrm{d}\boldsymbol{u} = \dot{\boldsymbol{u}}\mathrm{d}t + (\boldsymbol{u}^{+} - \boldsymbol{u}^{-})\mathrm{d}\eta \tag{2.14}$$

$$\mathrm{d}\boldsymbol{P} = \boldsymbol{\lambda}\mathrm{d}t + \boldsymbol{\Lambda}\mathrm{d}\eta \tag{2.15}$$

with the measure $\mathrm{d}\boldsymbol{u}$ for the velocities, based on the continuous part $\dot{\boldsymbol{u}}\mathrm{d}t$ and the discrete part defined by the left and right limit $\boldsymbol{u}^{-}$ and $\boldsymbol{u}^{+}$. Additionally, the measure $\mathrm{d}\boldsymbol{P}$ for the impact consists of a continuous part $\boldsymbol{\lambda}\mathrm{d}t$ and a discrete part $\boldsymbol{\Lambda}\mathrm{d}\eta$. For free motion without impact, the discrete parts disappear and only the continuous parts remain (Flores 2022, p. 148). The system of equations can be represented by a Linear Complementary Problem (LCP).
In general, there are two simulation methods for non-smooth dynamics: event-driven and time-stepping. Event-driven schemes treat contact changes as events. When an event occurs, integration pauses and the LCP is solved using algorithms such as Lemke's. Between events, classical ODE solvers compute the equations of motion. This approach is accurate but time-consuming and only suitable for scenarios with few contacts. (F. Pfeiffer 2008, p. 540)
In contrast, the time-stepping methods widely used today use a time discretization of the dynamics (Moreau 1988). The discretized equations of motion are combined with the contact constraints and the resulting LCP is solved for the next state of motion (F. Pfeiffer 2008, p. 540). Problems are usually solved for the next velocity (first order accuracy) (Drumwright and Trinkle 2017, p. 24). In these methods, multiple events in a time interval $[t_0, t_0 + \Delta t]$ *"are treated as if they occur at a single time"* (Drumwright and Trinkle 2017, p. 24).
The steps for solving non-smooth dynamics using the time-stepping method are demonstrated using the *MBSim*[1] software as an example (Clauberg 2013, p. 16-17):

1. Calculation of the new generalized coordinates
2. Calculation of the contact kinematics (contact distances, contact velocities)
3. Solving of the complementary problem

[1] `https://www.mbsim-env.de/`

Non-smooth methods do not have the numerical problems of compliant methods. They are more stable and robust and allow large integration steps, resulting in fast simulations. However, in some cases multiple solutions can occur because of an undetermined system (Flores 2022, p. 137-138). Another disadvantage is drift because velocity solvers do not recognize position errors. One solution is to use additional forces or constraints. New approaches such as position based dynamics solve this by using positions directly (Müller et al. 2020, p. 2).

2.2 Contact models

There are many contact models in the literature. A good overview of existing contact models can be found in Flores and Lankarani (2016) and Corral et al. (2021). These works focus on contact force models for the compliant approach. In addition, Flores (2022) shows recent developments in the area of contact mechanics. For non-smooth methods, J. Bender et al. (2014) give a comprehensive overview of common contact models and solver techniques.
Hereafter, the Elastic Foundation Model is introduced. It serves as the theoretical basis for area contact models. Based on this, selected area contact models are presented and analyzed.

2.2.1 Elastic foundation model

The *Elastic Foundation Model (EFM)* (Johnson 1985, p. 104) is an approximation of an elastic contact. An ideal elastic contact model must take into account the pressure distribution on the contact, which determines the penetration of all points on the contact surface. However, it is complex to calculate this pressure distribution. The idea of the EFM is to consider bodies rigid but cover them with a thin elastic layer, also referred to as "elastic foundation", with the layer thickness b (see Figure 2.4). The model provides a good balance between computational effort and physical accuracy (Hippmann 2004a, p. 347).

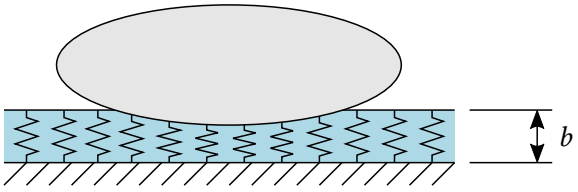

Figure 2.4: Graphical representation of the Elastic Foundation Model. The elastic layer with the layer thickness b is shown in blue. The grey shape is the second of the two bodies in contact. Sheer is ignored, i.e. the springs do not interact. This figure is adapted from Johnson (1985, p. 104).

The pressure p at a point on the contact surface depends only on the displacement at this point u and the elastic modulus K of the layer (Johnson 1985, p. 105):

$$p = \frac{K}{b} \cdot u. \tag{2.16}$$

2.2.2 Polygonal contact model

The *Polygonal Contact Model (PCM)* (Hippmann 2004b) is a compliant contact algorithm for complex, non-convex shapes. It is based on the Elastic Foundation Model and requires a polygonal mesh representation for the surfaces of the bodies. The PCM can be included in multibody simulations and behaves like a force element (Hippmann 2004a, p. 348).
For the contact detection of PCM, one body of the contact pair is defined as *Base* and the other one as *Target* (see Figure 2.5). The PCM algorithm performs preprocessing for all defined contact pairs. This includes the import of the mesh and the generation of data structures. For PCM, the surface of a body must be represented by triangles. Polygons with more than three edges are automatically split into triangles.

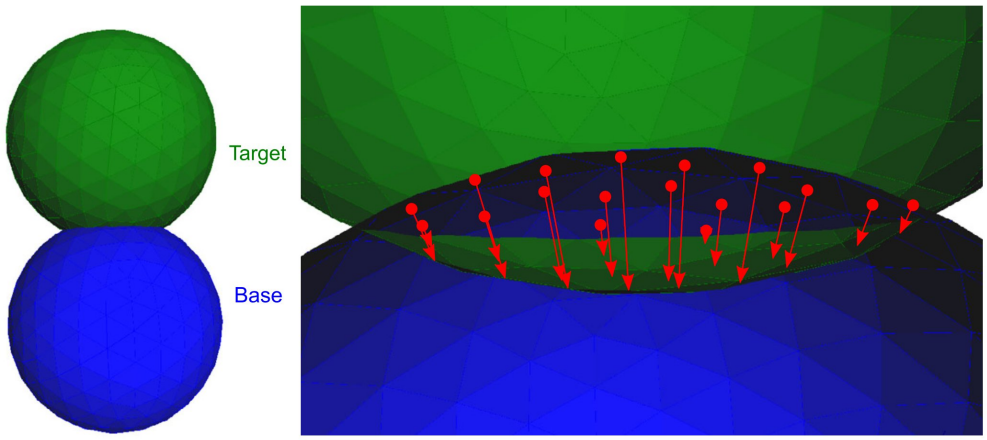

Figure 2.5: Graphical representation of contact elements in the Polygonal Contact Model. Bodies are represented by polygons. One body is defined as *Base*, the other one as *Target*. This figure is from Hippmann (2024, p. 220).

The PCM algorithm consists of the following steps (Hippmann 2004a, p. 348):

1. Collision detection, including BVHs, to determine which triangles of the *Base* body are in contact with the *Target* body (defined as contact elements)
2. Construction of the intersecting surface area (intersection polygon)
3. Determination of the involved triangles for the *Base* body

For these triangles, contact elements are generated (Hippmann 2024, p. 220):

4. Construction of a penetration line towards the inside of the body for these triangles
5. The penetration depth is the distance from the beginning of the penetration line to its intersection with the surface of the *Target* body
6. Calculation of the normal force and the friction force for each contact element

7. Generation of the resulting force for the body (sum of the forces of all contact elements)

The PCM contact algorithm has recently been improved. A new method called *penetration detection* is presented in Hippmann (2024). This new technique generates the penetration lines in the preprocessing and uses them instead of the triangles for the BVH collision tests in step 1. As a consequence, steps 2, 3, and 4 are not longer required. After the collision check based on BVHs, the penetration depths of the involved penetration lines can be calculated. The other steps remain unchanged. This enhanced version of the PCM algorithm is simpler. It is more robust regarding the quality of the polygonal representation. And it has a higher efficiency in terms of computation time (Hippmann 2024, p. 229).
The contact force of the PCM algorithm is made up of a normal force and a frictional force in the tangential direction. The former is made up of an elastic force and a damping force. The basis for the elastic force is the pressure at a contact point depending on the displacement, introduced in Equation (2.16). The elastic modulus for linear elastic layers is given by:

$$K = \frac{1-\nu}{(1+\nu)(1-2\nu)} \cdot E \tag{2.17}$$

with the Young's modulus E and the Poisson's ratio ν (Hippmann 2004a, p. 347). The elastic part of the normal force of a contact element k, f_{ck}, can then be calculated by:

$$f_{ck} = \frac{K}{b} \cdot A_k \cdot u_{nk} \tag{2.18}$$

with the area A_k, the displacement of the contact point u_{nk}, and the layer thickness b (see Figure 2.4). The pressure is assumed to be constant within a triangular contact element. Same applies to the displacement in normal direction (Hippmann 2004a, p. 347). In a similar way, a damping force is calculated by (Hippmann 2004a, p. 356):

$$f_{dk} = \begin{cases} d \cdot A_k \cdot \dot{u}_{nk} & \text{if} \quad u_{nk} \geq u_d \\ d \cdot A_k \cdot \dot{u}_{nk} \cdot \frac{u_{nk}}{u_d} & \text{if} \quad u_{nk} < u_d \end{cases} \tag{2.19}$$

with the damping factor d. A linear transition for the damping force for penetrations smaller than u_d is used to avoid discontinuous contact forces at the start and end of a collision. Both the elastic and the damping part of the normal force are added in the following way to avoid non-physical tension behavior (Hippmann 2004a, p. 356):

$$f_{nk} = \begin{cases} f_{ck} + f_{dk} & \text{if} \quad f_{ck} + f_{dk} > 0 \\ 0 & \text{if} \quad f_{ck} + f_{dk} \leq 0. \end{cases} \tag{2.20}$$

The friction force in tangential direction is calculated using a regularized friction model:

$$f_{tk} = \begin{cases} \mu \cdot f_{nk} & \text{if} \quad v_{tk} \geq v_\epsilon \\ \mu \cdot f_{nk} \cdot \frac{v_{tk}}{v_\epsilon}(2 - \frac{v_{tk}}{v_\epsilon}) & \text{if} \quad v_{tk} < v_\epsilon \end{cases} \tag{2.21}$$

with the friction coefficient μ and the velocity in tangential direction v_{tk}. For velocities smaller than v_ϵ, the friction force decreases quadratically (Hippmann 2004a, p. 357). This is an approximated friction model similar to the one in Equation (2.3).

2.2.3 Voxelmap pointshell algorithm

The *Voxelmap Pointshell Algorithm (VPA)* from Sagardia et al. (2014) can be used to calculate contact forces between complex, non-convex shapes. It is an improvement to the algorithm shown in McNeely et al. (2005). The concept is to discretize one body by a voxel grid, where each voxel contains the distance to the surface. This is also called a Signed Distance Field (SDF). The surface of the second body is represented by points. If a point intersects with a voxel on the surface, this voxel is bisected by a tangent plane. The penetration depth is the distance between the point and the plane if the point is below the plane, otherwise it is zero. Based on the penetration depth, a compliant contact force model is applied. As the collision check between a point and a voxel is very fast, the algorithm is very performant. An example for the algorithm is shown in Figure 2.6.

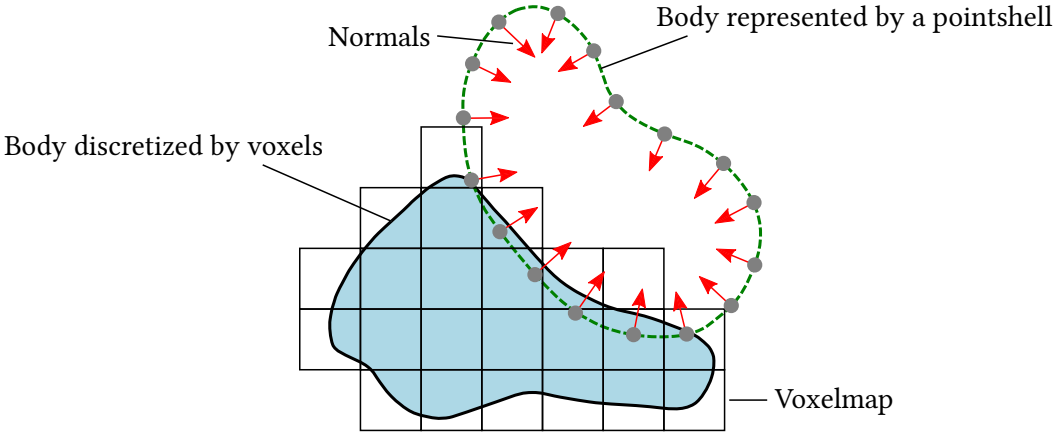

Figure 2.6: An example for the Voxelmap Pointshell Algorithm. One body represented by voxels collides with another body represented by points. The grid represents the voxelmap and the red arrows represent the normals for the point shell. This figure is adapted from McNeely et al. (2005, p. 43).

2.2.4 Pressure field contact model

The *Pressure Field Contact (PFC)* model (Elandt et al. 2019) is an approximate model for the contact force between complex, non-convex shapes. This method combines the Elastic Foundation Model with hydrostatic pressure. In a preprocessing step, pressure fields are computed for all bodies. These pressure fields are continuous, with the pressure increasing towards the inside of the body. An exception to this are ideal rigid bodies. Here the pressure field jumps from zero to infinity. These pressure fields do not represent the real pressure. Therefore, no equilibrium condition in mechanics has to be fulfilled.
When two bodies collide, the contact surface is calculated. Both bodies have a pressure field. The surface where the pressure is equal is the contact surface. It can consist of several

unconnected surfaces for non-convex bodies. Figure 2.7 shows the contact surface for a sphere colliding with a plane. BVHs are used for the broad phase of the collision detection. The contact surface is discretized with triangles. For the calculation of the resulting contact force, the pressure effects of all triangles are integrated over the entire contact surface.
The PFC model has been adapted to non-smooth physics engines (Masterjohn et al. 2022) by using an approximation at the velocity level.

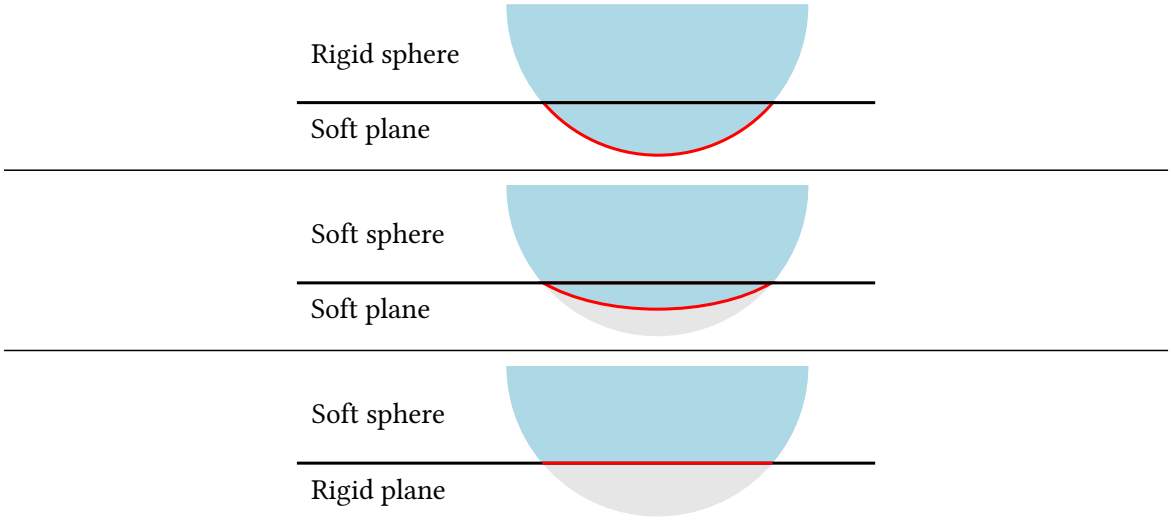

Figure 2.7: Contact between a sphere and a plane based on the *Pressure Field Contact* model. The contact surface for all combinations of soft and rigid bodies is marked in red. If both bodies are soft, the contact surface is the area where the pressure is equal. This figure is adapted from Elandt et al. (2019, p. 8240).

2.3 Simulation environments

In this section, simulation environments are analyzed. An overview of common simulators and their classification is given. Tools and works for the simulation of grasping processes are also presented. In addition, works on body contacts in system simulation environments and approaches for Variable-Structure Systems are discussed.

2.3.1 Overview and classification

There are many simulation environments with different objectives. For the purpose of this work, a selection of common simulators is classified according to their application domain. These include robotics and automation, (multi-domain) system simulation, multibody simulation, and manufacturing simulation. Physics engines are another class, as they are usually integrated into simulation environments to calculate the (contact) dynamics.

First, well-known **physics engines** are introduced below.
One well-known engine is *Bullet* (Coumans 2024). The library offers rigid-body dynamics in real-time based on the non-smooth collision response approach. The *Open Dynamics Engine* (Smith 2024) is similar to Bullet. Both engines are widely used for physics simulations in games and simulation environments. The aim of the *MuJoCo* (Todorov et al. 2012) engine ("Multi-Joint dynamics with Contact") is robotics simulation. It uses a modified version of the non-smooth approach in which the complementary condition is relaxed and approximated (Todorov 2014). The *PhysX* engine (Nvidia 2024) is developed by Nvidia and used in games and simulators. It offers rigid and soft body dynamics, including cloth simulation. The engine offers hardware acceleration, which is largely limited to Nvidia graphics cards. *Project Chrono* (Tasora et al. 2016) was originally developed as a multibody system simulator. Today it is more of a physics engine. Like PhysX, it can be used for rigid and soft body dynamics. However, the focus is on vehicles interacting with deformable terrain and the interaction between solids and fluids. All of these engines are open source and use the non-smooth approach for the collision response. PhysX and Project Chrono also include a compliant implementation. Table 2.1 gives an overview of the engines mentioned.

Table 2.1: Overview of common physics engines.

Physics engine	License	Collision response approach
Bullet	Open source	Non-smooth
ODE	Open source	Non-smooth
MuJoCo	Open source	Non-smooth
PhysX	Open source	Non-smooth and compliant
Project Chrono	Open source	Non-smooth and compliant

Various well-known simulation environments are now presented according to their application domain. The first application domain is **robotics and automation**.
A common simulation tool is *CoppeliaSim* (formerly known as V-REP) (Rohmer et al. 2013). It offers versatile capabilities for simulating robots, including forward and inverse kinematics, sensor simulation, and path planning. Various physics engines can be used for the multibody dynamics, including Bullet, Open Dynamics Engine, and MuJoCo. Another robotics simulator is *Gazebo* (Koenig and Howard 2004). It provides an extensive collection of examples. Bullet and the Open Dynamics Engine are integrated. The focus of *Drake* (Tedrake and Drake Development Team 2019) is the simulation of detailed robot dynamics with the aim of control development. It uses its own engine and comes with a framework for optimization. In contrast to CoppeliaSim, Gazebo and Drake are open source.
Another application domain is **system simulation**. Here, the aim is the simulation of multiple domains such as mechanical, electrical, fluid, and thermal within one model.

Modelica is an object-oriented and free modeling language (Modelica Association 2017). It provides multi-domain modeling in a component-oriented and acausal manner. This allows the modeling of complex systems consisting of multiple components. Modelica models are highly flexible because they allow full access to the equations that describe the physical behavior. There are several open and commercial libraries that can be used to create models. An example is the free and open source *Modelica Standard Library (MSL)* (Modelica Association 2020). It contains model components from multiple domains, including mechanical multibody elements (Otter et al. 2003). An example of a commercial library is the *DLR Visualization 2 Library* (Kümper et al. 2021). There are several tools for using Modelica, including the commercial tools *Dymola* (Dassault Systèmes 2025) and *SimulationX* (ESI Group 2024), and the open source software *OpenModelica* (Fritzson et al. 2020). More details on Modelica and Modelica libraries are provided in Section A.2.1 and Section A.2.2.
Simscape is an object-oriented extension to Simulink that can be used within the MATLAB/Simulink environment (MathWorks 2024). It allows the acausal modeling of components from multiple domains similar to Modelica. However, there are fewer libraries available for Simscape than for Modelica. Studies have shown a higher computational performance of Modelica over Simscape when it comes to multi-domain models (De Marco 2023).
There are also specialized tools for **multibody simulation**.
Simpack (Rulka 1990) is a commercial multibody simulation software. It allows the development of complex models with a high Level of Detail and provides a fast and robust solver. The software can be used to model nonlinear system behavior and to perform frequency analysis. *ADAMS* (Ryan 1990) is another commercial tool for multibody simulation. It is similar to Simpack, but differs in the concepts and solvers used for the model description. Both are used in a variety of domains, including the automotive, railway, and wind energy industries.
Another domain that uses specialized tools is **manufacturing simulation**.
DELMIA (Dassault Systèmes 2024) focuses on process planning and logistics (supply chain management). Among other things, it uses material flow simulations. *ISG-virtuos* (ISG Industrielle Steuerungstechnik 2024) aims at the validation of automation software with Virtual Commissioning (see Section 2.4). It uses real-time simulations such as Software in the Loop (SiL) and Hardware in the Loop (HiL), as well as other machine simulations, including fieldbuses. An overview of the mentioned simulation environments classified according to their application domain is shown in Table 2.2.

2.3.2 Simulation of grasping processes

This section provides an analysis of tools and works that focus specifically on the simulation of grasping processes and their application to robotic manipulation.
GraspIt! (Miller and Allen 2004) is a simulation environment for robotic grasping. The framework can be used for the planning, analysis, and optimization of grasping processes. It provides a robot library with multiple models of robots, hands, and grippers. Collision detection uses BVHs and AABBs. When two bodies come within a predefined contact threshold, all contact points are identified and the resulting contact forces are calculated.

Table 2.2: Overview of simulation environments classified by the application domain.

Application domain	Simulation environment	License
Robotics and Automation	CoppeliaSim Gazebo Drake	Commercial Open source Open source
System simulation	Modelica Simscape	Commercial and open source Commercial
Multibody simulation	Simpack ADAMS	Commercial Commercial
Manufacturing simulation	DELMIA ISG-virtuos	Commercial Commercial

OpenGRASP (León et al. 2010) is another simulation tool for grasping processes. The toolkit is designed for extensibility and interoperability. It is based on the *OpenRAVE* framework (Diankow 2010), enhanced with a robot editor and plugins for sensors and actors. A physics abstraction layer was developed to use physics engines other than the default Open Dynamics Engine in OpenRave. Both tools use a non-smooth approach at the velocity level for the collision response and are open source projects.
The *RobWorkPhysicsEngine* (Thulesen 2016) is a physics engine focused on robotic manipulation and assembly for scenarios with a tight fitting. It is part of the *RobWork* framework (Ellekilde and Jorgensen 2010). The RobWorkPhysicsEngine extends the first order non-smooth collision response approach with a second order integration method for the position update (Thulesen and Petersen 2016, p. 2062). As a result, the accuracy is higher and thin contact layers required for assembly scenarios can be used (Thulesen and Petersen 2016, p. 2062). An improved model is used to handle multiple contacts. The LCP used in non-smooth solvers is replaced with a convex optimization problem that distributes the force across multiple contacts instead of *"randomly select one of the contacts to have a positive contact force and the other contact forces to be zero."* (Thulesen and Petersen 2016, p. 2063).

2.3.3 Body contacts in system simulation environments

Manipulation and assembly processes can be simulated not only in specialized simulation tools. System simulation environments are suitable for detailed simulations, such as analyzing the joint torques in the robot during an assembly process. However, rigid body contacts are required for modeling. This section discusses the contact modeling in the system simulation environments identified in Section 2.3.1: Modelica and Simscape. Neither environment provides the ability to model complex, non-convex contacts out of the box.
For **Modelica**, several publications deal with collision detection and response as mentioned

in Reiser and Reiner (2023). In a number of works, Modelica components, e.g. from the Modelica Standard Library (MSL), are coupled with external libraries to extend their capabilities. They all use the compliant approach to calculate the contact forces.
In Otter et al. (2005), the multibody part of the MSL is extended with centralized collision handling. Several surface descriptions are tested and compared. These include parametric surfaces, surfaces based on algebraic constraints, and polygonal surfaces. For the latter, the *SOLID* collision detection library (van den Bergen 2004, p. 219-249) is used to calculate the body distances or penetration depths based on the GJK algorithm (together with the EPA). Another extension to the MSL was built in Hofmann et al. (2014). They use the collision detection capability of the *Bullet* physics engine to calculate penetration depths. In both works, a unique ID for each body must be manually defined by the user, as the Modelica language does not yet provide this capability (Hellerer and Buse 2017).
In a similar way, Buse et al. (2023) extend the multibody elements of the MSL with an external contact calculation for hard body contacts and soft soil contacts. For hard contacts, the *libccd* with the MPR algorithm is used to determine collisions between convex bodies. In all three works, each contact pair is represented by a single point contact with related penetration depth. In the work of Elmqvist et al. (2015), a framework for the body interaction based on the Discrete Element Method is introduced. The framework extends the multibody components of the MSL. Bodies are represented by a polygonal surface. They use AABBs for the broad phase and calculate the intersection volume in the narrow phase. Based on this volume, the contact force is calculated. Both Buse et al. (2023) and Elmqvist et al. (2015) are more user friendly, because they do not require manually defined unique IDs.
There are also native Modelica approaches. The library developed in Oestersötebier et al. (2014) provides idealized contacts for elementary geometries based on contact surfaces. They simplified the Elastic Foundation Model (see Section 2.2.1) and use single force elements based on contact points with penetration depths. Another compliant approach is the event-driven contact model from Bortoff (2020). A finite state machine is used to represent the transition between the contact states. The focus of this model is the dynamics analysis for control algorithm development. Both works require predefined contact pairs.
Another possibility is the integration of Modelica models into other simulation environments. The framework developed in Bardaro et al. (2017) extends the *Gazebo* simulator with Modelica multibody contact dynamics. The idea is to create detailed, equation-based contact models for selected body pairs in Modelica and integrate them into the Gazebo environment. These models use the collision detection capability of Gazebo to calculate the penetration depths, but replace the contact forces with those from the Modelica model.
In **Simscape** there are also works that use contacts between bodies. In Pozzi et al. (2022), a robotic grasping process is modeled in the MATLAB/Simulink environment using Simscape. A simulation of a walking excavator chassis based on Simscape is presented in Hu and Zhao (2024). Both works use the *"Spatial_Contact_Force"* block from the Simscape *Multibody* library to model the contact between the gripper and the grasped body. This block can be used to model the contact between two convex bodies (MathWorks 2023). Each body pair must be predefined in the model by the user.

2.3.4 Approaches for variable-structure systems

Variable-Structure Systems (VSS) (i.e. models with equations that change during the simulation runtime) are another way to simulate manipulation and assembly processes. Equation-based modeling languages such as Modelica are well-suited for system simulation. Their declarative approach allows the creation of complex models based on component libraries. However, once an equation-based model is created, the equations cannot be changed during the simulation runtime. Therefore, these modeling environments typically cannot be used to model VSS. This circumstance applies to most of the simulation environments, not just the equation-based ones.
There has been some work on how to model VSS in equation-based languages.
In Zimmer (2010), the modeling language *Sol* is proposed as a modification of Modelica. That is, a component model can contain several model variants. For example, the model of a machine can contain two models with different Levels of Detail. An interpreter was developed to handle the equations dynamically during runtime.
A Battery Electric Vehicle (BEV) model based on the concept of VSS is developed in Stüber (2017). The aim is to simulate the battery aging. Both a detailed and a simplified model are created for the BEV. The former is used during driving and the latter during idling. The result is a higher computation time for the simulation based on VSS due to the overhead for switching between models.
A different approach is introduced in Tinnerholm et al. (2022). They developed a Modelica compiler in the *Julia* programming language (Bezanson et al. 2017). Modelica is extended with new operators to enable VSS. They propose explicit VSS, where the transition is defined by the user and all equations are known before the simulation starts. In this case, a possibly large model containing all variants is compiled. Another possibility are implicit VSS, where predefined events trigger transitions that lead to re-compilation at runtime.
In Neumayr and Otter (2023), another extension for equation-based languages is presented to allow for VSS. In their approach, variables in a model can be added or removed during the simulation runtime (due to changes in the number of equations or states) without the need to re-compile the code. An implementation can be found in the open source modeling language *Modia*[2] and its multibody extension *Modia3D*[3] (Neumayr 2025).
A solution for VSS in Modelica based on dynamically constrained objects is introduced in Reiser and Reiner (2023). The work consists of collision detection and constraining bodies with forces and is implemented natively in Modelica. It is based on the Constraint Force Equation method developed at NASA (Toniolo et al. 2008). A constraint network allows a stable and precise simulation of VSS in the multibody domain (e.g. robot tool changers). The approach is limited to the elementary geometries sphere and rectangle surface and unique IDs must be defined by the user.

[2] `https://github.com/ModiaSim/Modia.jl`
[3] `https://github.com/ModiaSim/Modia3D.jl`

2.4 Virtual commissioning

This section provides an overview of Virtual Commissioning (VC). VC is the method of testing control software in a simulation model of a production system before it is used in the real plant (Wünsch 2008). The aim is to detect errors in the automation software at an early stage, as these are responsible for the majority of delays in commissioning (Wünsch 2008, p. 2). This can both reduce commissioning time and improve software quality (Reinhart and Wünsch 2007), but comes with the cost of model creation (Striffler and Voigt 2023, p. 678). The most common approaches for VC are (Lechler et al. 2019; Striffler and Voigt 2023):

- In HiL, the automation software runs on a real hardware controller, connected to the simulation model of the plant.
- SiL is similar, but instead of a real hardware controller, an emulated software controller is used to run the software.
- The connection of an emulated controller with a real plant is called reality in the loop or hybrid commissioning.

An overview is provided in Figure 2.8.

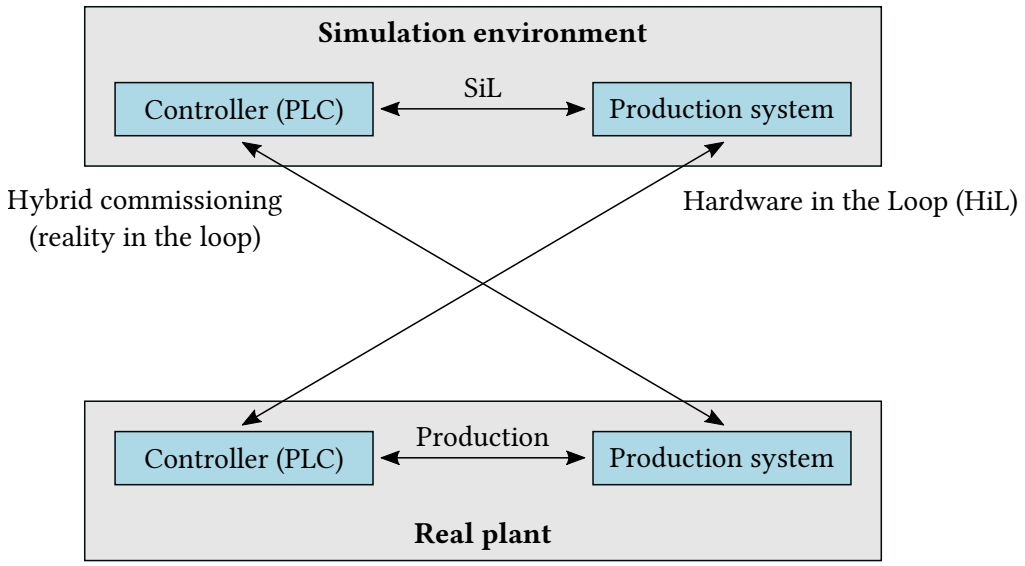

Figure 2.8: Types of Virtual Commissioning, including Hardware in the Loop (HiL), Software in the Loop (SiL), and hybrid commissioning (reality in the loop). This figure is adapted from Lechler et al. (2019, p. 1126).

A comprehensive review of current works in VC can be found in Lechler et al. (2019) and Striffler and Voigt (2023). For the purpose of this work, two aspects are analyzed in the following: detailed physics simulations and models with multiple Levels of Detail.

2.4.1 Detailed physics simulations

In this section, work is presented that used detailed physical behavior models for VC.
Lacour (2012) developed a procedure model for creating physics-based behavior models for VC in the material flow domain. The basis is a Computer-Aided Design (CAD) model that is first converted into a triangulated polygonal model. Then, a collision model is created by convex decomposition (see Section 2.1.1). Finally, the physical model is created by adding physical parameters and defining kinematic chains. The concept was implemented and evaluated as a software prototype. Physics engines with a non-smooth collision response approach are used, including *PhysX* (see Section 2.3.1).
The aim of the *AVANTI* project is to extend VC with physical behavior models (Gulan et al. 2014). A plant is described by a geometric model, where the components are classified into moving and fixed parts. Again, the basis is the CAD model and physical parameters are added to build the behavior model. The focus is on using standards such as *AutomationML* (Lüder and Schmidt 2024) for the models. Non-smooth physics engines are used for simulation.
Another work in the *AVANTI* project focused on the provision of models. Normally, the user of a production system creates behavior models for all parts of the plant. Süß et al. (2015) propose that manufacturers instead provide tested behavior models for their components. The Functional Mockup Interface (FMI) standard (Blochwitz et al. 2011) is used to ensure compatibility while protecting the manufacturer's intellectual property. The coupling of the component models to a system simulation is achieved by co-simulation.
In the work of Köslin et al. (2014), a Modelica library was developed that aims at the VC of material flow systems in general and conveyor technology in particular. Components were built for both bulk cargo and breakbulk cargo processes, as well as the conversion between the two. The library supports HiL simulations.
In the review of Lechler et al. (2019), use cases of VC for production systems are analyzed. In the manufacturing domain, they found one work that uses detailed models at the process level: Kübler et al. (2017) used physical behavior models for the VC of a run-to-run controller. These types of controllers are used to optimize processes. Here, one step in the manufacturing process for batteries (used in BEVs) is represented by a physical behavior simulation (HiL). The results of this simulation are used as input for the next process step. This allows the run-to-run controller to be tested in advance.

2.4.2 Models with multiple levels of detail

In this section, models with multiple Levels of Detail (LsoD) are discussed. In the context of this work, these models are referred to as hybrid models.
Frieben et al. (2015) developed a model of a material flow system with variable modeling depth. They distinguish between three LsoD: idealized function models (logical functionality in discrete-time), principle feasibility models (additionally with time behavior), and system-specific behavior models (detailed physical models, including effects such as friction). In addition, they developed a method for adaptive switching. Low-detail models are replaced

by detailed models when more information is needed. Switching conditions are specified when the model is created. In their use case, a workpiece is transported by a shuttle. Idealized models are used for straight travel. For cornering, detailed models are used to account for the effects of centrifugal forces. They used the library for idealized contacts from Oestersötebier et al. (2014) presented in Section 2.3.3.
In the work of Hoher (2016), a hybrid model was developed for the simulation of material flow systems consisting of two Levels of Detail. Typically, several problem-specific simulations with different objectives are developed during the design process of a plant. Therefore, the work aims at coupling two material flow models with different LsoD into one. As a result, the modeling effort can be reduced. Two models were developed. Both use a non-smooth approach. The impulses are calculated based on a kinetic collision response (detailed model) and a kinematic collision response (substitute model). The former provides accurate results, but does not guarantee real-time simulation. Substitute models, on the other hand, always run in real-time, but offer less detail.
Another approach is the definition of LsoD based on the hierarchical structure of production plants, as presented in Puntel-Schmidt and Fay (2015). Here, the following LsoD are defined with a focus on material flow systems. The *macroscopic* LoD covers time behavior based on dead time models. In constrast, the *microscopic* LoD includes detailed behavior models. Two intermediate LsoD are used if a more detailed distinction is necessary.

2.5 Summary and critique of the state of the art

Rigid body contacts in multibody models require collision detection and response (see Section 2.1). The former is usually split into a broad phase with BV checks such as OBB or AABB and a narrow phase with algorithms such as *GJK* or *MPR*. There are two basic methods for the latter. Compliant approaches (also referred to as "regularized" and "penalty-based") use the penetration depth between two colliding bodies to generate the contact force that separates these bodies. There are several contact force models, including *PCM* and the *Voxelmap Pointshell Algorithm*. Non-smooth methods (also referred to as "constraint-based" and "complementarity") use unilateral constraints and consider bodies to be rigid. They typically work on the velocity level and can be solved using time-stepping methods.
Simulation environments can be classified according to their application domain: robotics and automation, system simulation, multibody simulation, and manufacturing simulation (see Section 2.3.1). Some of the tools available are based on physics engines such as *Bullet*, *Open Dynamics Engine*, or *MuJoCo*. These engines use the non-smooth approach for the collision response. There are also physics engines that provide both the compliant and non-smooth approach, including *PhysX* and *Project Chrono*.
With the exception of the system simulation tools, all other simulation environments are limited to their respective areas of application. There are tools specifically designed to simulate manipulation and assembly processes: *GraspIt!*, *OpenGRASP*, and *RobWorkPhysicsEngine* (see Section 2.3.2). In comparison to system simulation environments, they focus on the robotics

domain and are limited in their functionality. For example, with these tools it is not possible to model topics such as detailed robot dynamics, energy consumption or flexible bodies.
Therefore, the handling of contacts in system simulation environments has been discussed. In both Modelica and Simscape there is no way to model complex, non-convex contacts between rigid bodies for manipulation and assembly processes out of the box. Several works have attempted to address this issue (see Section 2.3.3). Some methods are limited to primitive geometries or are not real-time capable. Others require predefined contact pairs. This limits the flexibility required for manipulation and assembly (a body is picked up by multiple grippers and placed on multiple tables). None of the analyzed approaches is suitable for complex assemblies consisting of multiple bodies.
Approaches for Variable-Structure Systems were also analyzed (see Section 2.3.4). Some works are limited in their flexibility because they are designed for a specific application domain, or require unique IDs to be defined by the user. Others are not scalable due to their complexity. More recent methods, such as *Modia3D*, do not yet have the multi-domain capabilities (model libraries for multiple domains) to create a system simulation.
Finally, the basics of Virtual Commissioning were presented (see Section 2.4). This includes simulation techniques such as HiL and SiL. Several works on detailed physics simulation were evaluated. Most of the works focus on material flow systems and the connection between the simulation and the control. In addition, models with multiple Levels of Detail were discussed. Two works were analyzed, both in the material flow domain.
To date, there has been little work in the area of detailed simulation of manipulation and assembly processes at the system simulation level, especially in combination with other domains such as energy consumption or flexible bodies.

3 Simulating assembly processes in structure-invariant environments

In this chapter, a novel method for the modeling and simulation of manipulation and assembly processes in structure-invariant environments is presented. First, an example is introduced and an overview of the concept is given. Based on this, algorithms are described in the following sections. The first one enables the connection of components during simulation runtime based on substitute contact forces. Another one allows the stable simulation of assemblies. Finally, the topic of switching between contact models is addressed.

3.1 Overview and block scenario example

The aim of this work is the efficient simulation of manipulation and assembly processes in structure-invariant environments. Structure-invariant means that no structural changes are possible during runtime. Components cannot be connected or disconnected. However, this capability is necessary for manipulation and assembly processes. The following two solutions allow components to be connected during runtime: Using contact forces between components and applying substitute contact forces that hold components in place. The former provides detailed results, while the latter can be computed in real-time.
The contributions in the following sections are presented using the *Block Scenario* example shown in Figure 3.1. It consists of four blocks, namely P, C, O, and B, which are assembled into an assembly. The component names represent the first letter of the component color. The *Block Scenario* example was first published in Reiser (2021).

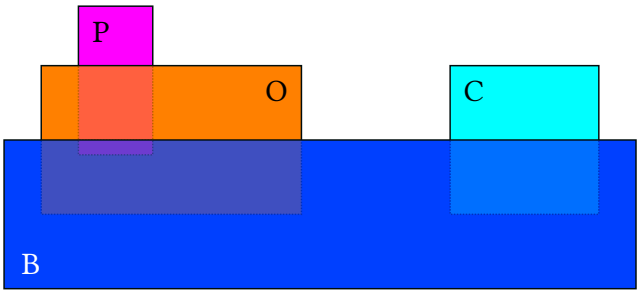

Figure 3.1: Overview of the *Block Scenario* example. It consists of four parts in the form of blocks: the pink part P, the cyan part C, the orange part O, and the blue part B (base). The parts are assembled into an assembly in several steps.

The related assembly process starts with the four blocks placed on a table. In several steps, they are assembled into an assembly. Figure 3.2 illustrates the basic steps of the assembly process. Part C is assembled directly into base part B. Component P, on the other hand, is first assembled into O. The parts P and O together are then assembled into base B. In the final step, the entire assembly is moved to the other table.
Several aspects can be demonstrated in this example: All components are manipulated by multiple robots and grippers and placed on multiple grounds. Components are pre-assembled into subassemblies prior to final assembly. The center of mass of the subassembly is not at the barycenter of part O, therefore a torque might be applied to the gripper.

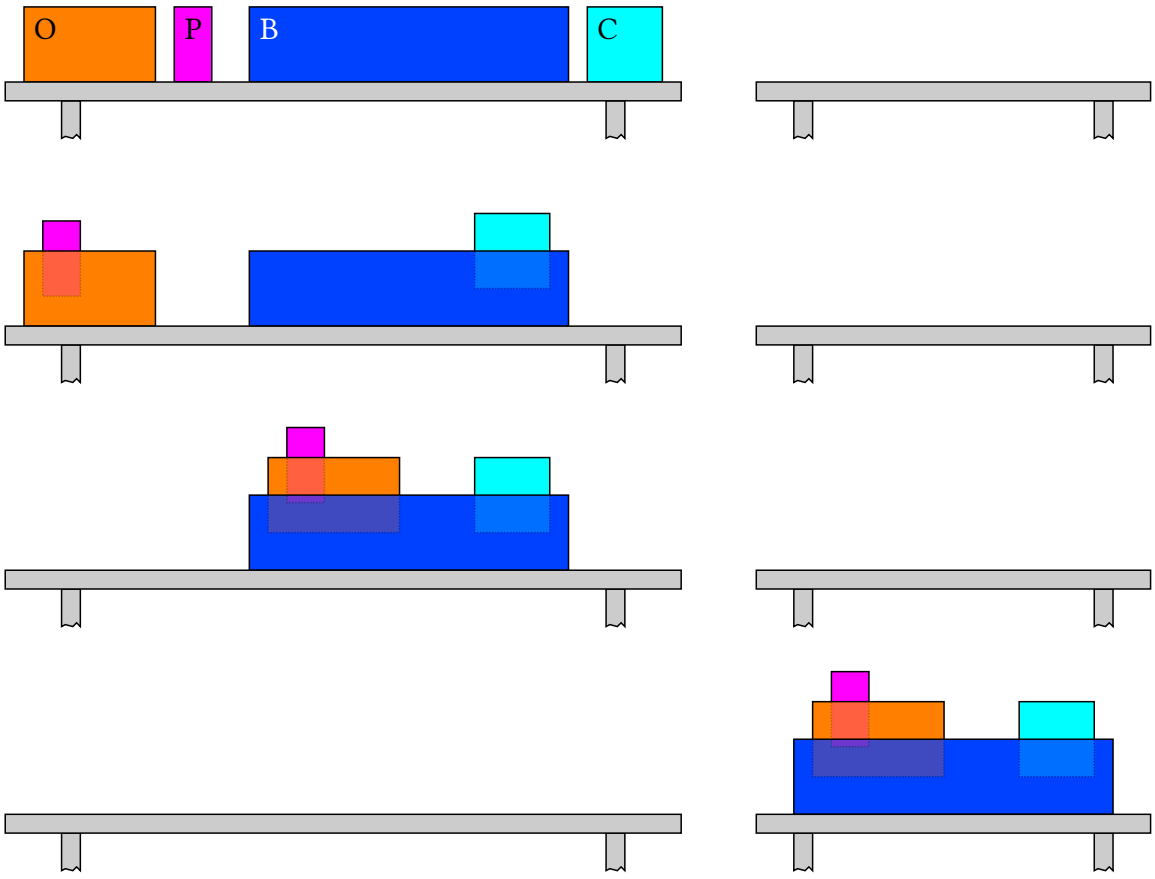

Figure 3.2: Basic assembly process steps for the *Block Scenario* example. The process starts with four blocks placed on the left table. First, the pink component P is assembled into the orange component O and the cyan component C is assembled into the base B (blue). Then, the subassembly consisting of P and O is assembled into the base B. Finally, the entire assembly is picked and placed on the right table.

3.2 Idea and concept

In this section, the basic idea and concept for the simulation of manipulation and assembly processes in structure-invariant environments is presented. It was first published in Reiser (2021) and Reiser and Reiner (2023). First, some basics about component types and component interactions are introduced. Then, the basic ideas of automatic state detection and combining fast and detailed models by using component models are presented. Finally, the concepts are broken down into individual algorithms, which are then presented in the following sections of this chapter.

3.2.1 Component types

In a multibody scenario in the robotics domain, there are multiple objects that interact with each other. Examples include robots, grippers, drills, tool changers, feeders, cameras, sensors, parts, and assemblies. For the purpose of this work, these objects are called *components*. The following component types are the most important ones:

- *Grippers* (usually attached to robots)
- *Grounds* such as tables
- *Parts* that are being manipulated or assembled

For the purpose of this work, only these three component types are primarily considered. However, other components such as robots may be used in models of assembly processes. *Ground* components (e.g. tables) are usually fixed in space and *Grippers* are mounted on robots, which also determines their position and orientation. However, components of type *Part* are manipulated and assembled during a process. Their spatial pose is initially specified in a simulation model and must then be continuously determined. The component types are shown in Figure 3.3.

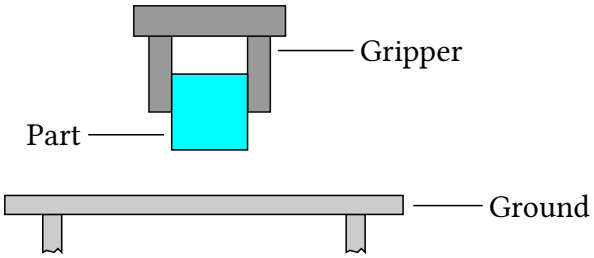

Figure 3.3: Overview of the relevant component types for assembly processes: there are *Grippers*, *Grounds* (e.g. tables), and *Parts* that are being manipulated or assembled.

3.2.2 Component interactions

The interactions between components form the basis for the algorithms presented in the following sections. As discussed in Section 3.2.1, there are three major components: *Gripper*, *Ground*, and *Part*. The component interactions relevant for manipulation and assembly processes are illustrated in Figure 3.4.
In principle, a component of type *Part* can interact with all other components. A *Part* can form a connection with a *Gripper* or a *Ground*. And a *Part* can be part of an assembly. In this case, it forms a connection with one or more other *Parts*. Components of type *Gripper* can only interact with components of type *Part*, but not with components of type *Ground*. Similarly, *Ground* components can only interact with *Part* components. However, collision checks between *Grippers* and *Grounds* are possible for the testing of robot programs.
One objective of this work is flexibility. There are many approaches that require predefined contact pairs (see Section 2). However, this limits flexibility considerably. If the contact between a *Gripper* and a *Part* is predefined in a model, it may not be possible to manipulate that *Part* with another *Gripper*. The aim is therefore to generate connections between components automatically. This is presented in the next section.

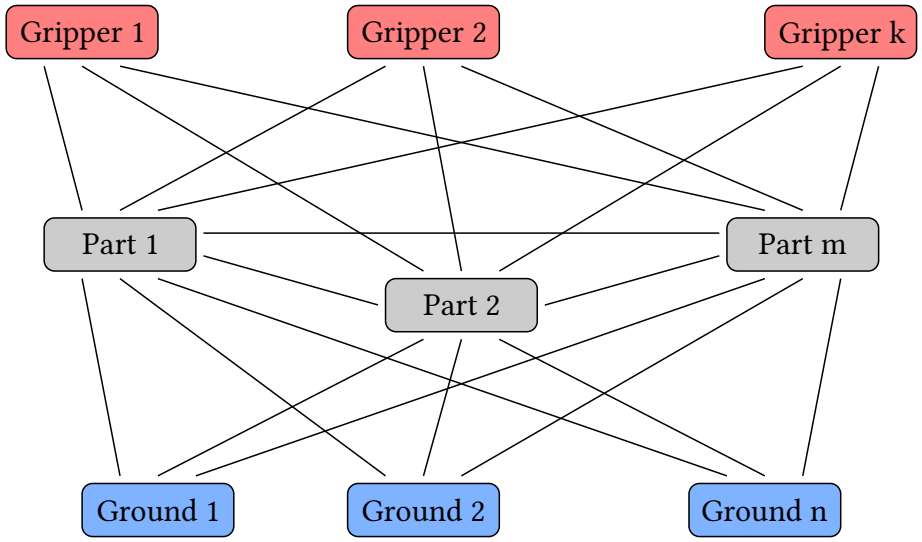

Figure 3.4: Overview of the interactions between components based on their component types. The lines represent possible connections between the components. Connection pairs must involve at least on *Part* component. It is not possible to connect two components of type *Gripper*, two components of type *Ground*, or a component of type *Gripper* and one of type *Ground*.

3.2.3 Automatic state detection

As mentioned in Section 3.2.1, there are three component types. *Ground* components do usually not move and *Grippers* are attached to robots. The pose can therefore be determined for both. *Parts*, on the other hand, are manipulated during a process. A *Part* can be in the following states (shown in Figure 3.5):

- Free motion
- Grasped by a *Gripper*
- Placed on a *Ground*
- Part of an assembly

The basic idea is that *Parts* can automatically determine their state and, based on that, use different models and algorithms to interact with other components in the scenario. This concept was partially introduced in Reiser (2021). The basis for all interactions between these components is collision detection. If a *Part* has no collision, it is in the state free motion. If it collides with a *Gripper*, it will be grasped by that *Gripper*, depending on whether the *Gripper* is closed. When the component is in contact with a *Ground*, it forms a connection with the *Ground*. If multiple collisions occur, the components involved are prioritised.
Automatic state detection is only required for the substitute models based on substitute contact forces. When using detailed contact forces, the state does not need to be known. In this case, the components interact directly through contact forces.

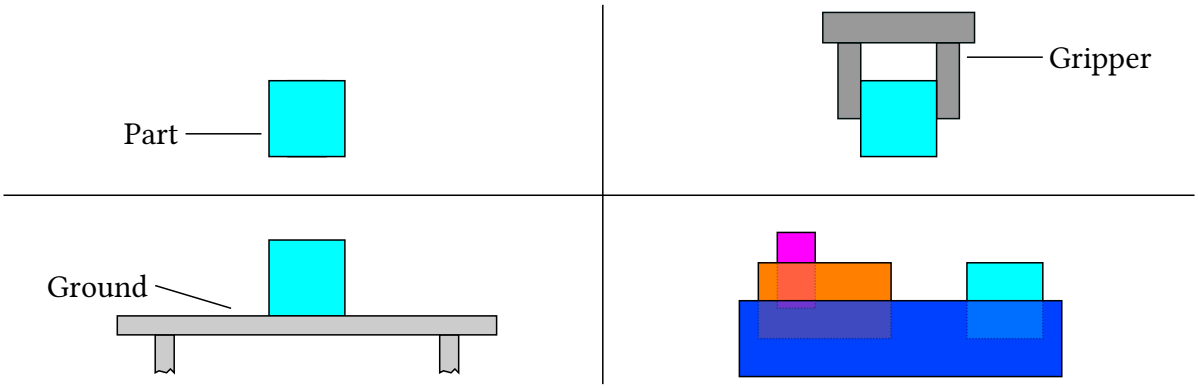

Figure 3.5: The states of a component of type *Part*. It can be moving freely in space (top left), grasped by a *Gripper* (top right), placed on a *Ground* (bottom left), or part of an assembly (bottom right).

3.2.4 Combining fast and detailed models

One overall objective of this work is to reduce the modeling effort. This is achieved by combining fast substitute models based on substitute contact forces and detailed contact force models. The former are real-time capable and can be used for Virtual Commissioning, the latter provide detailed results and are suitable for tasks like optimization.
The idea is to develop **component models** for all the components mentioned in Section 3.2.1 (*Grippers*, *Grounds*, and *Parts*) that contain both fast and detailed models as well as algorithms for manipulation and assembly. These component models are then used to create a process model for a specific assembly process. This model can then be used for multiple application tasks within the engineering process (e.g. Virtual Commissioning or feasibility studies).
Figure 3.6 shows component models for the component types *Part* and *Gripper*. Both contain basic information such as a name and a link to the associated 3D model. Algorithms are integrated to enable the simulation of manipulation processes with fast substitute models or detailed contact force models. In addition, the *Gripper* contains models of the mechanics as well as an input for opening and closing. The *Ground* component model is built in the same way. A specific process can now be modeled by using these component models. The simulation model for the *Block Scenario* example introduced in Section 3.1 is also shown in Figure 3.6. Robot models are additionally used here.

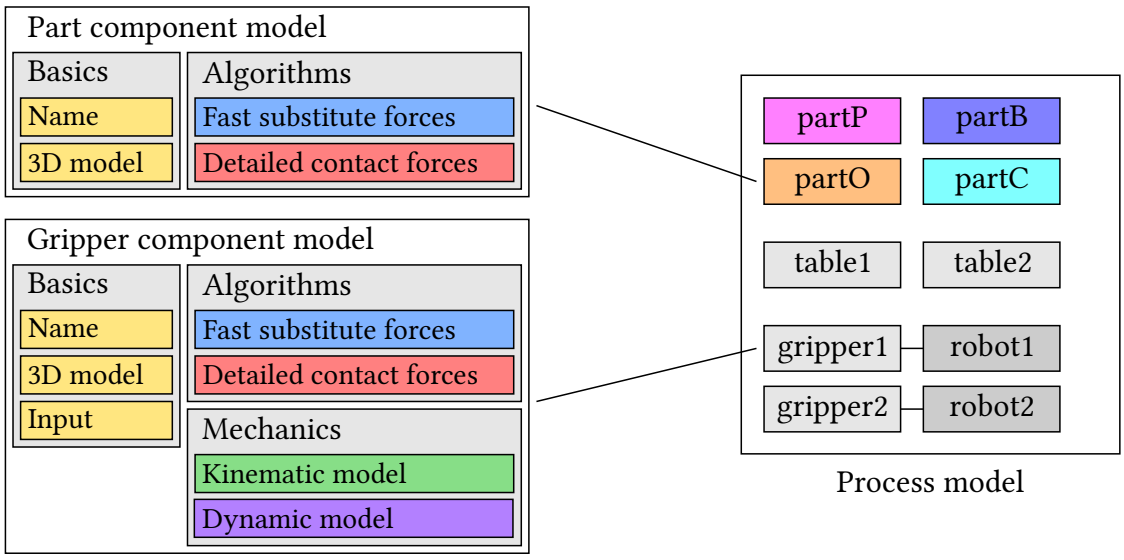

Figure 3.6: Component models for component types *Part* and *Gripper*. They contain both fast and detailed models and algorithms for manipulation and assembly processes. A model of a specific assembly process can be build using these component models. The process model for the *Block Scenario* example is shown on the right.

3.2.5 The algorithm for component interaction

The component interactions presented in Section 3.2.2 are enforced based on forces. There are two ways to connect components during runtime in structure-invariant environments: contact forces and substitute contact forces. The former require a contact force algorithm that uses the polygonal representation of the components. The latter are substitute models that approximate the behavior between the components.
In both cases, an overall algorithm is required to handle the component interactions. Therefore, the Algorithm for Component Interaction (ACI) is developed. A flowchart of this algorithm is shown in Figure 3.7.

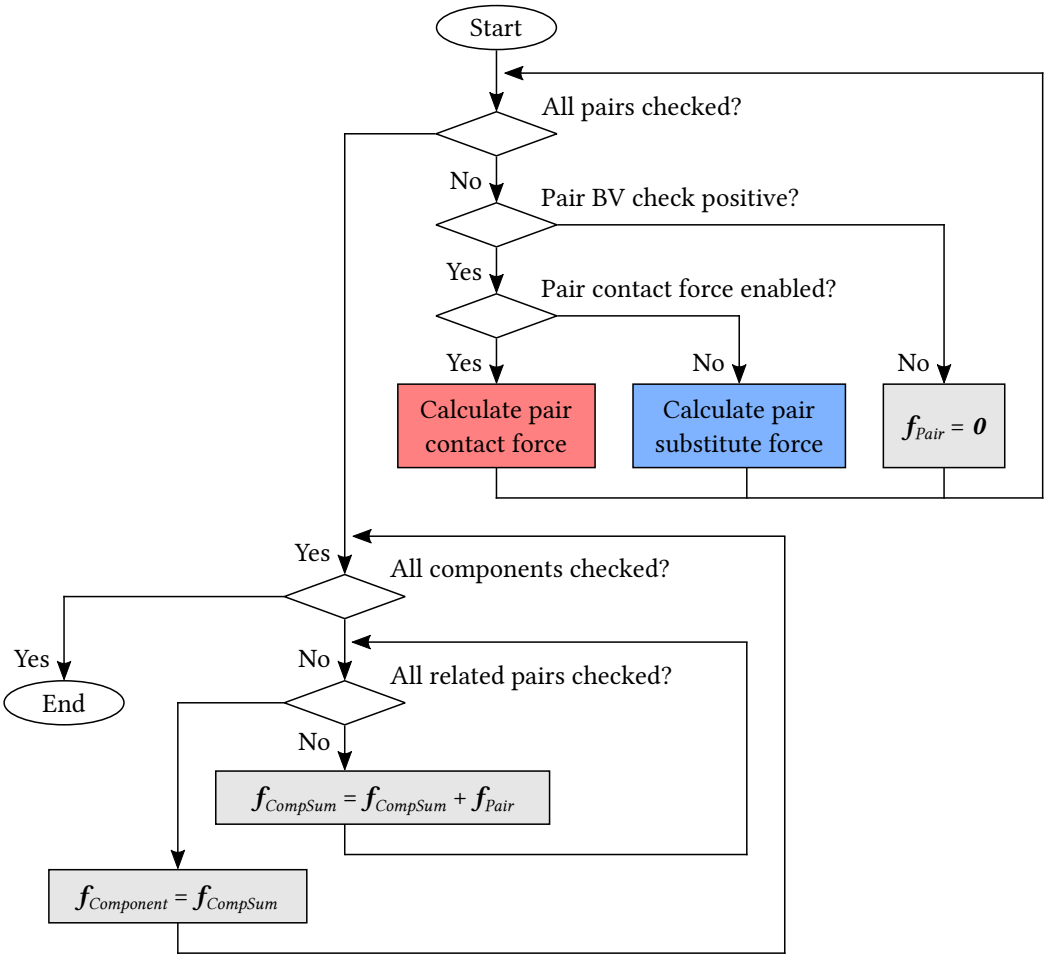

Figure 3.7: A flowchart of the Algorithm for Component Interaction (ACI). It consists of two parts. First, a force is calculated for each component pair (either a contact force or a substitute force). Then, the total force for each component is added up.

The ACI consists of two parts. The first part analyzes the component pairs, the second part analyzes the components. For each component pair, a BV check is performed first. This sequence uses the two-phase approach to collision detection, introduced in Section 2.1.1. If the components are at a distance from each other, the detailed and computationally intensive collision check can be omitted. In this case, the components have no collision and the force is zero. If the BV check is positive, the next step is to check if the contact force is enabled.
If so, a contact force is calculated. This includes collision detection, construction of the intersection surface, calculation of multiple penetration depths and forces, and generation of the resulting contact force.
Otherwise, a substitute contact force is calculated. This includes collision detection, offset calculation, and the calculation of a control force to constrain the components.
When all component pairs have been analyzed, the next step is to analyze the components. For each component, all related component pairs (component pairs that contain the current component) are analyzed. The forces of all related component pairs are summed and stored as the resultant force for the component.
The ACI uses several algorithms and methods to perform the tasks mentioned above:

- An existing algorithm is used to calculate the contact forces. The selection of an appropriate contact force algorithm is discussed in Section 3.3.
- The substitute contact forces that hold the components in place are calculated using the Substitute Force Algorithm introduced in Section 3.4.
- In addition, another algorithm specifically for assemblies, the Assembly Graph Partner Search Algorithm, is introduced in Section 3.5.
- Furthermore, the switching between the contact force algorithm and the Substitute Force Algorithm is presented in Section 3.6.

In this section, only forces that are used to hold components in a certain position are mentioned. Similarly, torques are used to specify the orientation of the components. This applies to both the case where contact forces are used and the case where substitute contact forces are used. The algorithm for contact forces also provides contact torques. The Substitute Force Algorithm also includes torques to constrain the orientation of the components.

3.3 Selection of the contact force model

As mentioned in Section 3.1, there are two possibilities to connect components during runtime in a structure-invariant environment:

- Using contact forces between components
- Applying substitute contact forces that hold components in place

The latter is addressed through algorithms that are presented in the following sections. In contrast, an existing contact force algorithm is used for the former. Selected contact models for the compliant collision response approach are presented in Section 2.2. Physics engines are also introduced in Section 2.3. Most of them use the non-smooth approach, but some also use both collision response approaches.

3.3.1 Classification of contact models

There are multiple categories to classify existing physics engines and contact forces models. Relevant to this work are:

- Body deformability (rigid or deformable bodies, respectively hard and soft bodies)
- Friction (frictional contact and frictionless contact)
- Collision response (non-smooth or compliant approach)
- Contact geometry (point contact or volume contact)
- License (open source or commercial)

In terms of **body deformability**, only rigid bodies are considered (see Section 2.1.1). Contact models and engines for deformable bodies are not analyzed. However, some of the engines mentioned in Section 2.3.1 can also simulate deformable bodies. Strictly speaking, a soft boundary layer is also assumed for rigid body models (applies to both the compliant and the non-smooth approach). This is explained during the presentation of the Elastic Foundation Model (see Section 2.2). The EFM itself can be seen as a theoretical concept and basis for several contact models and is therefore not part of the following comparison.
Contact models with **friction** are required to model and simulate manipulation and assembly processes. For example, without friction it would not be possible to grasp components by force. In addition, many assembly processes cannot be modeled without friction. For this reason, only contact models and physics engines that include frictional forces are analyzed in Chapter 2. And only these are part of the comparison.
As far as the **license** is concerned, the decision is straight forward. All physics engines presented in Section 2.3 are open source and all contact models presented in Section 2.2 are

either open source or at least published. There are also commercial physics engines, but they do not offer significant advantages (Hummel et al. 2012; Erez et al. 2015). Therefore, a contact model that is available as an open source project is used.
This leaves two categories. Existing physics engines and contact models can now be classified in two dimensions. Regarding the **collision response** approach, there are compliant and non-smooth contact models. And the **contact geometry** can be either point contact or volume contact (also related to the collision detection method). Table 3.1 shows the comparison of selected existing contact models and physics engines.
Many collision algorithms use convex geometries and provide point contacts. This can cause several problems, because the contact point can move. For example, when a cuboid is placed on a table. Small rotations of the cuboid change the position of the point with the maximum penetration. The aim of this work is to model and simulate assembly processes where concave bodies are common. Collision detection with convex shapes is therefore not sufficient. Convex decomposition is one way of using these collision detection algorithms. However, instabilities can occur at the transitions between the convex shapes. The alternative is volume contacts. Here, the contact surface and penetration volume are taken into account. This results in more realistic behavior and higher accuracy, especially for complex, non-convex shapes. The disadvantages are increased complexity and lower performance.

Table 3.1: Comparison of selected contact models and physics engines in terms of collision response and contact geometry. There are compliant and non-smooth response methods and the geometry can be either a point or a volume contact.

	Collision response		Contact geometry	
Contact model	Non-sm.	Compl.	Point	Volume
Bullet (Coumans 2024)	x		x	
Open Dynamics Engine (Smith 2024)	x		x	
MuJoCo (Todorov et al. 2012)	x		x	
PhysX (Nvidia 2024) [1](Narang et al. 2022)	x	(x)[1]	x	(x)[1]
Project Chrono [2](Tasora et al. 2016)	x	(x)[2]	x	(x)[2]
PCM (Hippmann 2004b)		x		x
VPA (Sagardia et al. 2014)		x		x
PFC [3](Masterjohn et al. 2022)	(x)[3]	x		x
IdealizedContact (Oestersötebier et al. 2014)		x	x	
Body to Body Contact (Buse et al. 2023)		x	x	

3.3.2 Integration into system simulation environments

The objective of this work is to enable detailed system-level simulation of manipulation and assembly processes in combination with other domains. This is achieved by using a system simulation environment that is extended for the simulation of these processes.
The type of collision response has a strong influence on how well the contact model can be integrated into a system simulation environment. Other factors also play a role. In general, there are several aspects to consider:

- Accuracy (first or second order accurate, respectively velocity or acceleration level)
- Solvers (general purpose solvers or specific solvers such as time-stepping methods)
- Interfaces (standalone use of the contact force model)

One objective is a high **accuracy** and stability. For example, detailed dynamics models are used to represent the behavior of a robot. Physics engines such as Bullet, Open Dynamics Engine, or MuJoCo use point-contacts and non-smooth approaches. These methods use velocity-stepping and "*are first-order accurate at best*" (Elandt et al. 2019, p. 8239). They have no error estimate or error control (Elandt et al. 2019, p. 8239). Furthermore, according to Zechmair and Morel (2022, p. 2), non-smooth approaches can have a weakness with mechanic chains. This is relevant, for example, when modeling grippers with four-bar joints.
Regarding **solvers**, non-smooth approaches often use time-stepping methods to solve the LCP. Most physics engines such as Bullet and Open Dynamics Engine use an iterative Gauss-Seidel solver (J. Bender et al. 2014, p. 261). General purpose or system simulation environments, on the other hand, use solvers for Ordinary Differential Equations (ODEs). Examples are the variable-step solvers *DASSL* and *Cvode* (both implicit, multi-step, and variable order), *Esdirk* (implicit, single step, and fixed order), and the fixed-step solvers *Euler* and *Runge-Kutta* (both explicit) (Otter and Brembeck 2022). According to the work of Machado et al. (2012, p. 100), it is complicated to implement the non-smooth approach in a general purpose environment. It is possible to link contact models and system models using co-simulation based on the FMI standard (Blochwitz et al. 2011). However, the flexibility is limited.
Another aspect are the **interfaces**. In the best case, a contact model is designed to take the position and velocity of the rigid bodies as input and return the contact forces as a result. In other cases, extensive customization may be required for integration.

3.3.3 Evaluation and selection

The contact models and physics engines mentioned in Section 3.3.1 are now evaluated. Several categories were introduced in the two sections above. As only rigid body and friction contact models available as open source projects are considered, these categories are not part of the evaluation. The remaining categories are collision response type, contact geometry, simulation accuracy, solvers and interfaces. For evaluation purposes, these categories are combined into four new categories:

- Geometry: possibility to use complex, non-convex shapes
- Accuracy: volume contacts and simulations on the acceleration level
- Solvers: usability of the model with general purpose solvers
- Interfaces: standalone use of the model

The results are shown in Table 3.2. As expected, the non-smooth approaches are limited in their geometric representations and less accurate. *PhysX* and *Project Chrono* are an exception, as they also provide a compliant model. For a detailed comparison of physics engines, see Hummel et al. (2012) and Erez et al. (2015).
The most suitable models are those that offer complex, non-convex geometries, acceleration level precision, and the use of general purpose solvers. These are the PCM, the PFC model, and the VPA. The latter is less accurate, since it is optimized for fast simulations. The PCM and the PFC model are relatively similar. However, the PCM is easier to integrate as it is specifically designed for integration in multibody environments. It is also a well-established algorithm used in several multibody tools, such as *Simpack* (Hippmann 2024). As a result, the Polygonal Contact Model is used as the contact force model in this work.

Table 3.2: Evaluation of selected contact models and physics engines in terms of geometry (use of complex, non-convex shapes), accuracy (use of volume contacts and simulations with acceleration level precision), solvers (usability of general purpose solvers), and interfaces (standalone use of the model).

Contact model	Geom.	Accur.	Solv.	Interf.
Bullet (Coumans 2024)	-	-	-	+
Open Dynamics Engine (Smith 2024)	-	-	-	+
MuJoCo (Todorov et al. 2012)	-	-	-	+
PhysX (Nvidia 2024)	+	+	-	+
Project Chrono (Tasora et al. 2016)	+	+	-	+
PCM (Hippmann 2004b)	++	++	+	++
VPA (Sagardia et al. 2014)	++	+	+	+
PFC (Masterjohn et al. 2022)	++	++	+	+
IdealizedContact (Oestersötebier et al. 2014)	-	+	+	+
Body to Body Contact (Buse et al. 2023)	+	-	+	++

3.4 Algorithm for dynamic structural changes

In structure-invariant environments, it is not possible to connect or disconnect components during simulation runtime. However, this capability is required for the substitute models. In this section, an algorithm is presented that allows components to be dynamically constrained together in simulations. It is called the Substitute Force Algorithm. In addition, an extension for manipulation and assembly processes is introduced.

3.4.1 The substitute force algorithm

The basis for constraining components during runtime for the substitute models is the Substitute Force Algorithm (SFA). It calculates the substitute contact force for a component pair that holds both components together and was first published in Reiser (2021). The opening and closing of a connection is defined, for example, by commands for the gripper in the robot program. The flowchart of the algorithm is shown in Figure 3.8.
The SFA starts with a collision check. Either the convex hulls or the convex decompositions of the components can be used. If the collision check is negative, the components are not in contact with each other. In this case, the substitute contact force is zero. The same applies if the substitute force for one of the two components is disabled. If the collision check for

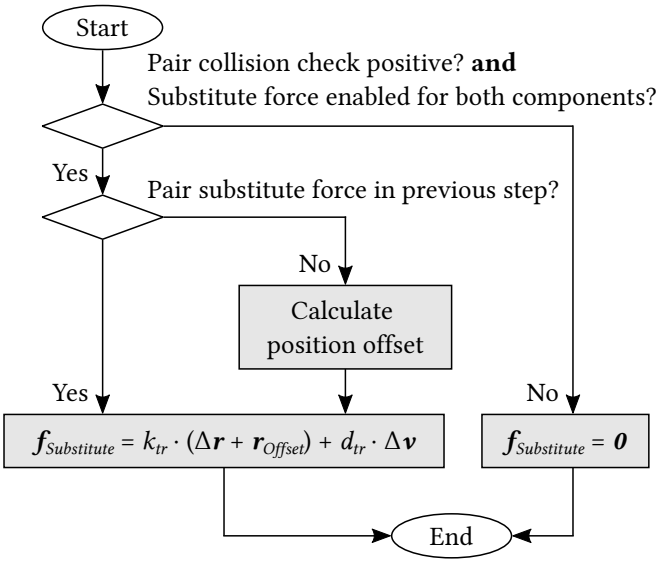

Figure 3.8: A flowchart of the Substitute Force Algorithm used to calculate the substitute contact force for a component pair. If the substitute force is enabled for both components and a collision occurs for the pair, the force is calculated based on Equation (3.5). The substitute force allows components to be connected or disconnected during the simulation runtime.

the pair is positive and the substitute force is enabled for both components, the next step is to check if this pair had a substitute force in the previous simulation step. If not, an offset is calculated for the position and orientation. Then, the substitute contact force is calculated. Similar to the PCM algorithm (see Section 2.2.2), one component of the pair is defined as the *Base* and the other one as the *Target*. The substitute contact force is calculated from the perspective of the *Base* component. The resulting substitute contact force for the *Target* component is then calculated from the substitute contact force for the *Base* component. The substitute force consists of two parts: a force for the position and a torque for the orientation. The calculation of the **force** starts with the position offset $\boldsymbol{r}_{Offset,\ Base}$:

$$\boldsymbol{r}_{Offset,\ Base} = \boldsymbol{r}_{Base} - \boldsymbol{r}_{Target} \tag{3.1}$$

where $\boldsymbol{r}_{Base}$ is the *Base* position and $\boldsymbol{r}_{Target}$ the position of the *Target* component. This offset is now transformed from the world frame to the frame of the *Target* component:

$${}^{Target}\boldsymbol{r}_{Offset,\ Base} = \boldsymbol{T}_{Target} \cdot \boldsymbol{r}_{Offset,\ Base}. \tag{3.2}$$

The desired position for the *Base* component $\boldsymbol{r}_{desired,\ Base}$ is given by:

$$\boldsymbol{r}_{desired,\ Base} = \boldsymbol{r}_{Target} + \boldsymbol{T}_{Target}^{T} \cdot {}^{Target}\boldsymbol{r}_{Offset,\ Base} \tag{3.3}$$

where $\boldsymbol{T}_{Target}$ is the rotation matrix of the *Target* component. Figure 3.9 shows the position offsets for two component pairs in the *Block Scenario* example (one *Part* is constrained to the *Ground* and one to the *Gripper*).

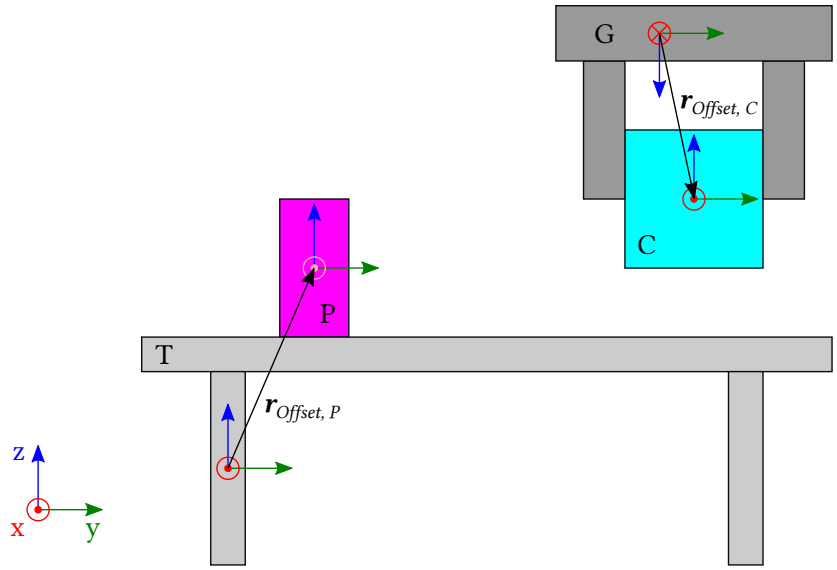

Figure 3.9: Overview of the substitute contact force calculation for parts of the *Block Scenario* example. Component P (*Base*) is constrained to the table T (*Target*) and component C (*Base*) is constrained to the gripper G (*Target*). The image shows the frames of the components and the position offsets for both pairs.

The substitute contact force for the *Base* component is then calculated by:

$$\boldsymbol{f}_{Substitute,\ Base} = k_{tr} \cdot (\boldsymbol{r}_{desired,\ Base} - \boldsymbol{r}_{Base}) + d_{tr} \cdot (\boldsymbol{v}_{Target} - \boldsymbol{v}_{Base}) \tag{3.4}$$

$$\boldsymbol{f}_{Substitute,\ Base} = k_{tr} \cdot (\boldsymbol{r}_{Target} + \boldsymbol{T}_{Target}^{T} \cdot {}^{Target}\boldsymbol{r}_{Offset,\ Base} - \boldsymbol{r}_{Base}) + d_{tr} \cdot (\dot{\boldsymbol{r}}_{Target} - \dot{\boldsymbol{r}}_{Base}) \tag{3.5}$$

where k_{tr} is the translational spring constant, d_{tr} the translational damping constant, and $\boldsymbol{v}_{Target}$ and $\boldsymbol{v}_{Base}$ the velocities of the *Base* and *Target* component. The resulting substitute force for the component *Target* is calculated according to Newton's third law as follows:

$$\boldsymbol{f}_{Substitute,\ Target} = -1 \cdot \boldsymbol{f}_{Substitute,\ Base}. \tag{3.6}$$

The calculation of the **torque** starts with the rotation offset $\boldsymbol{T}_{Offset,\ Base}$:

$$\boldsymbol{T}_{Offset,\ Base} = \boldsymbol{T}_{Target} \cdot \boldsymbol{T}_{Base}^{T} \tag{3.7}$$

where $\boldsymbol{T}_{Target}$ is the rotation matrix of the *Target* rotation matrix and $\boldsymbol{T}_{Base}$ the rotation matrix of the *Base* component. The desired rotation matrix for the *Base* component is:

$$\boldsymbol{T}_{desired,\ Base} = \boldsymbol{T}_{Offset,\ Base} \cdot \boldsymbol{T}_{Target}. \tag{3.8}$$

The relative rotation is then:

$$\boldsymbol{T}_{relative,\ Base} = \boldsymbol{T}_{desired,\ Base} \cdot \boldsymbol{T}_{Base}^{T}. \tag{3.9}$$

For small rotations, the rotation angles can be taken from the rotation matrix as follows (they are resolved in the frame of the *Base* component):

$${}^{Base}\boldsymbol{\alpha}_{relative,\ Base} = \begin{pmatrix} \boldsymbol{T}_{relative,\ Base}\,[2,\ 3] \\ -\boldsymbol{T}_{relative,\ Base}\,[1,\ 3] \\ -\boldsymbol{T}_{relative,\ Base}\,[1,\ 2] \end{pmatrix}. \tag{3.10}$$

The difference in the angular velocity $\Delta\boldsymbol{\omega}_{Base}$ is based on the angular velocity of the *Base* and *Target* component. All velocities are resolved in the frame of the *Base* component:

$${}^{Base}\Delta\boldsymbol{\omega}_{Base} = {}^{Base}\boldsymbol{\omega}_{Target} - {}^{Base}\boldsymbol{\omega}_{Base}. \tag{3.11}$$

The substitute torque for the *Base* component is then calculated by:

$${}^{Base}\boldsymbol{t}_{Substitute,\ Base} = k_{rot} \cdot {}^{Base}\boldsymbol{\alpha}_{relative,\ Base} + d_{rot} \cdot {}^{Base}\Delta\boldsymbol{\omega}_{Base} \tag{3.12}$$

where k_{rot} is the rotational spring constant and d_{rot} is the rotational damping constant. This torque is now transformed from the frame of the *Base* component to the world frame:

$$\boldsymbol{t}_{Substitute,\ Base} = \boldsymbol{T}_{Base}^{T} \cdot {}^{Base}\boldsymbol{t}_{Substitute,\ Base}. \tag{3.13}$$

As with the substitute force, the resulting substitute torque for the component *Target* is calculated according to Newton's third law as follows:

$$\boldsymbol{t}_{Substitute,\ Target} = -1 \cdot \boldsymbol{t}_{Substitute,\ Base}. \tag{3.14}$$

Offsets are only calculated at time steps when connections are closed. This works well with fixed-step solvers. However, for variable-step solvers that have error control, it is important to ensure that the offset is not set more than once. For the SFA, a small relative velocity between the two components is assumed because otherwise unphysical effects might occur.

3.4.2 Extension for manipulation and assembly

The Substitute Force Algorithm can be used to constrain multiple components in a multibody scenario. However, additional functionality is required to simulate manipulation and assembly processes. These include the limitation and prioritizing of connections and hierarchy levels for components of type *Part*.
Figure 3.10 shows the pickup of component C from the *Block Scenario* example. In the first step, the *Gripper* G is in the correct position and the *Part* C is constrained to the table T (type *Ground*). The next step is to connect the *Part* component to the *Gripper*. When the substitute force for the *Gripper* is enabled, there are two substitute forces acting on component C.
Component C must therefore **limit and prioritize the connections**. This is done based on the component types introduced in Section 3.2.1. For the example in Figure 3.10, this means that, from the perspective of C, G is higher prioritized than T. In addition, a *Part* component can be either constrained to a *Gripper* component, to a *Ground* component, or to several other *Part* components (not relevant in this example). For this reason, the connection of component C switches from *Ground* T to *Gripper* G when the substitute contact force for the *Gripper* is enabled. In this step, the position offset is also recalculated (now based on the *Gripper*, see Figure 3.10). This extension enables the simulation of manipulation processes.
Another aspect are assemblies. The assembly process for the *Block Scenario* example shown

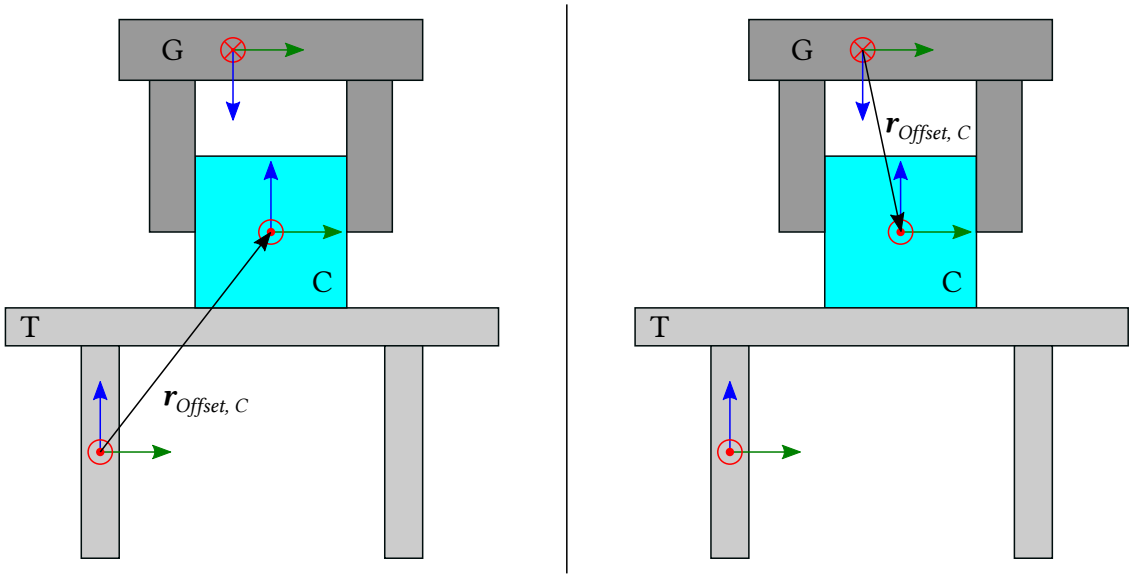

Figure 3.10: The component C from the *Block Scenario* example (component type *Part*) is picked by the *Gripper* G. The image shows two simulation steps. In the first step, shown on the left, C is constrained to the table T (type *Ground*). The right side shows the second step, where C is constrained to the *Gripper* G. The connection switches when the substitute contact force for the *Gripper* is enabled.

in Figure 3.2 can partly be simulated with the current extension (see Section 3.1). Component C is picked with a *Gripper* (connection switches from *Ground* to *Gripper*) and assembled to B (connection switches from *Gripper* to B). The same applies to component P. However, the subassembly consisting of P and O cannot be picked. If the *Gripper* picks component O, it terminates its connection to P and P drops to the table.
It is therefore necessary to introduce **hierarchy levels** to the components of type *Part*. Only the differentiation of components by hierarchy levels is relevant. It doesn't matter how many hierarchy levels there are. An example are the following five hierarchy levels:

- *Lowest*
- *Low*
- *Medium*
- *High*
- *Highest*

For the example in Figure 3.2, component B has the hierarchy level *High*, component P the hierarchy level *Low*, and the components C and O have the hierarchy level *Medium*. The hierarchy levels represents the assembly sequence.
Figure 3.11 shows a flowchart of the extended Substitute Force Algorithm. The algorithm is now extended with both the limitation and prioritization of connections and hierarchy levels for components of type *Part*. Unlike the original SFA, the extended SFA does not iterate over the pairs, but over the components. The flowchart shows how the resulting substitute contact force for one component is calculated depending on its component type.
The process for one component of type *Ground* in the extended SFA consists of iterating over all other components. Components of type *Gripper* are ignored and for components of type *Part* the substitute contact force is calculated and added if they are not grasped.
For one component of type *Gripper* it is similar. Components of type *Ground* are ignored and all components of type *Part* are considered for the force calculation.
For *Part* components, the first step is to check whether the component is grasped by a *Gripper*. If this is the case, the substitute contact force for the connection to the *Gripper* is calculated and added to the sum of the forces. If no *Gripper* is involved, it is checked if the component has a connection with a *Ground*. In this case, the associated constraining force is calculated and added to the sum of the forces.
It is then iterated over all the other components (also called counterparts), both for the case where the component is grasped and for the case where it is connected to the ground. Counterparts of type *Part* are taken into account, but only if they are not grasped or are grasped and their hierarchy is higher than that of the current component.
With this extended Substitute Force Algorithm, the assembly process for the *Block Scenario* example (see Figure 3.2) can now be simulated. The subassembly consisting of P and O can

now be picked with the *Gripper* and assembled to B. In addition, the entire assembly can be manipulated by grasping component B.
However, there are still limitations, especially for complex assemblies. Therefore, an additional algorithm for the stable simulation of assemblies is introduced in the next section.

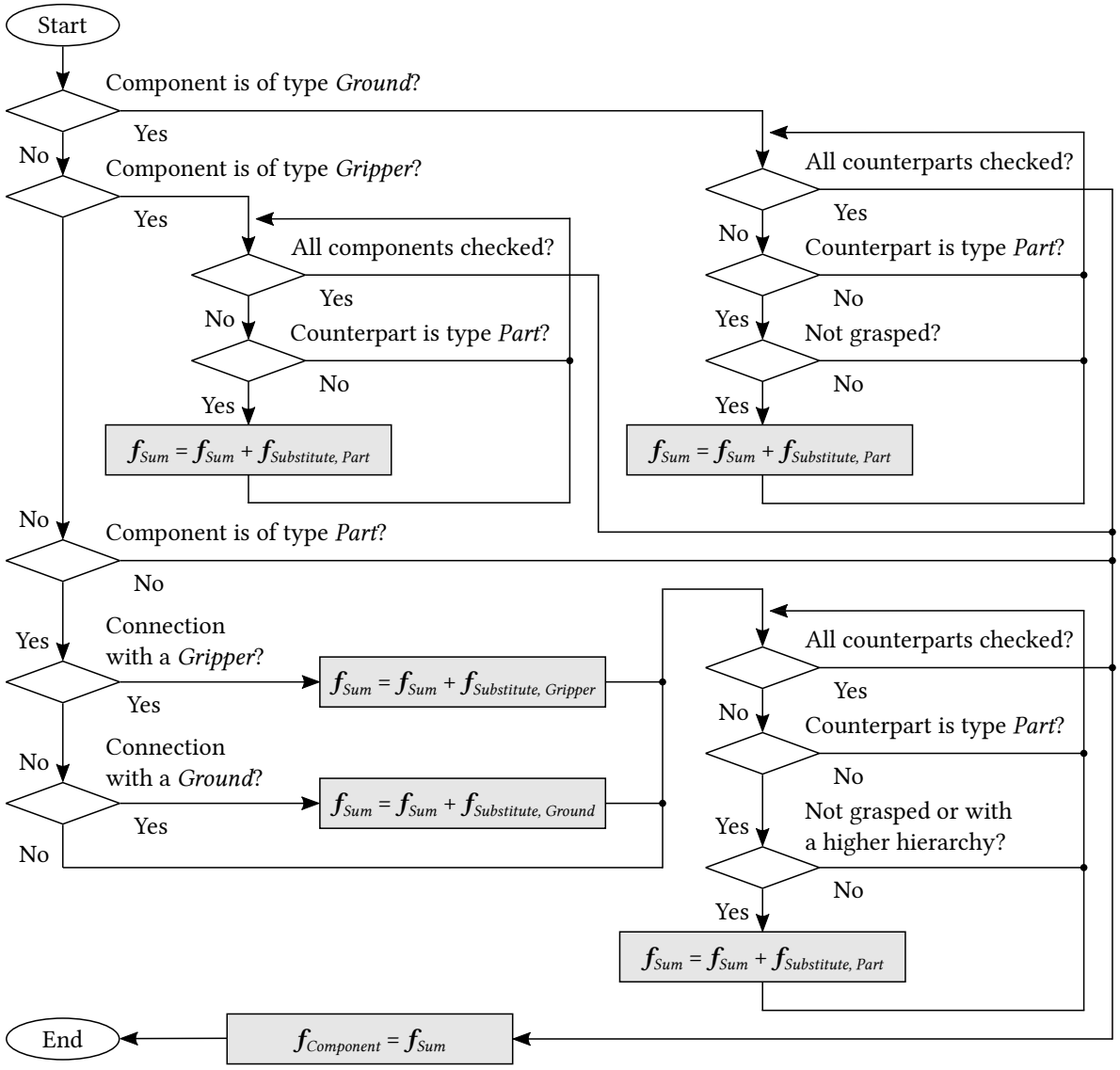

Figure 3.11: A flowchart of the Substitute Force Algorithm extended for manipulation and assembly. It shows the process of calculating the substitute contact force for one component depending on its type. For components of type *Part*, it is checked if they are connected to a *Gripper* or a *Ground* and other *Part* components are only considered, if they are not grasped or are grasped and have a higher hierarchy.

3.5 Algorithm for the stable simulation of assemblies

In Section 3.4, an algorithm is introduced that allows to connect or disconnect components during simulation runtime. This Substitute Force Algorithm is used for the substitute models and allows to model manipulation processes and simple assemblies. However, the algorithm has limitations for more complex assemblies. For this reason, in this section another algorithm specifically for the stable simulation of assemblies is presented. It is called the Assembly Graph Partner Search Algorithm. First, the need for an additional algorithm is discussed, and then the algorithm itself is presented.

3.5.1 Motivation

The Substitute Force Algorithm (presented in Section 3.4.1) is sufficient for manipulation processes or simple assemblies. For example, a *Part* component placed on a *Ground* or a *Part* component grasped by a *Gripper* can be modeled well with the algorithm (see Figure 3.9). In both cases, only one substitute contact force occurs for each component. In general, a *Part* component can be constrained to a *Ground* (e.g. a table), a *Gripper*, or another *Part* component. If multiple *Part* components are connected within an assembly, multiple substitute contact forces can occur for the involved components. This can cause problems.
Figure 3.12 shows the substitute contact forces between all components and the related contact graph for the *Block Scenario* example. Part P is connected to part O. Part O and part C are

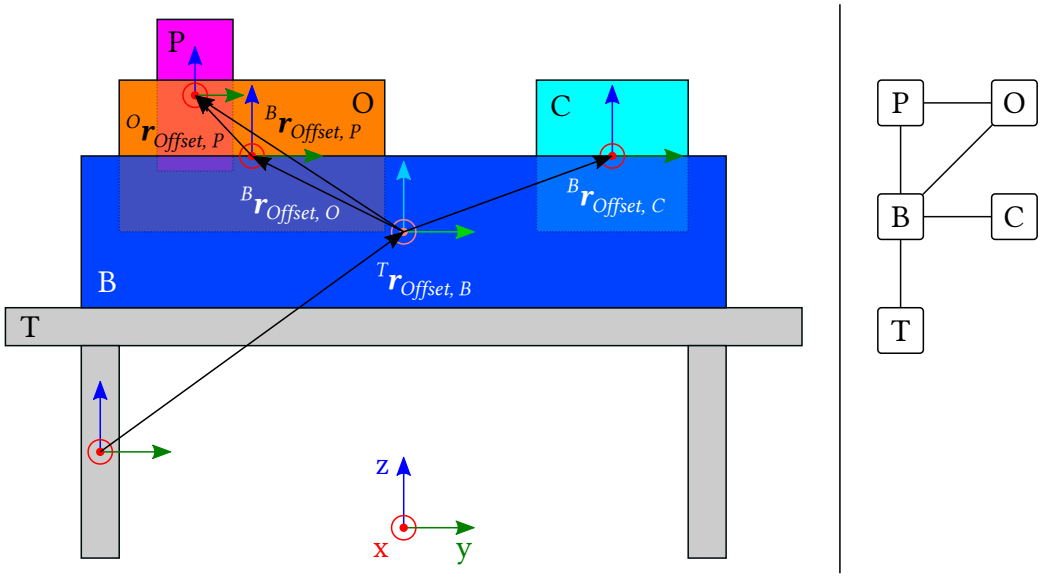

Figure 3.12: The assembly of the *Block Scenario* example placed on the table T. The image shows the component frames and the position offsets for all substitute forces on the left. On the right is the graph of all the contacts between the components.

connected to the base B. The base B is connected to the table T. Part P is also connected to the base B because the collision detection is based on convex hulls and the components P and B overlap. Even if a more detailed collision detection is used (e.g. convex decomposition), some component geometries lead to scenarios like this one.
The contact graph in Figure 3.12 shows all contacts between the components. The contact graph G_C can be described as follows:

$$G_C = (V,\ E) \tag{3.15}$$
$$V = \{P,\ O,\ B,\ C,\ T\} \tag{3.16}$$
$$E = \{(P,\ O),\ (P,\ B),\ (O,\ B),\ (B,\ C),\ (B,\ T)\} \tag{3.17}$$

where V is a finite set of vertices and E a set of edges.
Component C is constrained to B, which in turn is constrained to T. This dependency can be described by a Path Vertices Sequence (PVS). A PVS is defined as a sequence of vertices in a graph whereby the vertices and the edges are distinct (see Section A.1.1).
One possible PVS from the *Part* component C to the table T (*Ground* component) is:

$$PVS_{C\to T} = (C,\ B,\ T). \tag{3.18}$$

Each edge in the Path Vertices Sequence (3.18) represents a substitute contact force between two components. This results in two problems for the simulation model: substitute contact forces connected in series and substitute contact forces connected in parallel.
One problem is **substitute contact forces connected in series**. In the contact graph in Figure 3.12, the dependency between component C and component T is the PVS defined in Equation (3.18). As the stiffness for the elastic part of the substitute contact force calculation is limited, there is a position error for the held *Part* components. The spring-damper model for the connection of the *Part* components B and C with the table T (type *Ground*) is illustrated in Figure 3.13. The position difference for a single component B, Δz_B, with mass m_B and translational spring constant k_{BT} is calculated by equating the weight force and the spring force:

$$f_{s,\,B} = f_{g,\,B} \tag{3.19}$$
$$\Rightarrow \Delta z_B \cdot k_{BT} = m_B \cdot g \tag{3.20}$$
$$\Rightarrow \Delta z_B = \frac{m_B \cdot g}{k_{BT}}. \tag{3.21}$$

For a combination of part B and C connected in series, the position difference for both components changes (see Figure 3.13). The position difference for part B, $\Delta z'_B$, is now:

$$f'_{s,\,B} = f_{g,\,B} + f_{g,\,C} \tag{3.22}$$
$$\Rightarrow \Delta z'_B \cdot k_{BT} = (m_B \cdot g) + (m_C \cdot g) \tag{3.23}$$
$$\Rightarrow \Delta z'_B = \frac{(m_B + m_C) \cdot g}{k_{BT}}. \tag{3.24}$$

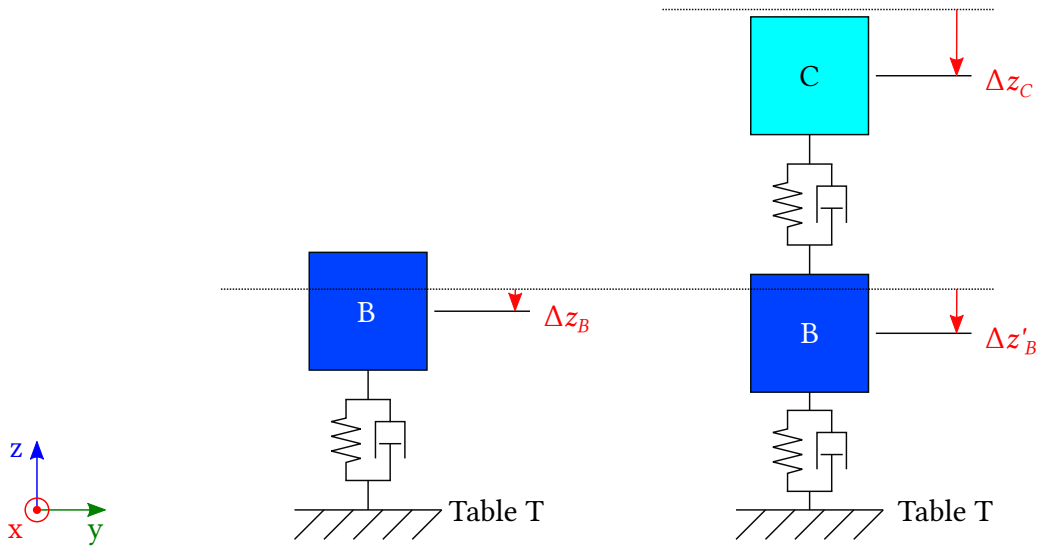

Figure 3.13: A spring-damper model for the components C and B of the *Block Scenario* example. Both are connected with the table T. The position difference for part B without part C is shown on the left. On the right is the position difference for part B and C when they are connected in series.

The position difference of part C, Δz_C, results from $\Delta z'_B$ and the deviation based on the spring between part B and C with spring stiffness k_{CB} (equivalent to Equation (3.21)):

$$\Delta z_C = \Delta z'_B + \frac{m_C \cdot g}{k_{CB}}. \tag{3.25}$$

This means that the position error increases for both component B and component C.
Another problem is **substitute contact forces connected in parallel**. Substitute contact forces connected in parallel lead to stiffness connected in parallel. This occurs in part P. There are two possible PVSs from part P to the table T:

$$PVS_{P\to T} \in \big\{(P,\ B,\ T), \tag{3.26}$$

$$(P,\ O,\ B,\ T)\big\}. \tag{3.27}$$

For PVS (3.26), the stiffness of part P, k_P, is the stiffness between part B and part P. In case of PVS (3.27), the stiffness of part P changes to k'_P, resulting from two parallel springs:

$$k_P = k_{PB} \tag{3.28}$$

$$k'_P = k_{PB} + k_{PO}. \tag{3.29}$$

This increases the stiffness, resulting in deteriorated numerical properties of the model. To prevent this, the step size must be reduced. But then it might not be possible to simulate the model in real-time. Also, finding the maximum step size is not trivial, as it is not explicitly known in advance how many parallel substitute forces will occur during an assembly process.

Both problems can be avoided by **using single connections** so that no substitute contact force connected in series or connected in parallel occurs. This can be achieved by connecting all components of the assembly directly to the respective gripper, table, or assembly base. Instead of generating chains of substitute contact forces, a single direct substitute contact force is calculated for each *Part* component of the assembly. The position offset for each part of the assembly is calculated when the assembly comes into contact with the gripper or table. These offsets are then used as references for the substitute contact force calculations.
The result of using single connections for the position difference is shown in Figure 3.14. The position difference for part B is now that of a single component as shown in Equation (3.21). For part C, the position difference Δz_C changes from Equation (3.25) to:

$$\Rightarrow \Delta z_C = \frac{m_C \cdot g}{k_{CT}} \tag{3.30}$$

where m_C is the mass of the component and k_{CT} the translational spring constant.
The use of single connections also prevents substitute contact forces connected in parallel and the associated increase in the stiffness of the model. The stiffness of part P changes from Equation (3.29) to the single connection to the table:

$$k_P = k_{PT}. \tag{3.31}$$

The aim of using single connections is to ensure the stable and efficient positioning of assemblies during runtime for real-time simulations and tasks such as Virtual Commissioning. Here, the focus is on aspects such as collision detection, and contact forces are not relevant. However, if contact forces are required, the detailed contact force model in the form of the Polygonal Contact Model can be used. The loads on the tables and robots are identical in the static case and are an approximation in the dynamic case for the substitute model.

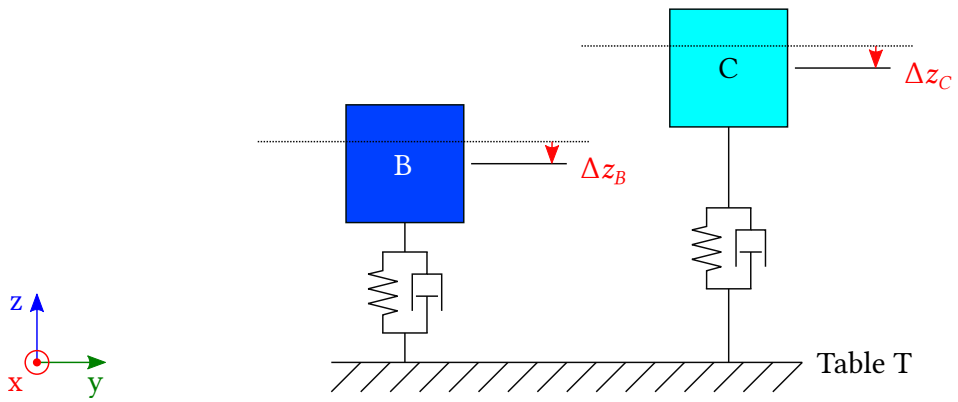

Figure 3.14: The spring-damper model for the components C and B of the *Block Scenario* example from Figure 3.13 now with single connections to the table T. The weight force of part C no longer affects the position difference of part B and the position difference of part B no longer affects the position difference of part C because the parts are no longer connected in series.

3.5.2 The assembly graph partner search algorithm

A solution for the problems presented in Section 3.5.1 is the **Assembly Graph Partner Search Algorithm (AGPSA)**. This algorithm finds all possible PVSs for a given component of type *Part*. Based on the list of possible PVSs, a partner component is identified that fits best. *Gripper* components are preferred over *Ground* components, which in turn are preferred over *Part* components. In this way, any *Part* component within an assembly can be constrained directly to a *Gripper* or *Ground*, without substitute forces connected in series or parallel.
The AGPSA is explained using the *Block Scenario* example. Figure 3.15 shows the *Block Scenario* assembly placed on the table T (as in Figure 3.12), with component O now grasped by the *Gripper* G. All Path Vertices Sequences for the *Part* component P are:

$$PVS_P \in \big\{(\boldsymbol{P},\ \boldsymbol{O},\ \boldsymbol{G}), \tag{3.32}$$
$$(P,\ B,\ C), \tag{3.33}$$
$$(\boldsymbol{P},\ \boldsymbol{B},\ \boldsymbol{T}), \tag{3.34}$$
$$(P,\ O,\ B,\ C), \tag{3.35}$$
$$(\boldsymbol{P},\ \boldsymbol{O},\ \boldsymbol{B},\ \boldsymbol{T}), \tag{3.36}$$
$$(P,\ B,\ O,\ G)\big\}. \tag{3.37}$$

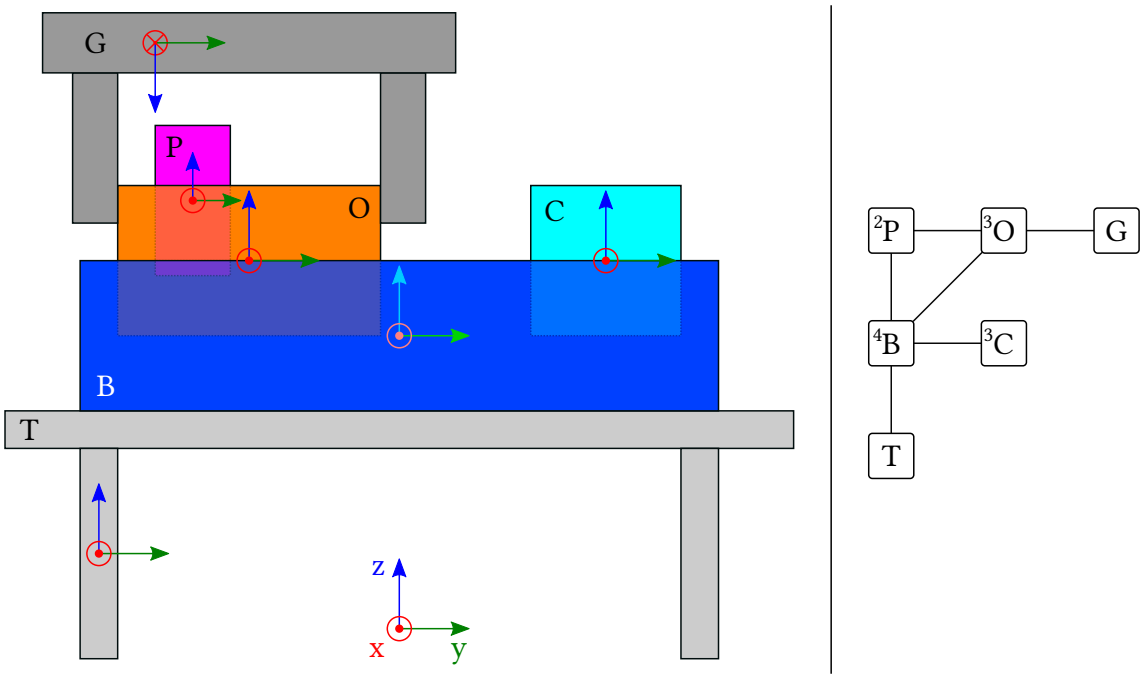

Figure 3.15: The assembly of the *Block Scenario* placed on the table T. Part O is grasped by the gripper G. The graph with the contacts between all components is shown on the right. The numbers represent the hierarchy for the components of type *Part*. A higher number indicates a higher hierarchical level.

But not all of these Path Vertices Sequences are useful. The PVSs (3.33), (3.35), and (3.37) do not provide feasible results. Only the PVSs marked in bold are feasible. Therefore, the hierarchy levels presented in Section 3.4.2 are now used to represent the assembly sequence. In the assembly of the *Block Scenario* example shown in Figure 3.15, component B has the highest hierarchy, namely 4. For components O and C, the hierarchy level is 3 and for component P it is 2. These hierarchy levels are shown in the contact graph in Figure 3.15.
The Path Vertices Sequences for component P are now generated using the AGPSA. The algorithm works as follows. It finds all PVSs for a given *Part* component. For each *Part* component, all components in contact with it are analyzed. Components can only be added to PVSs if they are not already part of them. This prevents loops. In addition, the hierarchy level of the *Part* components is now taken into account. Now a counterpart can be added to a PVS in the following cases:

- The counterpart is of type *Gripper* (hierarchy level is irrelevant)
- The counterpart is of type *Ground* (hierarchy level is irrelevant)
- The counterpart is of type *Part* and has a higher hierarchy level than the hierarchy level of the current *Part* component

Based on this restriction, the PVSs (3.33), (3.35), and (3.37) are avoided:

- The last step for PVS (3.33) is impossible because the hierarchy of component C is lower than the hierarchy of component B
- The same applies to PVS (3.35)
- The second step of PVS (3.37) is also not possible because the hierarchy of component O is lower than the hierarchy of component B

With this modification, the possible PVSs for the *Part* component P are:

$$PVS_P \in \big\{(P,\ O,\ G), \tag{3.38}$$
$$(P,\ B,\ T), \tag{3.39}$$
$$(P,\ O,\ B,\ T)\big\}. \tag{3.40}$$

The Assembly Graph Partner Search Algorithm is shown in Algorithm 1. The algorithm requires a list of all components for a given model of a manipulation or assembly process. It iterates over all the components in this list. Only components of type *Part* are considered. Components of type *Gripper* and *Ground* are skipped. For components of type *Part*, an empty queue and an empty list for the resulting PVSs are created in the first step. Then, a vector is created for the first PVS and the current component is added as the first element. This vector is then added to the queue.
The following actions are then performed in a loop as long as there are elements in the queue. A vector is created with the first element in the queue. Its last element is stored

Algorithm 1 Assembly Graph Partner Search Algorithm (AGPSA)

Require: *componentList* (a list containing all components in the model)

```
1:  for (each component in componentList) do
2:      if (component.type==Part) then
3:          Create empty queue and empty vector resultingPVSs
4:          Create vector firstPVS and add component
5:          queue.enqueue(firstPVS)
6:          while (queue is not empty) do
7:              currentPVS = queue.dequeue()
8:              currentComponent = currentPVS.back()
9:              newElementsFound = false
10:             for (each collidedComp in currentComponent.collidedCompList) do
11:                 if ((collidedComp is not part of currentPVS) and
12:                     (collidedComp.type==Gripper or collidedComp.type==Ground
13:                     or (collidedComp.type==Part and
14:                     collidedComp.hierarchy>currentComponent.hierarchy))) then
15:                       newPVS = currentPVS
16:                       newPVS.add(collidedComp)
17:                       queue.enqueue(newPVS)
18:                       newElementsFound = true
19:                 end if
20:             end for
21:             if (not newElementsFound) then
22:                 resultingPVSs.add(currentPVS)
23:             end if
24:         end while
25:         Create discover variables compGripper, compGround, compPart
26:         for (each PVS in resultingPVSs) do
27:             Check PVS.back().type and save PVS.back() to respective discover
28:             variable (if multiple PVSs fit to one type, the shortest one is considered)
29:         end for
30:         if (compGripper) then
31:             component.partner = compGripper
32:         else if (compGround) then
33:             component.partner = compGround
34:         else if (compPart) then
35:             component.partner = compPart
36:         end if
37:     end if
38: end for
```

separately. A boolean variable is also created to indicate whether new elements have been found. The algorithm will now iterate over all components that have a collision with the current component. If one of these components is not part of the current PVS and is also either of type *Gripper* or *Ground* or of type *Part* and has a higher hierarchy level than the current *Part* component, a new PVS based on the current PVS is created and added to the queue. The boolean variable is also set to true. After iterating over all components that have a collision with the current component, the system checks whether at least one new element has been found. If not, the current PVS is added to the list of resulting PVSs.
When the queue is empty, the next step is to create boolean variables for the component types *Gripper*, *Ground*, and *Part*. These are then used to store whether there are PVSs that end with these component types. If a PVS ends with a *Gripper* component, this component is selected as the partner component. Otherwise, the system checks if a PVS ends with a component of type *Ground*. If so, it is selected as the partner component. If this is also not the case, the system checks whether a PVS ends with a *Part* component and selects this as the partner component. If the current component is not in contact with any other components, the list of resulting PVSs is empty and no partner component is found.
In the following, the steps of the AGPSA to find the PVSs for component P in the *Block Scenario* assembly scenario in Figure 3.15 are considered. Table 3.3 shows all steps. The algorithm starts with component P (step 1). The *Part* component P has a collision with the components O and B. Therefore, two PVSs are created and added to the queue. In step 2, the collided counterparts for the components O and B are examined. Based on the hierarchy levels, there exist two valid counterparts for component O and one for component B. Two

Table 3.3: The steps of the Assembly Graph Partner Search Algorithm to find all PVSs for the *Part* component P in the *Block Scenario* assembly scenario in Figure 3.15. The PVSs found at the end are marked in bold.

Step	Queue	New found PVSs	Resulting PVSs
1	(P)	$(P,\ O)$ $(P,\ B)$	
2	$(P,\ O)$ $(P,\ B)$	$(P,\ O,\ G)$ $(P,\ O,\ B)$ $(P,\ B,\ T)$	
3	$(P,\ O,\ G)$ $(P,\ O,\ B)$ $(P,\ B,\ T)$	$(P,\ O,\ B,\ T)$	$(P,\ O,\ G)$ $(P,\ B,\ T)$
4	$(P,\ O,\ B,\ T)$		$(\boldsymbol{P},\ \boldsymbol{O},\ \boldsymbol{G})$ $(\boldsymbol{P},\ \boldsymbol{B},\ \boldsymbol{T})$ $(\boldsymbol{P},\ \boldsymbol{O},\ \boldsymbol{B},\ \boldsymbol{T})$

PVSs are extended and one new PVS is created. In step 3, the first two PVSs are moved to the resulting PVSs, since no new entries can be added. In addition, the third PVS is extended. Finally, the remaining PVS is moved to the resulting PVSs (step 4). The previous steps refer to lines 3 to 24 of the AGPSA shown in Algorithm 1.
The resulting PVSs are then used to determine a partner component for component P. For this purpose, the last component of each PVS is taken into account. Components of type *Gripper* are preferred over components of type *Ground*, which in turn are preferred over components of type *Part*. See lines 25 to 36 of the AGPSA shown in Algorithm 1 for details. The resulting partner component for component P in the *Block Scenario* assembly scenario in Figure 3.15 is the *Gripper* G.
Based on the partner component found, the **substitute contact force** must now be calculated. The Substitute Force Algorithm from Section 3.4.1 is used and slightly modified for this purpose. A flowchart of the resulting algorithm is shown in Figure 3.16. It calculates the substitute contact force for a component pair. In contrast to the Substitute Force Algorithm extended for manipulation and assembly (see flowchart in Figure 3.11), iterating over the component pairs is used again here, as in the original Substitute Force Algorithm (see flowchart in Figure 3.8).
The SFA based on the AGPSA starts with checking if a partner component has been found for either the *Base* or the *Target* component of the pair and if this partner component matches

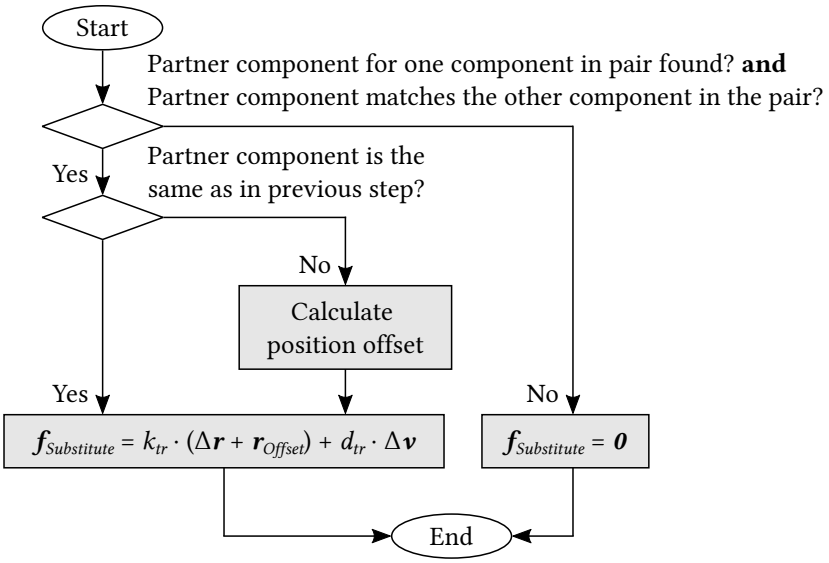

Figure 3.16: A flowchart of the Substitute Force Algorithm based on the Assembly Graph Partner Search Algorithm. Unlike the original SFA (see Figure 3.8), here the substitute contact force calculation is based on whether and which partner component is found. In a similar way, a torque is calculated to constrain the orientation.

the other component of the pair. If no partner component has been found, the substitute contact force is zero. If a partner component has been found, the next step is to check if this partner component has changed since the previous simulation step. If so, an offset is calculated. Then, the substitute contact force is calculated between for the component pair. The Assembly Graph Partner Search Algorithm is now applied to the *Block Scenario* assembly placed on the table T from Figure 3.12. Figure 3.17 shows the substitute contact forces between the components generated with the AGPSA. There are no substitute contact forces connected in series or substitute contact forces connected in parallel, unlike in the original algorithm. All components of type *Part* in the scenario are directly constrained to the *Ground* component T.

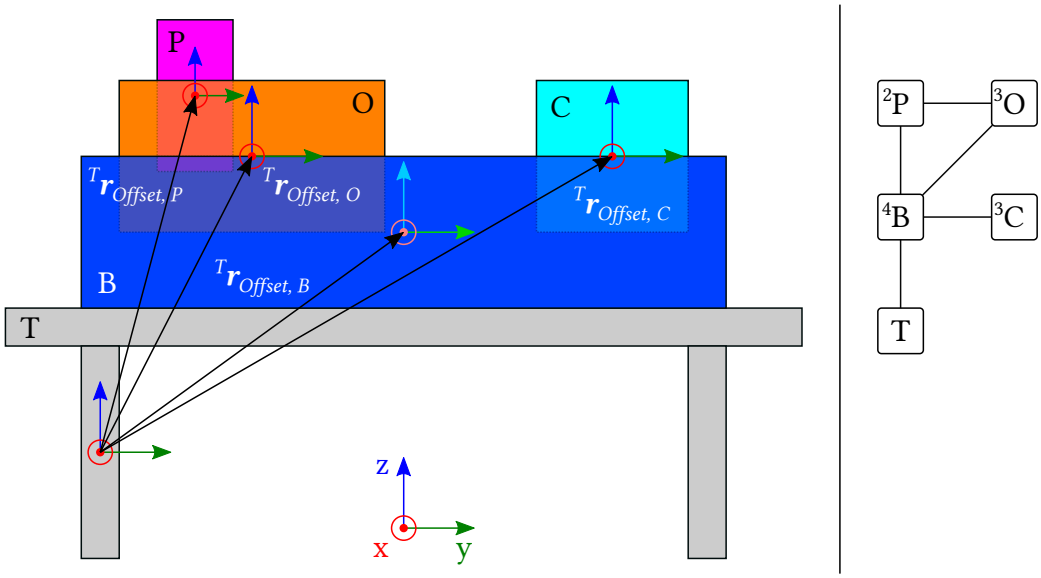

Figure 3.17: The *Block Scenario* assembly placed on the table T from Figure 3.12 now with the Assembly Graph Partner Search Algorithm. The image shows the component frames and the position offsets for all substitute contact forces (left) and the graph of all contacts between components, including the component hierarchies (right).

3.6 Switching between substitute and contact models

In this section, the switching between the fast substitute models and the detailed contact models is described. First, Levels of Detail are defined based on the models and algorithms presented in the previous sections. Then, the switching between the Substitute Force Algorithm and the contact force algorithm based on component models is shown.

3.6.1 Levels of detail

In this section, Levels of Detail (LsoD) are defined, at which the algorithms and models can be applied. There are several ways to define LsoD (see Section 2.4.2). The most obvious distinction for this work is that between the fast substitute models based on substitute contact forces and the detailed models based on contact forces.
Another distinction can be made regarding the robots and grippers in a simulation model. Kinematic chains can be simulated based on kinematics models or rigid body dynamics models. The former are only useful in conjunction with the fast substitute models. The use of contact dynamics models requires dynamics models of the kinematic chains. This is the only way to correctly simulate the influence of contact forces on the robot.
Therefore, three LsoD are defined for this work: fast substitute models based on substitute contact forces in combination with both kinematics models and dynamics models and detailed contact force models in combination with dynamics models. Table 3.4 shows the three LsoD and the algorithms and models used for each.
The "**LoD 1: Kinematics**" is used for the kinematic testing of robot programs before they are applied to the real system. It is based on fast substitute models based on substitute contact forces for the component interaction and kinematics models for the robots and grippers. Simulations at LoD 1 can be used for Virtual Commissioning (see Section 2.4). This includes tasks such as time analysis, program logic testing or collision checking. The dynamics of the system are not considered. Therefore, the components grasped do not affect the robot. Simulations at this LoD are real-time capable for scenarios up to a certain size.

Table 3.4: Levels of Detail and related algorithms and models at which process models can be used. Depending on the LoD, fast substitute contact force models or detailed contact force models are used for component interaction. Both kinematics and dynamics models are available for the robots and grippers.

Levels of Detail	Substitute forces	Contact forces	Robot kinematics	Robot dynamics
LoD 1: Kinematics	x		x	
LoD 2: Dynamics without contacts	x			x
LoD 3: Dynamics with contacts		x		x

The "**LoD 2: Dynamics without contacts**" also uses rigid body dynamics models for the kinematic chains such as robots and grippers. This makes it possible, for example, to simulate the load on the robots. In this case, the load consists of the inertial forces of the grasped components, but does not include any contact forces. Simulations at LoD 2 can be used for example for feasibility studies. Depending on the size of the process model, they can also be used for VC. The reason for the distinction between LoD 1 and LoD 2 is that the computation time for the former is significantly shorter. Models that cannot be computed in real-time at LoD 2 may be able to be computed in real-time at LoD 1.
The "**LoD 3: Dynamics with contacts**" is the most detailed one and is based on contact forces for the component interaction and rigid body dynamics models for the kinematic chains such as robots and grippers. Processes that have already been successfully tested at LoD 1 and LoD 2 can be analyzed in more detail at this level. For example, the load on the robot resulting from the inertia and contact forces of the grasped components can be analyzed here. In addition, simulations at this LoD can be used to optimize processes that depend on contact forces. Simulations at LoD 3 are typically not real-time capable. However, Software in the Loop (SiL) simulations are still possible (see Section 2.4).

3.6.2 Reduced modeling effort through component models

The fast substitute models based on substitute contact forces and the detailed contact force models are combined into the component models introduced in Section 3.2.4. The model of a specific assembly process is built using these component models. Therefore, the process model contains models at all LsoD. However, only the models at one LoD are active.
The process model for the assembly process of the *Block Scenario* example is build with component models, as shown in Figure 3.18. These component models are used for the component types relevant for manipulation and assembly, namely *Part*, *Gripper*, and *Ground* (de-

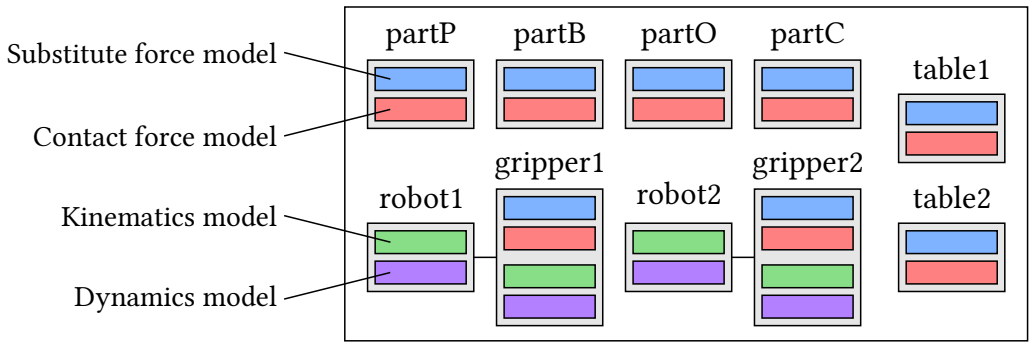

Figure 3.18: The process model for the *Block Scenario* example built with component models for the component types *Part*, *Gripper*, and *Ground*. Component models are also used for the robots, even though they do not interact with the *Part* components.

fined in Section 3.2.1). Each contains both fast substitute contact force models and detailed contact force models. The component model of the *Gripper* also contains a kinematics and dynamics model for the mechanical part. The robots in the process do not interact with the *Part* components. However, component models are still used for them. These models contain both a kinematics and dynamics model of the mechanical structure.
Figure 3.19 shows the models for the simulations at LoD 1 and LoD 3. For the simulation at LoD 1, the fast substitute models based on substitute contact forces are used for the interaction between the components. For all relevant component types, the contact force models are disabled and the substitute models are enabled. In addition, the kinematics models are used for the mechanical parts. This is relevant for the robots and grippers.
In contrast, for the simulation at LoD 3, the detailed contact force models are used for the interaction between the component types *Part*, *Gripper*, and *Ground*. Therefore, the respective models are enabled for these components. Here, the mechanical parts of the robots and grippers are described with dynamics models.
Through the use of component models, the desired Level of Detail for a simulation can be easily changed for the entire model. As a result, the modeling effort is significantly reduced. This is because different simulation models are usually built for different simulation tasks. For instance, a model with detailed contact forces is built for an optimization task. However, these models are usually not real-time capable. Therefore, an additional model must be built for tasks that require real-time capability, such as Virtual Commissioning. By using the component models introduced in this Section, only one process model of a specific manipulation or assembly process needs to be created.

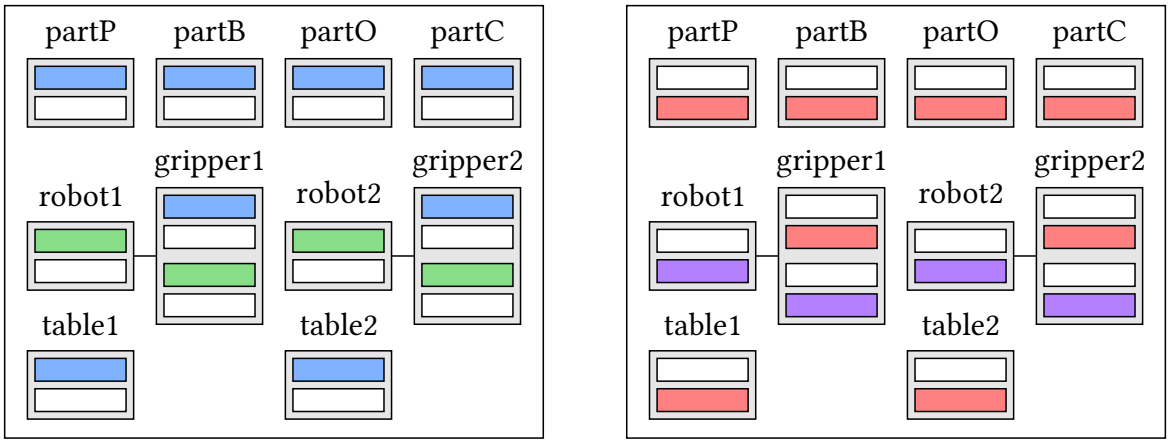

Figure 3.19: The process model of the *Block Scenario* example built with component models. The model for the simulation at LoD 1 is shown on the left, the model for the simulation at LoD 3 on the right. The former uses the substitute contact force models and kinematics models, the latter contact force models and dynamics models.

3.6.3 Process models with multiple levels of detail

The component models introduced in Section 3.6.2 reduce the modeling effort because a process model of a specific use case must be created only once and can then be simulated at multiple Levels of Detail. For example, simulations at LoD 1 are used for fast tests or iterative improvements of the robot programs. In this case, a low computation time is important to enable multiple tests in a short period of time. Simulations at LoD 3 can then be used to further optimize the assembly process, for example.
In the example in Section 3.6.2, the switching between the used models is performed centrally for the entire process model. However, there are use cases where a simulation with multiple Levels of Detail is beneficial. For example, if a scenario contains multiple processes and only one of these processes is optimized. Running the entire simulation at LoD 3 would result in long computation times. In this case, only the relevant process is simulated at LoD 3, while the other processes are simulated at LoD 1.
An example is illustrated in Figure 3.20. The scenario consists of three components of type *Gripper* (all attached to robots), seven components of type *Part* and three components of type *Ground*. There are two independent processes. One process comprises robot3, gripper3, part5, part6, part7, and table3. The other process comprises the remaining components. Figure 3.20 shows the process model for a simulation at multiple LsoD. The process that is optimized is simulated at LoD 3 and the other process at LoD 1. Simulations with multiple Levels of Detail require independent manipulation and assembly processes. However, tasks such as collision checks can still be performed for the entire model across all processes.

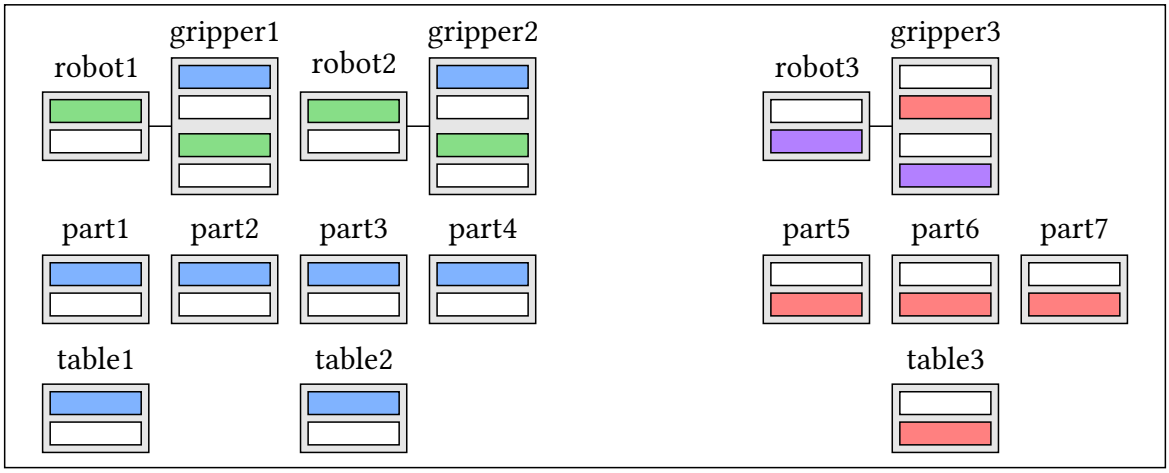

Figure 3.20: A process model for a scenario containing two processes. The process on the left is simulated at LoD 1, and the process on the right is simulated at LoD 3. This significantly reduces the computation time for simulations that only require LoD 3 for parts of the scenario.

4 Realization

In this chapter, the realization of the solution developed in Chapter 3 is presented. First, an overview of the developed software framework is shown. Then, the integration of the *libccd* package (including the MPR algorithm) and the Polygonal Contact Model algorithm is discussed. Furthermore, the interaction between *Modelica* models and the External Environment is discussed and optimizations to improve the computation time are presented.

4.1 Overview of the developed software framework

In this section, the developed software framework is presented. An overview of the developed software is given in Figure 4.1. The modeling language *Modelica* is used as system simulation environment (see Section 2.3.1). The figure shows a *Modelica* model on the left. It

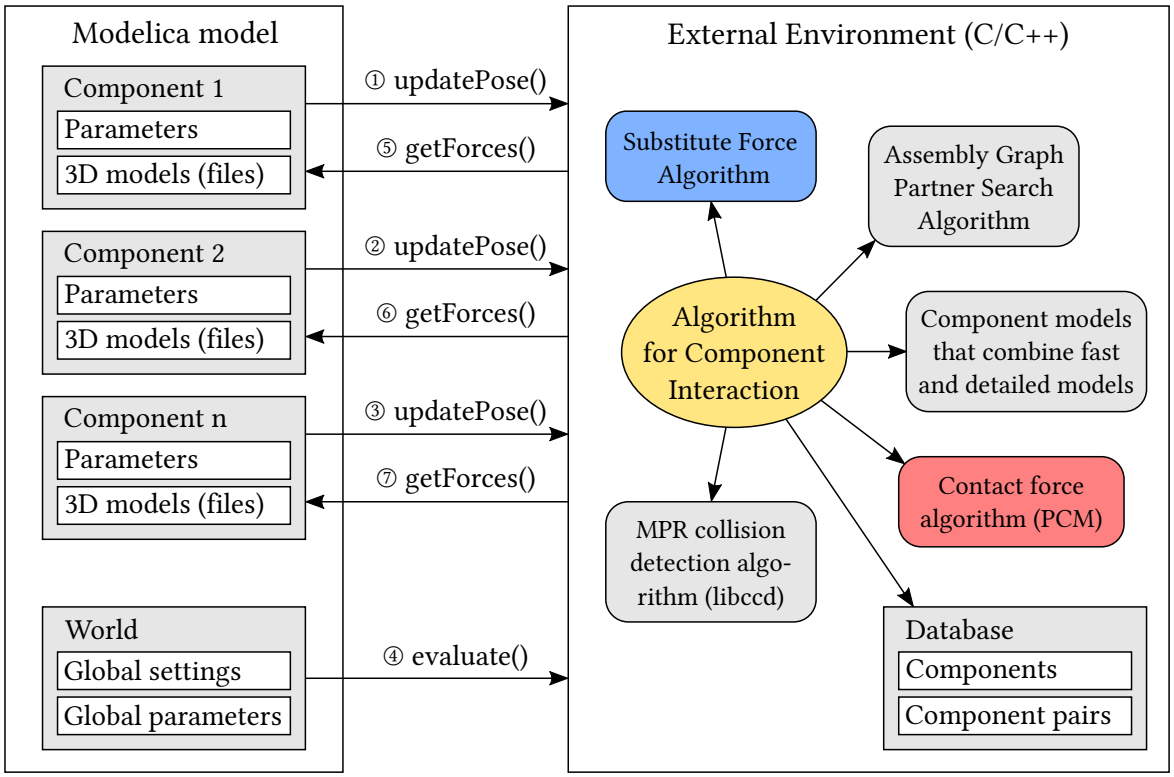

Figure 4.1: Overview of the developed software framework. The left side shows a *Modelica* model with several components and a World object. In the External Environment, the ACI calls various algorithms presented in Chapter 3, depending on the task at hand (right). The circled numbers indicate the order of the function calls.

contains several components and a World object. This model is created by the user to model a specific manipulation or assembly process (an example is the process model in Figure 3.6). The user can define parameters and specify 3D models for each component (the filenames are specified for this purpose). Global settings and parameters are defined in the World object. Here, it is possible to specify whether the fast substitute models or the detailed contact models are used.
This *Modelica* model is extended by an **External Environment (EE)**, shown on the right. It is developed in the C and C++ programming languages. The core of the environment is the Algorithm for Component Interaction. The ACI uses several algorithms depending on the respective task and uses a database containing all components and component pairs. Both algorithms developed within this work (introduced in Chapter 3) and existing algorithms (such as MPR and PCM) are used. Figure 4.1 also shows the interaction between the *Modelica* model and the EE. The sequence within a simulation step is as follows:

- First, all components in the *Modelica* model update their position and orientation in the External Environment (function calls 1 to 3 in Figure 4.1).
- Then, the evaluation is executed once centrally for the entire *Modelica* model (the function call is made in the World object, see function call 4 in Figure 4.1).
- Finally, all components in the *Modelica* model get their resultant force calculated in the External Environment (function calls 5 to 7 in Figure 4.1).

Figure 4.2 shows in more detail the sequence of the interaction between a *Modelica* model and the External Environment. Once all components have updated their position and orientation in the EE, the evaluation process in the EE is started. The evaluation starts with a global collision check. For this purpose, a collision check is performed for each component pair using the MPR algorithm (see Section 2.1.1).
The next step depends on whether the detailed contact models or the fast substitute models based on substitute contact forces are used (this is set in the World object). In the former case, the PCM algorithm (see Section 2.2.2) is used to calculate the contact forces for all component pairs. In the latter case, the system first checks whether the Assembly Graph Partner Search Algorithm (see Section 3.5.2) should be used (this is also set in the World object). If this is the case, it is executed. The substitute contact forces are then calculated for all component pairs based on the Substitute Force Algorithm (see Section 3.4.1). In both cases, the resulting force for all components is then calculated from the forces of the component pairs. This is done using the Algorithm for Component Interaction (see Section 3.2.5).
The forces calculated in the External Environment are now updated in the *Modelica* model for all components. In the *Modelica* model, these forces are applied to the frames as external forces and combined with other forces from within the model. In the case of the gripper, for example, these include the forces that press the gripper jaws together. All frames are in mechanical equilibrium, i.e. the sum of the forces is zero.

The *Modelica* model is then solved using integration methods for ODEs (see solvers in Section 3.3.2) based on the combined forces in the model. This completes the simulation step. The simulation time is then increased by the step size Δt. The previous sequence is then executed for the next simulation step.

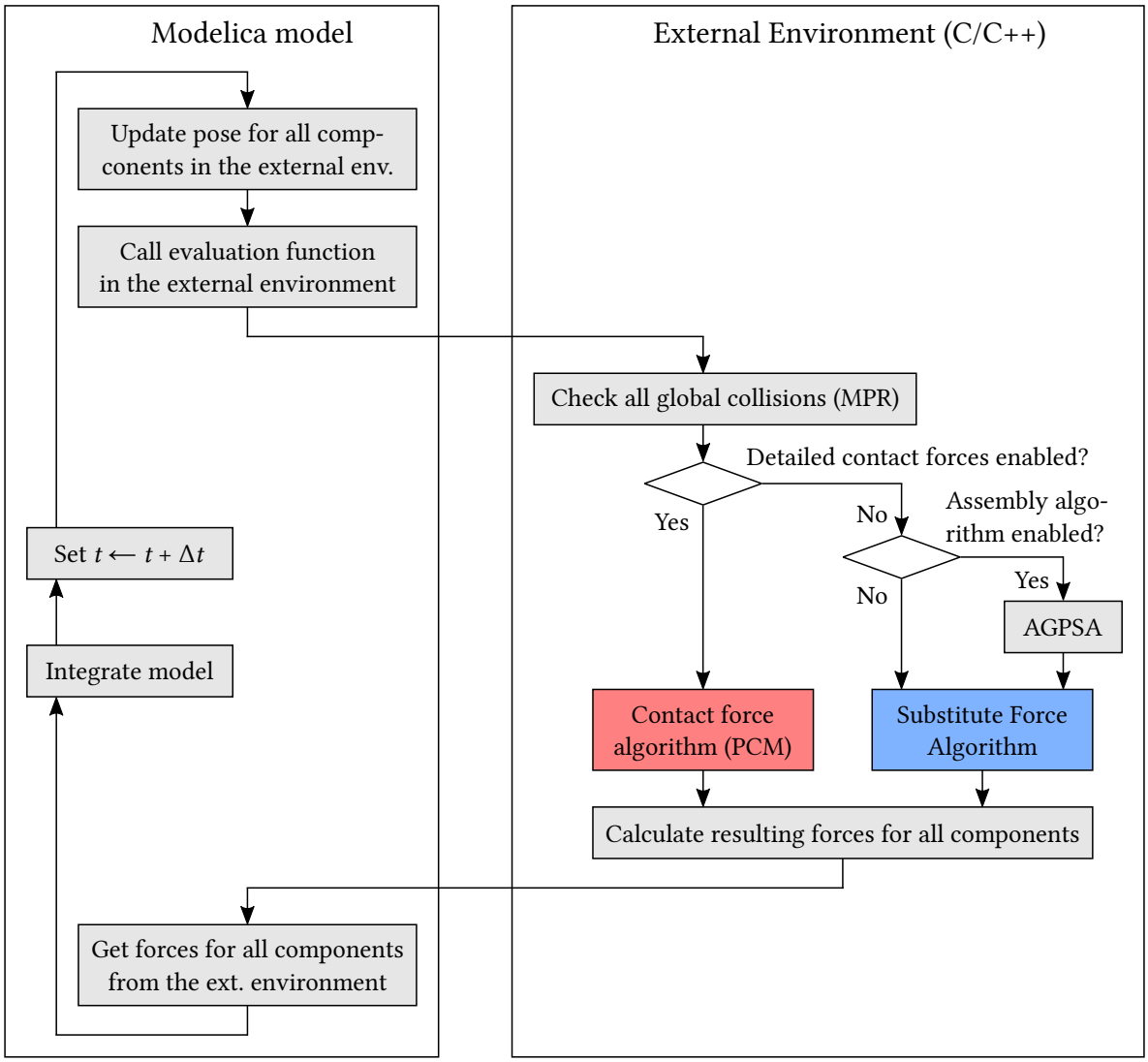

Figure 4.2: A flowchart showing the interaction between a *Modelica* model and the External Environment during the simulation runtime. The figure also shows the sequence for integrating the model and increasing the time t by the step size Δt.

4.2 libccd integration

The open source project *libccd* (Fiser 2018) provides an implementation of the Minkowski Portal Refinement (MPR) algorithm for collision detection (see Section 2.1.1). The MPR algorithm in *libccd* is used by creating support functions for specific shapes. In the work of Buse et al. (2023), support functions were created for primitives such as cube or sphere and for polygonal meshes based on the OBJ file type. The latter are used in this work.
Figure 4.3 shows the collision checking sequence for a component pair consisting of *Base* and *Target* based on the MPR algorithm. First, an AABB check is performed to see if the BVs overlap. This check is very fast and can significantly improve the computation time if the BVs do not overlap. This step represents the *broad phase* (see Section 2.1.1). If the check is positive, the process continues, otherwise there is no collision for the pair.
The MPR algorithm could then be called directly to check for overlap between the two components. Instead, all combinations of the convex parts of both components are checked for collision to allow convex decomposition. If no convex decomposition is used, both components consist of only one part, which is interpreted as convex. This second step is the *narrow phase* of the collision detection process.

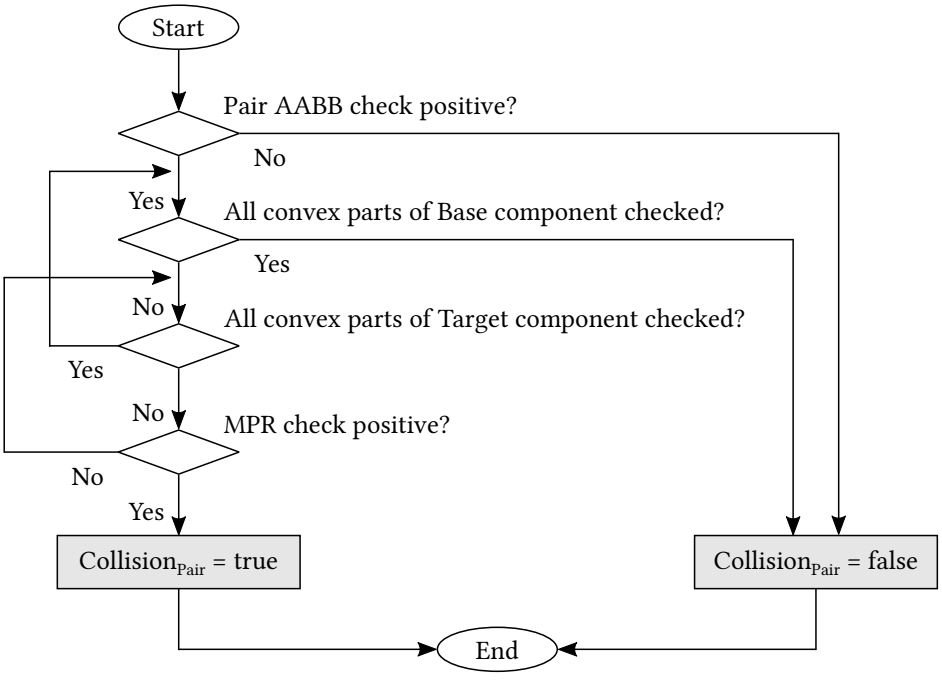

Figure 4.3: A flowchart showing the collision check for one component pair using the MPR algorithm. First, an AABB check is performed (*broad phase*). Then, all combinations of the convex parts of both components are checked for collisions using the MPR algorithm (*narrow phase*). This allows the use of convex decomposition.

The MPR implementation in the *libccd* project provides two functions (Fiser 2018):

- `ccdMPRIntersect()`
- `ccdMPRPenetration()`

The first function checks if two components overlap. It is sufficient for the substitute force algorithms presented in this work as they only require the speed and position of the components. The second also calculates the contact point, the contact normal and the penetration depth. This functionality can be used for advanced collision checking in simulations.
The *libccd* framework also provides an implementation of the Gilbert-Johnson-Keerthi (GJK) algorithm. It can be used to compute the shortest distance between two bodies even if there is no collision. In this work, the GJK algorithm is not used for the substitute force algorithms. However, it can be used for pure collision checking and offers advantages here, such as checking if a minimum distance between components is satisfied during a movement.

4.3 PCM integration

The Polygonal Contact Model (PCM) is open source software (Hippmann 2004b) with the source code provided on the website[1]. It is designed to be integrated into multibody environments. In a multibody scenario, a PCM contact element behaves like a force element. All components must be described as polygonal meshes in the OBJ file format.
The PCM is typically used in a multibody environment by defining PCM contacts between bodies in the model. That is, the modeler manually defines PCM contact pairs. This is not sufficient for the modeling and simulation of manipulation and assembly processes. Instead, in this work, all contact pairs are generated automatically.

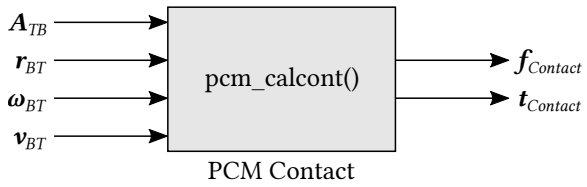

Figure 4.4: Inputs and outputs for the PCM contact block.

A PCM contact block is shown in Figure 4.4. The inputs include the transformation matrix from the *Target* to the *Base* component $\boldsymbol{A}_{TB}$ as well as the relative position $\boldsymbol{r}_{BT}$, the angular velocity $\boldsymbol{\omega}_{BT}$, and the velocity $\boldsymbol{v}_{BT}$ from the *Base* to the *Target* component. The block returns the contact force $\boldsymbol{f}_{Contact}$ and torque $\boldsymbol{t}_{Contact}$ acting on both components. The corresponding function is `pcm_calcont()`.

[1] `http://www.pcm.hippmann.org/`

Algorithm 2 A pseudocode showing the calculation of the contact force and torque for one component pair (consisting of *Base* and *Target* component) based on the PCM algorithm.

Require: Position, orientation, and (angular) velocity of the $Base$ and $Target$ component

1: Create variable $aabbIntersection$
2: $aabbIntersection$ = checkAabbIntersection($Base$, $Target$)
3: **if** ($aabbIntersection$) **then**
4: Create input variables $\boldsymbol{A}_{TB}$, $\boldsymbol{r}_{BT}$, $\boldsymbol{\omega}_{BT}$, and $\boldsymbol{v}_{BT}$
5: Create output variables $\boldsymbol{f}_{Contact}$ and $\boldsymbol{t}_{Contact}$
6: $\boldsymbol{A}_{TB} = \boldsymbol{T}_{Base} \cdot \boldsymbol{T}_{Target}^{T}$
7: $\boldsymbol{r}_{BT} = \boldsymbol{T}_{Base} \cdot (\boldsymbol{r}_{Target} - \boldsymbol{r}_{Base})$
8: $\boldsymbol{\omega}_{BT} = \boldsymbol{T}_{Base} \cdot (\boldsymbol{\omega}_{Target} - \boldsymbol{\omega}_{Base})$
9: $\boldsymbol{v}_{BT} = \boldsymbol{T}_{Base} \cdot (\boldsymbol{v}_{Target} - \boldsymbol{v}_{Base} - (\boldsymbol{\omega}_{Base} \times (\boldsymbol{r}_{Target} - \boldsymbol{r}_{Base})))$
10: $(\boldsymbol{f}_{Contact}, \boldsymbol{t}_{Contact})$ = pcm_calcont($\boldsymbol{A}_{TB}$, $\boldsymbol{r}_{BT}$, $\boldsymbol{\omega}_{BT}$, $\boldsymbol{v}_{BT}$)
11: $Base.force = \boldsymbol{T}_{Base}^{T} \cdot \boldsymbol{f}_{Contact}$
12: $Base.torque = \boldsymbol{t}_{Contact} + (\boldsymbol{r}_{BT} \times \boldsymbol{f}_{Contact})$
13: $Target.force = -1 \cdot \boldsymbol{T}_{Target}^{T} \cdot (\boldsymbol{A}_{TB}^{T} \cdot \boldsymbol{f}_{Contact})$
14: $Target.torque = -1 \cdot \boldsymbol{T}_{Target}^{T} \cdot (\boldsymbol{A}_{TB}^{T} \cdot \boldsymbol{t}_{Contact})$
15: **else**
16: $Base.force = (0, 0, 0)$
17: $Base.torque = (0, 0, 0)$
18: $Target.force = (0, 0, 0)$
19: $Target.torque = (0, 0, 0)$
20: **end if**

The calculation of the contact force and torque for one component pair with the PCM algorithm (integrated into the External Environment) is shown in Algorithm 2. Similar to the collision check with the MPR algorithm (see Figure 4.3), an AABB check is performed in the first step (see line 2). If it is negative, the components do not intersect and the contact force and torque is zero (see lines 16 to 19).
If the AABB check is positive, the next step is to calculate the inputs for the PCM contact function based on the position and velocity data of the two components (see lines 6 to 9). This step is not necessary for contacts that are predefined in the model, since the relative position and velocity between the two components can already be calculated in the model. The PCM contact is then calculated (see line 10).
Finally, the resulting contact force and torque for the *Base* and *Target* component is calculated based on the PCM result (see lines 11 to 14). All results are transformed to the world frame. This is necessary because each component may have multiple contact forces that need to be summed in the Algorithm for Component Interaction.

4.4 Interaction with Modelica

The realization of the solution developed in Chapter 3 is presented in Section 4.1. The developed software framework consists of a *Modelica* model that is extended by an External Environment (EE). Figure 4.1 shows the interaction between the model and the EE.
However, the sequence of function calls shown in Figure 4.1 is not possible without additional modifications of the *Modelica* model. This is because the order of external function calls is not specified in the *Modelica* language standard (Modelica Association 2017). Depending on the tool (*Dymola*, *SimulationX*, *OpenModelica*), the external functions might be called in the order in which the components are added to the model. For example, if O is added to the model in Figure 3.6 first, then C and P, then the World object, and finally B, the external functions are called in that order. This would mean that the evaluate function in the World object is called before the data from component B has been updated in the EE. But even if the order of the components is changed so that the World object is added to the model last, another problem arises: all components update their data in the EE and retrieve the resulting forces from the EE before the evaluate function is called. Thus, the forces are those calculated in the previous simulation step.
For this reason, a **method to call the external functions in the correct order** is implemented in the *Modelica* component models that are used to create process models (see Figure 3.6). This method is presented in Elmqvist et al. (2015). In principle, it is a clever combination of the inner/outer structure and flow variables of the connector. In detail, the method works as follows.

- The components in the *Modelica* model contain a dummy variable that gets a dummy response from calling the update function: $compUpdated = \text{update}(\boldsymbol{r},\ \boldsymbol{T},\ \boldsymbol{v},\ \boldsymbol{\omega})$.
- This dummy variable is equated to the flow variable of a connector of this component, which in turn is connected to the flow variable of the World object.
- The World object is integrated into the components as an outer object and contains an algorithm that defines the order:
  - The dummy variable $allCompsUpdated$ is equated with the flow variable of the connector of the World object.
  - Then, the function evaluate() is executed.
  - Another dummy variable $evaluationDone$ is set with a dummy value.
- In the components, the results are retrieved from the EE based on the following condition: $resultVector = \text{getResult}() + World.evaluationDone$.
- Finally, the resulting force and torque is extracted from the $resultVector$ variable.

This method ensures that the following sequence is followed. First, the `update()` function is executed in all components. The order in which the `update()` functions are executed is

not defined, but it is also not relevant. Then, the `evaluate()` function is executed centrally in the World object and the calculations are performed in the EE. Finally, the `getResult()` function is executed in all components. Again, the order is not defined and not relevant.
The developed software framework provides a *Modelica* library for the user. The user does not interact with the External Environment. The library is a **component library** with components that can be used by the user to build a model for a specific process. The following components are provided by the library:

- ManipulatedComponent
- ManipulationWorld
- Predefined grippers
- Ground components such as tables

The **ManipulatedComponent** is the central component that can be used to model *Part*, *Ground*, and *Gripper* components. It has the following parameters (see also Section 3.2.4):

- Associated 3D models:
  - Detailed mesh for the PCM algorithm (OBJ file)
  - Convex hull or convex decomposition for the MPR algorithm (OBJ file)
  - Optional file for visualization with texture (OBJ, STL, or GLB file)
- Component type (see Section 3.2.2)
- Hierarchy level (see Section 3.4.2)
- Component specific parameters for the SFA, AGPSA, PCM, and MPR algorithm

The **ManipulationWorld** object is used to set parameters for the entire model. The following parameters can be set by the user.

- Enable or disable the substitute force or contact force
- Selection of the substitute force algorithm: SFA, extended SFA, or AGPSA
- Optimization of the external function calls (see Section 4.5)
- Parallelization of the calculation of the PCM contacts (see Section 4.5)
- Global parameters for the SFA, AGPSA, PCM, and MPR algorithm

In addition, the library provides predefined grippers and ground components such as tables. Both are based on the *ManipulatedComponent*. The interaction with other *Modelica* libraries such as the *DLR Robots library* (Bellmann et al. 2020) and the *DLR Visualization 2 Library* (Kümper et al. 2021) takes place at the level of the *Modelica* model. Components from all libraries can be combined in *Modelica* models.

4.5 Optimizations to improve the computation time

The developed software framework (see Figure 4.1 in Section 4.1) is optimized in several aspects. In this section, two major aspects are presented: the optimization of function calls to improve the computation time and the parallelization of the PCM algorithm.
The motivation for the **optimization of the function calls** is as follows. Many solvers evaluate the system of equations not only for the current time step, but also for intermediate steps. In addition, the equations are sometimes evaluated multiple times for the same time step. As a result, the computationally intensive MPR algorithm or the PCM algorithm may be called multiple times with the same input for a component pair.
Optimization of the function calls prevents these algorithms from being called more than once in a row with the same input. For this purpose, the `update()` function, which is called for each component of the *Modelica* model, checks whether the data has changed. If it has, the new values for position, orientation, velocity, and angular velocity are updated in the EE and the *hasMoved* variable is set to true for this component. The function call optimization can be disabled using the $optimizeFunctionCalls$ variable in the world object. The `update()` function is shown in Algorithm 3.

Algorithm 3 A pseudocode showing the `update()` function called by all components in the *Modelica* model. The function checks if the data of a component has changed.

Require: $component$ (pointer), $position$, $orientation$, $velocity$, and $angularVelocity$
1: **if** ($world.optimizeFunctionCalls = true$ **and**
2: equal($component.position$, $position$) **and**
3: equal($component.orientation$, $orientation$) **and**
4: equal($component.velocity$, $velocity$) **and**
5: equal($component.angularVelocity$, $angularVelocity$)) **then**
6: // Do nothing
7: **else**
8: $component.hasMoved = true$
9: $component.position = position$
10: $component.orientation = orientation$
11: $component.velocity = velocity$
12: $component.angularVelocity = angularVelocity$
13: **end if**

The process in the `evaluate()` function is then as follows (see also Section 3.2.5). First, all component pairs are analyzed. If at least one of the two components has moved (*Base* or *Target*), the evaluation is performed. This includes the collision check based on the MPR algorithm and, depending on the specified Level of Detail, the calculation of the substitute force or the contact force. If neither of the two components has moved, the evaluation is skipped. The `getResult()` function will then return the result of the previous step which is still stored in the EE. Then, the components are analyzed. For each component, the re-

sulting force is calculated as sum of the forces from all related component pairs. Finally, the *hasMoved* flag is reset for all components. The process is shown in Algorithm 4.

Algorithm 4 A pseudocode showing the evaluation sequence when calling the `evaluate()` function from the World object in the *Modelica* model.

Require: *componentList* and *componentPairList* (pointers)
for (each *componentPair* in *componentPairList*) **do**
  if (*componentPair.Base.hasMoved* **or**
    componentPair.Target.hasMoved) **then**
      Create variable *aabbIntersection*
      aabbIntersection = checkAabbIntersection(*Base*, *Target*)
      if (*aabbIntersection*) **then**
        Check component pair collision with the MPR algorithm
        if *world.useContactForces* **then**
          Calculate contact force based on the PCM algorithm
        else
          Calculate the substitute contact force based on the SFA
        end if
      else
        // Skip component pair evaluation
      end if
  else
      // Skip component pair evaluation
  end if
end for
for (each *component* in *componentList*) **do**
  Calculate force sum from all related component pairs
  component.hasMoved = *false*
end for

This optimization of function calls can cut the computation times for simulations based on contact forces in half. A detailed analysis of the computation times for the *Block Scenario* example simulation is given in Section 5.2.4.
Another optimization is the **parallelization of the PCM contact calculation**. The PCM algorithm is based on component pairs. For each component pair, one PCM contact is created. All PCM contacts can be calculated independently and deterministically (a contact force is calculated from a relative position, orientation, and velocity). This makes the PCM algorithm thread-safe. The optimized PCM works as follows. All component pairs are analyzed, and if both components have moved and the AABB check is positive, the pair is added to a list. Then, the `pcm_calcont()` function is executed in parallel for this list based on functions of the C++ standard library. This parallelization can also cut the computation times for detailed simulations in half. More details are given in Section 5.2.4.

5 Applications and results

The method for simulating assembly processes in structure-invariant environments developed in Chapter 3 and realized in Chapter 4 is now applied to different use cases.
First, the assembly process for the *Block Scenario* example introduced in Section 3.1 is simulated and the results are presented. This example covers several aspects: the components interact with multiple robots, grippers, and grounds and subassemblies are involved. Based on this example, the suitability for simulating disassembly processes is also demonstrated. Furthermore, various analyses are performed based on the *Block Scenario* example. And the example is simulated with another robot type with detailed models of the drivetrain.
Finally, other use cases are presented. One example is the assembly process of an electric motor into the housing of a chainsaw. It demonstrates the suitability to simulate use cases involving complex, non-convex geometries. Another is the multi-domain simulation of the disassembly process of an electric motor, including a furnace heating process.
In addition to the use cases presented in the following sections, the method was applied to two use cases in the automation of aircraft production, namely in the projects *MFlex 2025* (Reiser et al. 2022; Reiser 2025) and *ProDigiES* (Koch et al. 2025).
All simulations are performed using *Dymola 2025x* on a *Windows 11* workstation with an *Intel Core i7-11700K* processor. The average of five simulations is used to calculate the computation times. The average real-time factor is defined as the computation time to simulation time ratio. For instance, if a simulation runs in half the time, the average real-time factor is 0.50. For the purpose of this work, "real-time" is defined as a statistical adherence to time constraints, also known as *firm real-time* or *soft real-time*. All robots are modeled with the *DLR Robots library* (Bellmann et al. 2020). This library provides an interpreter for programming robots, including opening and closing commands for the grippers. The interpreter uses the *Modelica Lua library* presented in Buse and Bellmann (2021). However, the robot axes angles and gripper commands are generated in advance in this work for a better comparability, because the interpreter creates additional events in the model. All visualizations are based on the *DLR Visualization 2 Library* (Kümper et al. 2021). Components of the *Modelica Standard Library (MSL)* (Modelica Association 2020) are used for other parts of the models.

5.1 Simulation of the block scenario example

The *Block Scenario* example is simulated in this section. Both the assembly process presented in Section 3.1 and a disassembly process are considered.

5.1.1 Overview and setup

The *Block Scenario* example is presented in Section 3.1. It consists of the four *Part* components P, C, O, and B. The scenario also contains two *Ground* components, Table1 and Table2, two

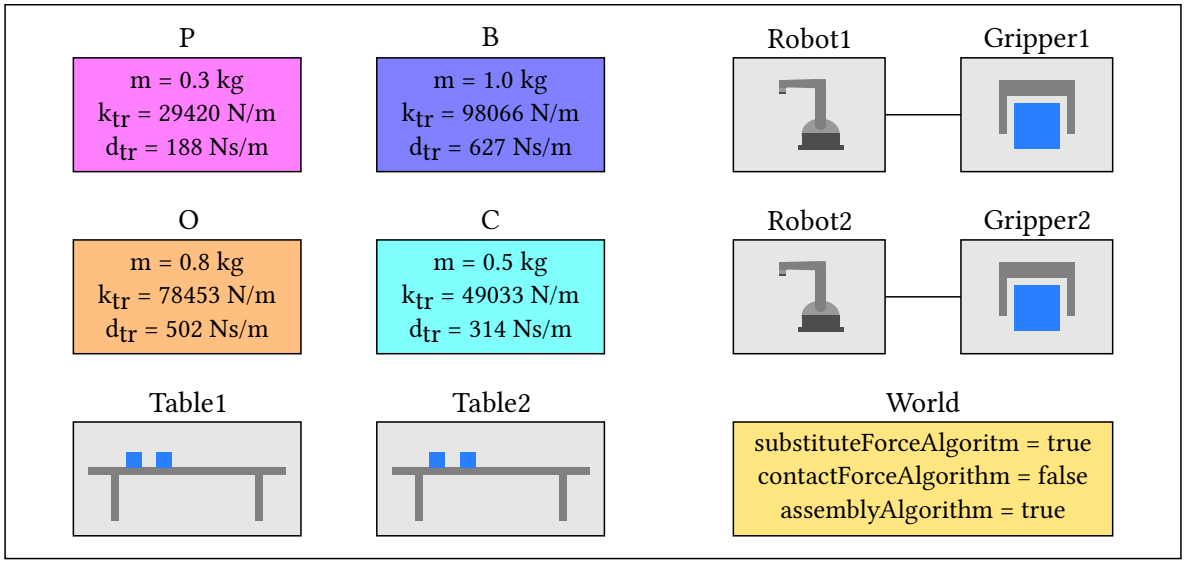

Figure 5.1: The process model of the *Block Scenario* example. It consists of the *Part* components P, C, O, and B, the *Gripper* components Gripper1 and Gripper2 (both attached to robots) and two *Ground* components Table1 and Table2. Global parameters are defined and algorithms are enabled in the World object.

robots, and two *Gripper* components, Gripper1 and Gripper2. Figure 5.1 shows the associated process model. This model is a more detailed version of the model presented in Figure 3.6. It contains all the components mentioned above and selected parameters for some of the components. The World object is used to define global parameters. The algorithms can also be activated here. The **simulation time** is 42 s. The robots are modeled with the *DLR Robots library* and the *DLR Visualization 2 Library* is used for the visualization of the assembly process. For the robots, simplified models of the drivetrains and controls are used. The remaining parts of the model are components of the *MSL*.
The parameters for the components of type *Part* are specified by the model creator. Only selected parameters, mostly for component P, are shown. The masses are defined as follows:

$$m_P = 0.3 \text{ kg} \tag{5.1}$$
$$m_C = 0.5 \text{ kg} \tag{5.2}$$
$$m_O = 0.8 \text{ kg} \tag{5.3}$$
$$m_B = 1.0 \text{ kg}. \tag{5.4}$$

Next, the **parameters for the Substitute Force Algorithm** are specified by the model creator. These can be calculated based on a given maximum position error. For the example, a maximum static position error Δr_{max} of one-tenth of a millimeter is specified:

$$\Delta r_{max} = 0.0001 \text{ m}. \tag{5.5}$$

The translational spring constant for component P, $k_{tr,\,P}$, can be calculated by equating the weight force and the spring force (similar to Equation (3.21)):

$$f_{s,\,P} = f_{g,\,P} \tag{5.6}$$

$$\Rightarrow \Delta r_{max} \cdot k_{tr,\,P} = m_P \cdot g \tag{5.7}$$

$$\Rightarrow k_{tr,\,P} = \frac{m_P \cdot g}{\Delta r_{max}} = \frac{0.3\ \text{kg} \cdot 9.80665\ \frac{\text{m}}{\text{s}^2}}{0.0001\ \text{m}} = 29419.95\ \frac{\text{N}}{\text{m}}. \tag{5.8}$$

The Equations (A.7) to (A.12) are used to calculate the translational damping constant $d_{tr,\,P}$ (see Section A.1.3): The desired position should be reached as quickly as possible while avoiding oscillations. For this reason, critical damping is used. The degree of damping D_P is therefore $D_P = 1$. The parameter $d_{tr,\,P}$ can now be calculated using Equation (A.12):

$$D_P = \frac{d_{tr,\,P}}{2 \cdot \sqrt{m_P \cdot k_{tr,\,P}}} = 1 \tag{5.9}$$

$$\Rightarrow d_{tr,\,P} = 2 \cdot \sqrt{m_P \cdot k_{tr,\,P}} = 2 \cdot \sqrt{0.3\ \text{kg} \cdot 29419.95\ \frac{\text{N}}{\text{m}}} \approx 187.89\ \frac{\text{Ns}}{\text{m}}. \tag{5.10}$$

The translational spring constants and damping constants for the other components are calculated in the same way. Furthermore, the rotational constants for the substitute contact force calculation are selected for all components.
In addition, the **parameters for the contact force algorithm** (PCM) are defined by the model creator. The following contact parameters are used globally for all components:

$$E = 1 \cdot 10^6\ \text{Pa} \tag{5.11}$$

$$\nu = 0.3 \tag{5.12}$$

$$\mu = 0.3 \tag{5.13}$$

$$v_\epsilon = 0.001\ \frac{\text{m}}{\text{s}} \tag{5.14}$$

$$d = 0.001\ \text{m} \tag{5.15}$$

where E is the Young's modulus, ν the Poisson's ratio, μ the coefficient of friction and v_ϵ the friction regularization velocity, and d the layer thickness. The last two parameters are different for the grippers:

$$\mu_{Gripper} = 0.90 \tag{5.16}$$

$$v_{\epsilon,\,Gripper} = 0.0001\ \frac{\text{m}}{\text{s}}. \tag{5.17}$$

These two parameters apply to all pairs consisting of a *Part* component and a *Gripper* component (i.e. for the interactions between the gripper jaws and the grasped components).
The process model shown in Figure 5.1 can now be used at different **Levels of Detail**. The Levels of Detail (LsoD) defined in Section 3.6.1 are used for this purpose.

5.1.2 Simulation results

For the *Block Scenario* assembly process (see Section 3.1), a **LoD 1 (kinematics)** simulation (see Table 3.4) is run first. The simulation is based on substitute contact forces and kinematics models for the robots. The parameters defined in Section 5.1.1 are applied. A second order Runge-Kutta solver, namely *Rkfix2*, with a fixed step size of 0.001 s is used. In comparison to an Euler or third/forth order Runge-Kutta solver, this solver has empirically demonstrated a good balance between accuracy and computation time for the developed solution.
The simulation was successfully completed. A visualization for multiple steps during the simulation is shown in Figure 5.2. The simulation starts with all four *Part* components placed on Table1 (see Section 3.1). At the simulation time 9.8 s, P is assembled into O by Gripper1 and C is about to be assembled into B by Gripper2. The subassembly consisting of P and O is about to be assembled into B at 20.5 s and the assembly is fully assembled at 27 s. Finally, the entire assembly is moved to Table2 using Gripper2. These steps are identical to those presented in Figure 3.2. The computation time was 4.00 s for a simulation time of 42 s. This results in an average real-time factor of 0.095. The simulation is therefore real-time capable and can be used for applications such as Virtual Commissioning.

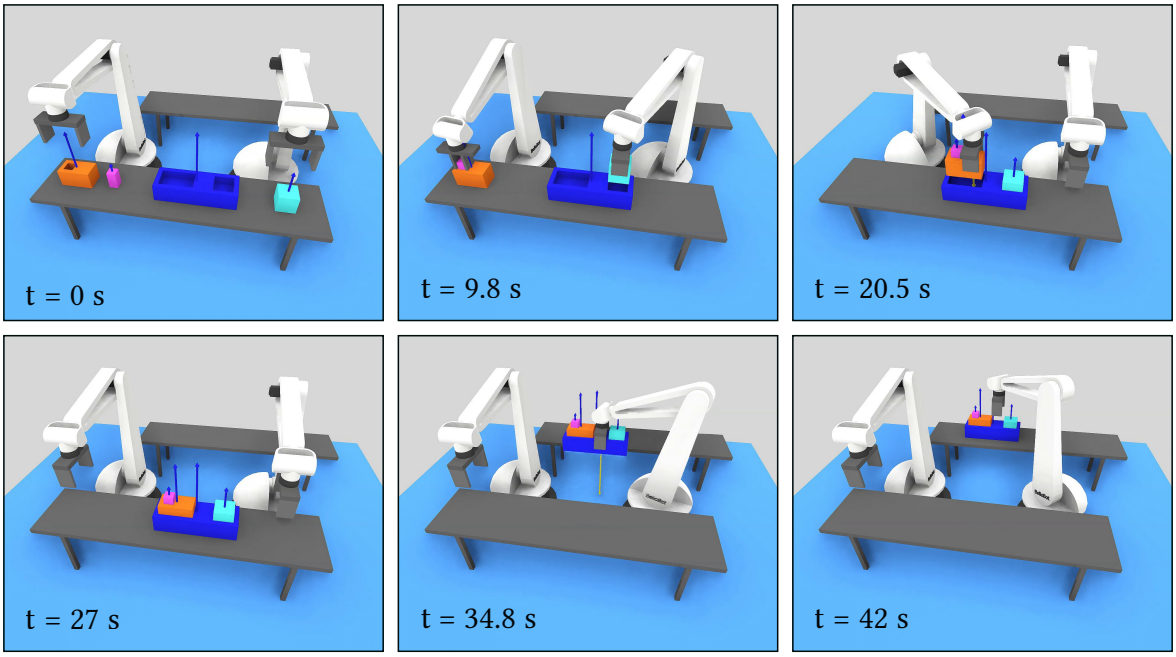

Figure 5.2: Visualization of the *Block Scenario* assembly process simulation for multiple steps. The simulation uses substitute forces and robot kinematics models (LoD 1). A *Rkfix2* solver with a fixed step size of 0.001 s is used. The simulation time is 42 s and the computation time is 4.00 s (average real-time factor of 0.095). The arrows represent the resulting forces for the *Part* (blue) and *Gripper* (yellow) components.

Next, a **LoD 2 (dynamics without contacts)** simulation (see Table 3.4) is performed. Unlike the *LoD 1* simulation, this one uses dynamics models of the robots. The *Rkfix2* solver is used again. However, the fixed step size must be reduced to 0.0002 s in order to maintain stability regarding the dynamics models of the robots. This simulation was also completed successfully. The simulation of the model took 20.4 s. The resulting average real-time factor is 0.49. The simulation runs in real-time, so it is still suitable for VC.
Finally, a **LoD 3 (dynamics with contacts)** simulation is run. The detailed contact forces based on the PCM algorithm are now used for the interactions between components. The visualization for multiple steps of the successful simulation is shown in Figure 5.3.
Here, the variable-step solver *DASSL* (Petzold 1982) with $2 \cdot 10^{-6}$ tolerance is used. A variable-step solver is used because the contact models require a small step size at the beginning or end of a contact or when multiple *Part* components are interacting, but a larger step size is sufficient otherwise. The *DASSL* solver showed in many applications empirical stability and is well-suited for system simulations because it works reliable across multiple domains.
The simulations were also successfully run with similar tolerances using the *Cvode* solver (Hindmarsh et al. 2005) and the *Esdirk45a* solver (Kennedy and Carpenter 2019). Here, the

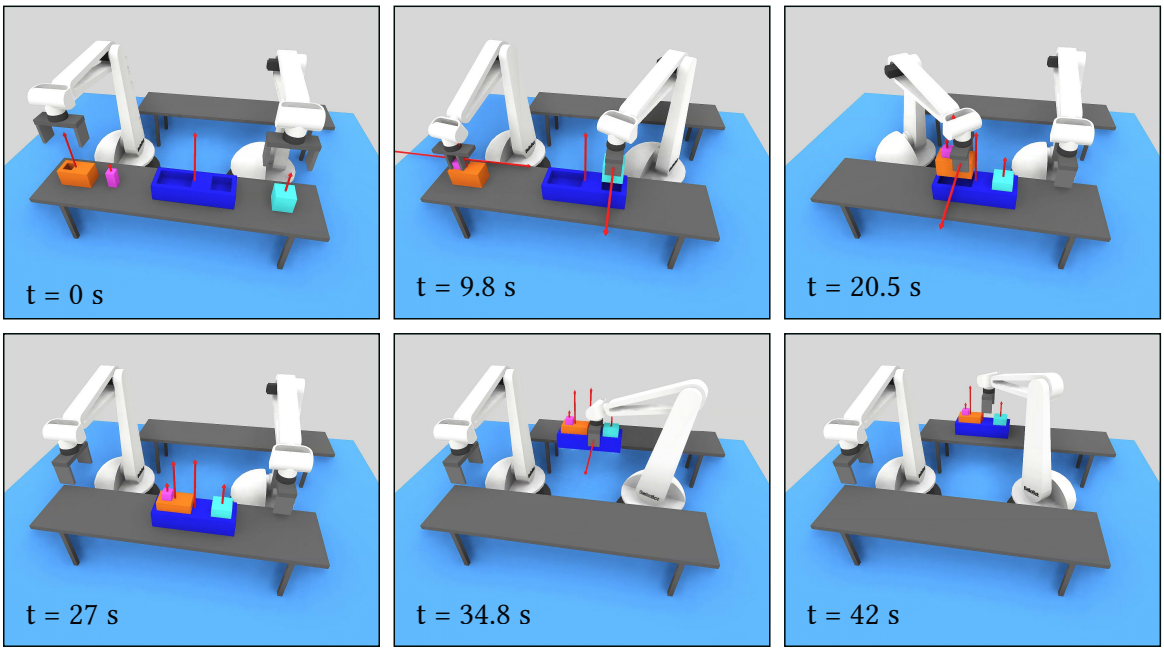

Figure 5.3: Visualization of the *Block Scenario* assembly process simulation at LoD 3 with detailed contact forces based on the PCM algorithm and robot dynamics models for multiple steps. An *DASSl* solver with $2 \cdot 10^{-6}$ tolerance is used. The simulation time is 42 s and the computation time is 415 s (average real-time factor of 9.88). The red arrows represent the contact forces between the *Part* and other components.

fifth order *Esdirk* solver (*Esdirk45a*) empirically demonstrated a good balance between accuracy and computation time in comparison to the third or forth order *Esdirk* solver.
The simulation has a simulation time of 42 s and a computation time of 415 s. This results in an average real-time factor of 9.88. Simulations with a real-time factor greater than 1 are not real-time capable. The model at this LoD can therefore not be used for HiL simulations, in which the real controller controls the process in the model in real-time.
Further computation times for selected solvers and step sizes (fixed-step solvers) or tolerances (variable-step solvers) are now compared for the *Block Scenario* assembly process. The results are shown in Table 5.1. They differ slightly because the visualization is disabled for a better comparison. Computation times greater than 10 minutes are rounded to full minutes or hours. The following criteria are defined for the evaluation.

- A *position error* occurs if the position of component C, P, O, or B at the end of the simulation $r_{Simulation}$ differs by more than 1 mm from the analytically calculated one:

$$\Delta r = |r_{Simulation} - r_{Analytical}| \tag{5.18}$$

$$position\ error = \begin{cases} 1 & \Delta r > 0.001 \text{ m} \\ 0 & \Delta r \leq 0.001 \text{ m}. \end{cases} \tag{5.19}$$

  This is a factor of 10 greater than the maximum position error Δr_{max} specified in Equation (5.5) to allow for certain numerical deviations.

- An *instable* simulation occurs when the simulation is aborted due to numerical instabilities of the solver before the simulation end time is reached.

- *Contact broken* means that at least one component has lost its connection to another one (i.e. the substitute contact force is set to zero).

The evaluation yielded the following results. The maximum step size for the *Rkfix2* solver is 0.001 s for simulations at LoD 1 and 0.0002 s for simulations at LoD 2. Both LoD 1 and LoD 2 simulations can be performed with the fixed-step solver *Rkfix2* and with the variable-step solvers *DASSL*, *Cvode*, and *Esdirk45a*.
Simulations at LoD 3 can only be performed with reasonable computational times using variable-step solvers. For the *DASSL* solver, the tolerance must be smaller than 0.0001. Otherwise, the simulation terminates due to numerical instabilities. The *Cvode* solver provides accurate results for tolerances of $1 \cdot 10^{-5}$ or less whereas the *Esdirk45a* solver can be used with tolerances up to 0.001. For the multi-step methods with variable step size such as *DASSL* and *Cvode*, increasing the tolerance does not necessarily lead to faster computation times. On the contrary, the computation time of the single-step implicit Runge-Kutta scheme of the *Esdirk45a* solver is primarily depending on the tolerance. Furthermore, the visualization creates additional events in the model. This has a greater impact on multi-step methods than on single-step methods.

Table 5.1: Computation times for selected solvers and step sizes or tolerances for the *Block Scenario* assembly process simulation at multiple LsoD. The visualization is disabled in this comparison, so the results differ slightly from the previous ones. The superscript indicates: [1]*position error* (i.e. the final positions of the *Part* components differ from the analytically calculated final positions by more than 1 mm).

		Computation time depending on the LoD		
Solver	Step size / tolerance	LoD 1 (kinematics)	LoD 2 (dynamics)	LoD 3 (dyn. contacts)
Rkfix2	$1 \cdot 10^{-6}$ s	54 min	65 min	(15 h)[1]
Rkfix2	$1 \cdot 10^{-5}$ s	343 s	394 s	*contact broken*
Rkfix2	0.0001 s	34.3 s	39.9 s	*instable*
Rkfix2	0.0002 s	17.2 s	20.8 s	*instable*
Rkfix2	0.001 s	3.9 s	*instable*	*instable*
Rkfix2	0.01 s	*contact broken*	*instable*	*instable*
DASSL	$1 \cdot 10^{-6}$	6.6 s	7.7 s	501 s
DASSL	$2 \cdot 10^{-6}$	5.9 s	7.0 s	405 s
DASSL	$1 \cdot 10^{-5}$	5.3 s	6.3 s	255 s
DASSL	0.0001	5.0 s	5.9 s	144 s
DASSL	0.001	4.8 s	(5.9 s)[1]	*instable*
DASSL	0.01	(4.7 s)[1]	(5.8 s)[1]	*instable*
Cvode	$1 \cdot 10^{-6}$	5.0 s	5.9 s	148 s
Cvode	$1 \cdot 10^{-5}$	4.5 s	5.2 s	106 s
Cvode	0.0001	4.2 s	4.9	(148 s)[1]
Cvode	0.001	(4.1 s)[1]	(4.8 s)[1]	*contact broken*
Esdirk45a	$1 \cdot 10^{-6}$	6.0 s	6.9 s	439 s
Esdirk45a	$1 \cdot 10^{-5}$	4.2 s	5.0 s	270 s
Esdirk45a	0.0001	4.0 s	4.5 s	206 s
Esdirk45a	0.001	(3.6 s)[1]	(4.3 s)[1]	193 s
Esdirk45a	0.01	(4.1 s)[1]	(4.3 s)[1]	(235 s)[1]

5.1.3 Disassembly

The *Block Scenario* example can also be used to simulate a disassembly process. The parameters from Section 5.1.1 are used. However, the initial positions of the *Part* components are different. In addition, a lower coefficient of friction $\mu = 0.1$ (see Equation (5.13)) is used so that the components can be disassembled without counter-holding. This is not necessary for the assembly process because here the table is used for counter-holding.
The disassembly process was successfully simulated at all LsoD. The visualization of the resulting simulation at LoD 1 is shown in Figure 5.4. The computation times were 4.12 s (average real-time factor 0.098) for the simulation at LoD 1 and 20.8 s (average real-time factor 0.50) for the simulation at LoD 2. A *Rkfix2* solver was used for both simulations with step sizes of 0.001 s and 0.0002 s. These computation times are slightly longer than those for the assembly process because more collision checks must be performed overall.
For the simulation at LoD 3, the computation time was 533 s using a *DASSL* solver with a tolerance of $2 \cdot 10^{-6}$. The computation time for the disassembly process at LoD 3 is longer than the computation time for the assembly process, as it requires more computationally intensive contact calculations (the assembly with many contacts exists for a longer time overall).

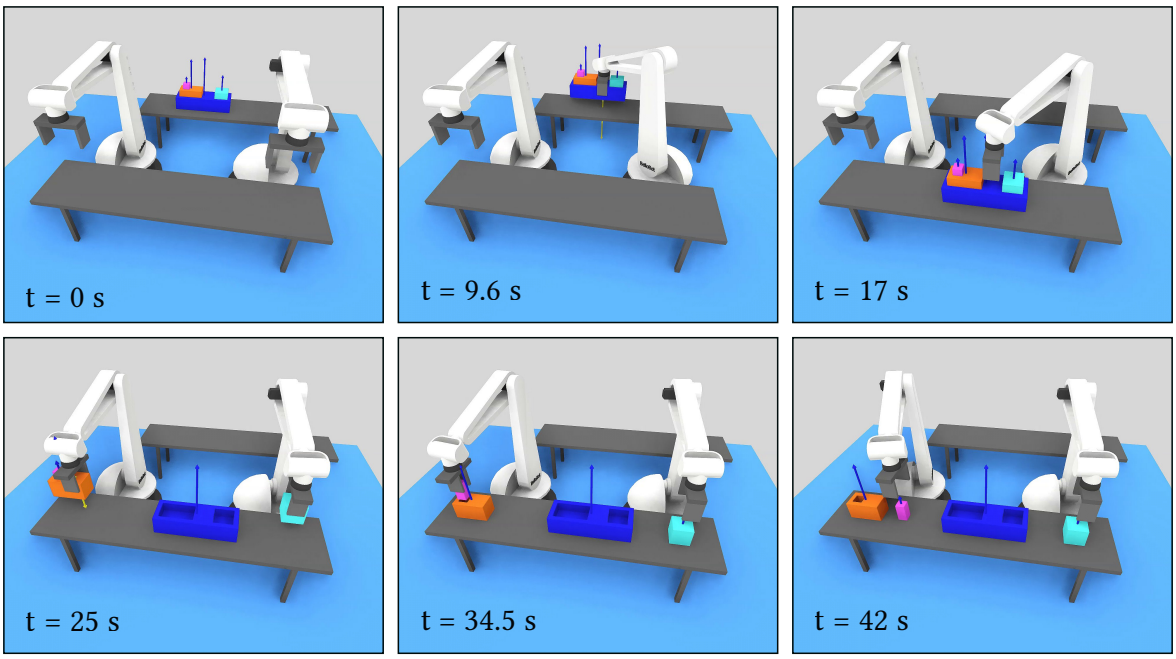

Figure 5.4: Visualization of the disassembly process simulation at LoD 1 for the *Block Scenario* example based on substitute forces and robot kinematics models. A *Rkfix2* solver with a fixed step size of 0.001 s is used. The simulation time is 42 s and the computation time is 4.12 s (average real-time factor of 0.098). The arrows represent the resulting forces for the *Part* (blue) and *Gripper* (yellow) components.

5.2 Analysis of the block scenario example

A detailed analysis is performed only for the *Block Scenario* example and not for the other use cases. This analysis includes an analytical evaluation of a grasped component in the static case, a comparison between the Substitute Force Algorithm and the contact force algorithm, and a comparison between the SFA and the AGPSA. It also includes a time analysis.

5.2.1 Grasped component in static case

First, a grasped component of type *Part* is analyzed. In the *Block Scenario* assembly simulation at LoD 2, at time 9.8 s, component C is grasped by Gripper2 and held on top of component B (see Figure 5.2 for details). This state can be considered as quasi static. Figure 5.5 shows the forces acting on component C and Gripper2. In this time step, the two components are considered as one component (rigidly connected by substitute contact forces).
The weight forces act on component C and Gripper2 in the z-direction. Hence, the gripper

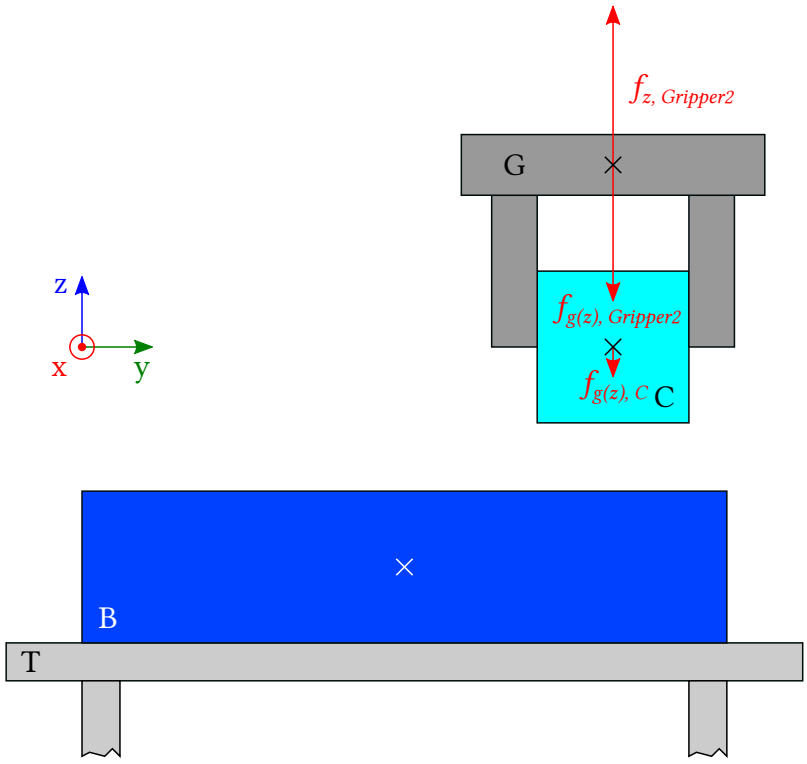

Figure 5.5: Forces acting on component C and Gripper2 at time 9.8 s in the *Block Scenario* assembly process simulation at LoD 2. In this time step, the two components are considered to be rigidly connected (by substitute contact forces).

acts on the robot with the following force (in the z-direction):

$$f_{z,\ Gripper2} = -f_{g(z),\ C} - f_{g(z),\ Gripper2} = -(m_C + m_{Gripper2}) \cdot g \tag{5.20}$$

$$\Rightarrow f_{z,\ Gripper2} = -(0.5\ \text{kg} + 3\ \text{kg}) \cdot 9.80665\ \frac{\text{m}}{\text{s}^2} \approx 34.3233\ \text{N}. \tag{5.21}$$

This analytical value is now compared with the results of the simulation:

$$f_{z,\ Gripper2,\ Simulation,\ Substitute\ forces}\ (t = 9.8\ \text{s}) = 34.3233\ \text{N} \tag{5.22}$$

$$f_{z,\ Gripper2,\ Simulation,\ Contact\ forces}\ (t = 9.8\ \text{s}) = 34.3233\ \text{N}. \tag{5.23}$$

Both the result based on substitute forces (Equation (5.22)) and the result based on contact forces (Equation (5.23)) match the analytically calculated value (Equation (5.21)).

5.2.2 Comparison between the contact and substitute force model

A more detailed comparison between the results of the simulation at LoD 2 (robot dynamics and substitute forces) and at LoD 3 (robot dynamics and contact forces) for the forces acting

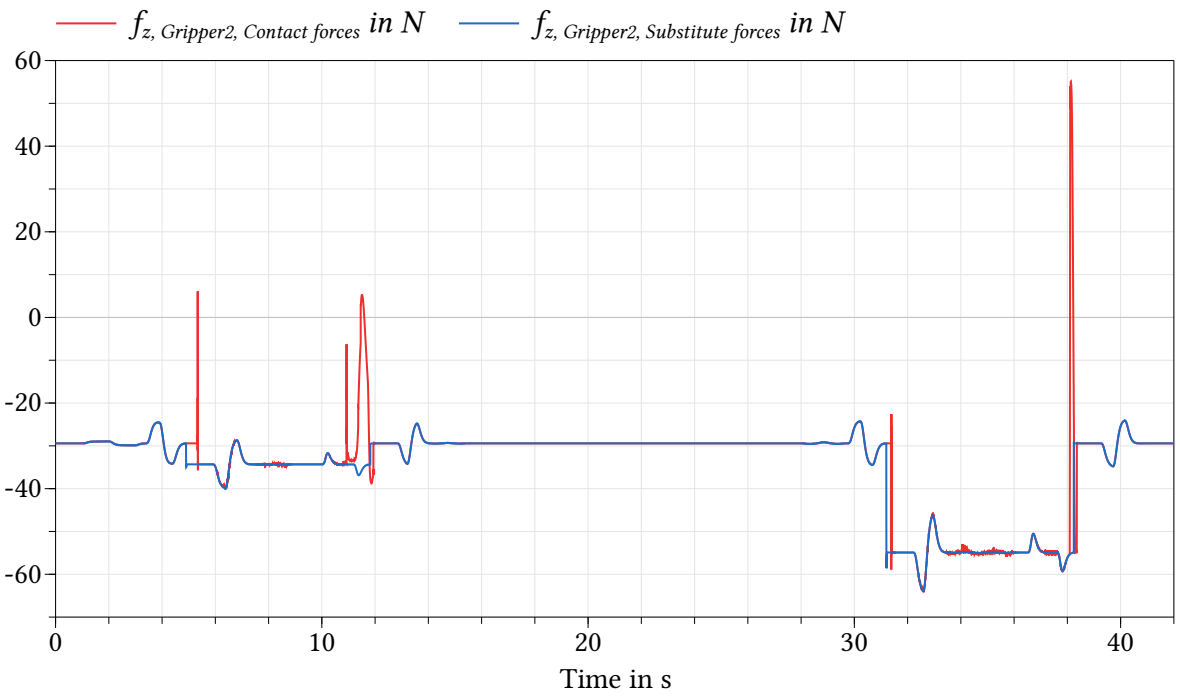

Figure 5.6: The resulting force for Gripper2 in the z-direction for both the simulation at LoD 3 based on contact forces (red) and at LoD 2 based on substitute forces (blue). Both simulations use robot dynamics models and the same solver settings as in Section 5.1.2. The curves differ at the points where the grasped components interact with a table or are assembled.

on Gripper2 is shown in Figure 5.6. The PCM algorithm is used as a reference for this validation (no hardware validation study is performed). A *DASSL* solver with $2 \cdot 10^{-6}$ tolerance is used for the former, a *Rkfix2* solver with a step size of 0.0002 s for the latter (same settings as in Section 5.1.2). The curves match to a large extent. Differences can be seen at the times when the grasped component or assembly interacts with the table or is assembled. Here, the detailed model provides contact forces in contrast to the substitute model. Examples are the pick up of component C (4.9 s), the assembly of component C (10.9 s), and the pickup (31.2 s) and placement of the entire assembly (38.0 s). In addition, the substitute contact force is applied immediately when the gripper closes (4.9 s and 31.2 s), whereas the contact force is applied when the jaws reach the component (5.1 s and 31.4 s).
Next, the torques of the robot axes are compared. Figure 5.7 shows the resulting torques from axis 2 and axis 3 of Robot2 (normalized to the maximum torque for each axis) for both the simulation at LoD 2 and LoD 3. The comparison of the torques is similar to the comparison of

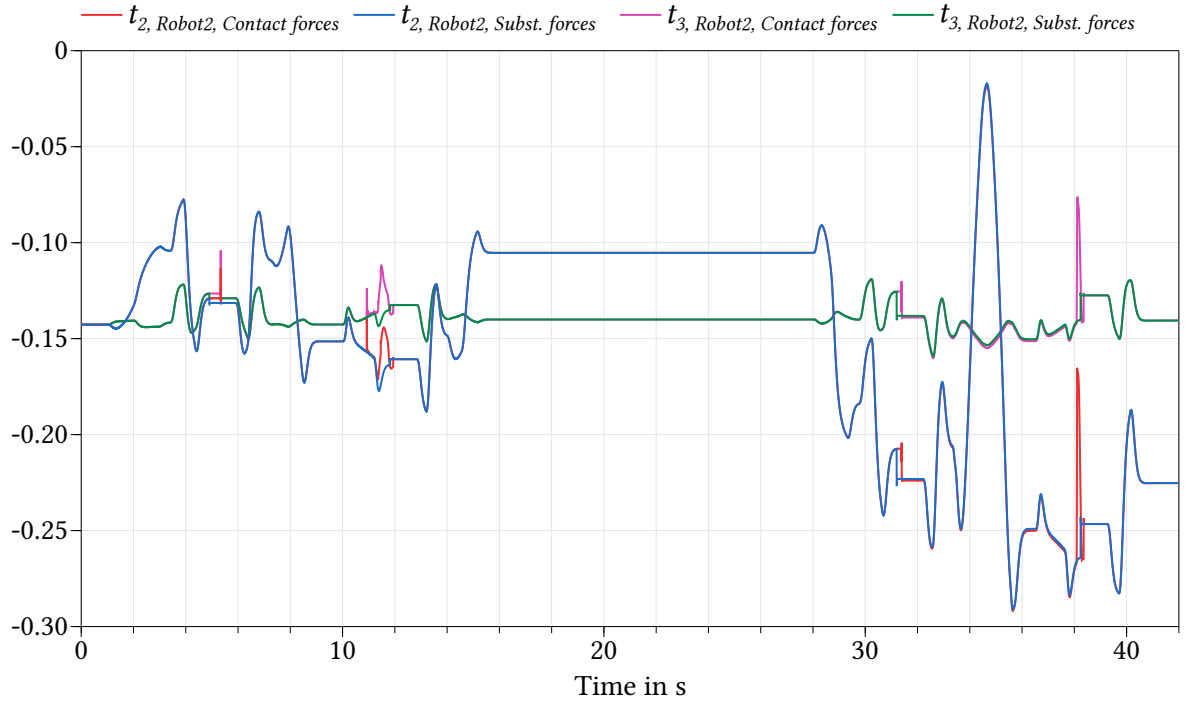

Figure 5.7: The resulting torques from axis 2 and axis 3 of Robot2 for both the simulation at LoD 2 based on substitute forces (*Rkfix2* solver with 0.0002 s step size) and at LoD 3 based on contact forces (*DASSL* solver with $2 \cdot 10^{-6}$ tolerance). Both simulations use dynamics models for the robots. The torques are normalized to the maximum torque for both axes. The curves differ at the points where the grasped components interact with the environment.

the forces in the gripper. The torque curves are very similar. Differences occur at the points where the grasped component or assembly interacts with the environment. Furthermore, the evaluation shows that the simulation at LoD 2 based on substitute contact forces is sufficient to determine the dynamic loads on the robot.
In addition, the positions of the components are compared. The *Part* component P is selected for the comparison because its position depends on multiple other components, including B and O, and it is manipulated the most. The comparison for the position of component P is shown in Figure 5.8. The position in x-direction, y-direction, and z-direction is compared between the simulation at LoD 3 and at LoD 2. The former simulation is run with an *DASSL* solver with $2 \cdot 10^{-6}$ tolerance, the latter with a *Rkfix2* solver with a fixed step size of 0.0002 s. The position error between the simulation based on contact forces and the simulation based on substitute forces is 0.00024 m for component P. The maximum position error in the entire model occurs for component O and is 0.00025 m.
At several points in time, the difference between the two results in the z direction has spikes. For example, at time 3.8 s and 16.0 s Gripper1 closes and at 31.2 s Gripper2 closes. At time 9.9 s and 22.8 s Gripper1 opens and at 38.2 s Gripper2 opens. At each of these times, the offset for the calculation of the substitute contact force is recalculated. The substitute contact

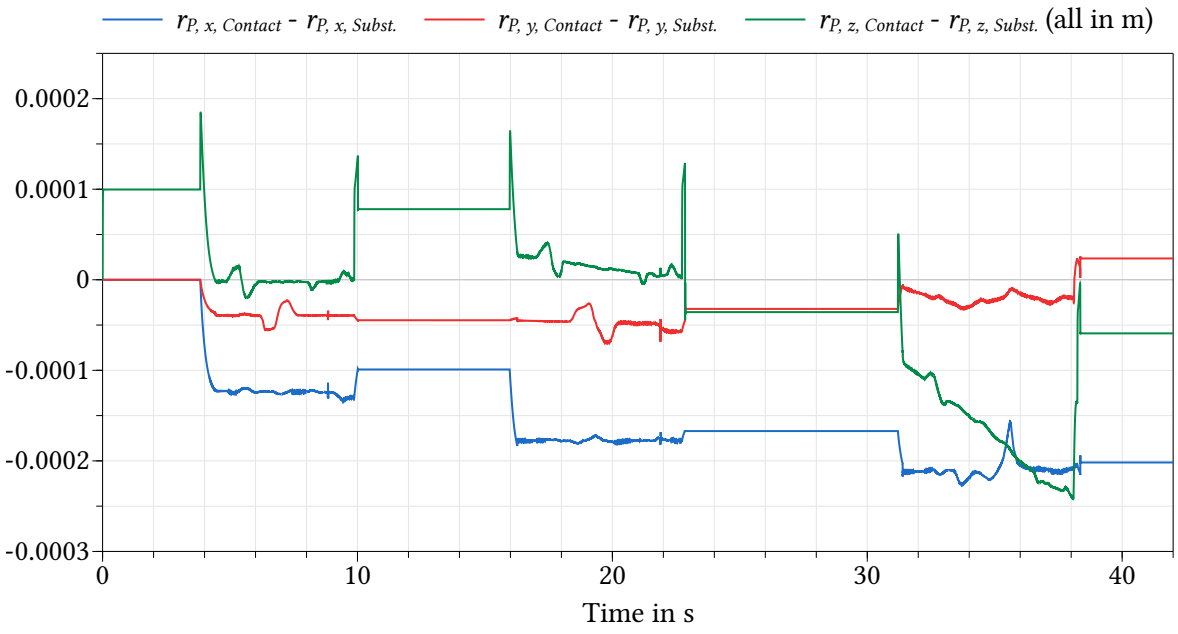

Figure 5.8: A comparison of the position of component P in x-direction (blue), y-direction (red), and z-direction (green). The results from the simulation with contact forces are compared with the results from the simulation with substitute forces. An *DASSL* solver with $2 \cdot 10^{-6}$ tolerance is used for the simulation at LoD 3 and a *Rkfix2* solver with a step size of 0.0002 s for the simulation at LoD 2.

force starts at zero and increases until the weight force is compensated. This means that the spring in the substitute contact force calculation must be retensioned. The analytical position difference is therefore the one defined in Section 5.1.1, the maximum position error of $\Delta r_{max} = 0.0001$ m. In addition, there is the numerical inaccuracy of the simulation, which depends on the tolerance (for variable-step solvers) or the step size (for fixed-step solvers).
The difference at 38.2 s is larger, because the entire assembly has been moved beforehand. This movement covers a greater distance than the movements of the individual components. Additionally, the assembly is heavier than a single component. Both of these points cause the assembly to slide down slightly in the model based on contact forces (LoD 3), since static friction is only approximated (see Section 2.2.2). In contrast, this effect does not occur for the simulation based on substitute contact forces (LoD 2).
In summary, by using the model based on substitute contact forces (LoD 2) instead of the model based on contact forces (LoD 3), the computation time can be reduced by 95 % while having a position error of less than 0.00025 m.

5.2.3 Comparison of both substitute force algorithms

In this section, the Substitute Force Algorithm extended for manipulation (see Section 3.4.2) and the Assembly Graph Partner Search Algorithm (AGPSA) (see Section 3.5.2) are compared. The *Block Scenario* example at LoD 1 is used with the parameters from Section 5.1.1.
The first step is to determine the maximum step size for the simulation based on the AGPSA. With the given parameters, the maximum step size is 0.005 s when using the *Rkfix2* solver. For a step size of 0.006 s or higher, at least one component loses its connection to another one. The resulting computation time is 0.97 s. For this step size, the maximum position error for the end positions of the components C, P, O, and B is 0.00042 m. In comparison, the maximum position error for a step size of 0.001 s is 0.00031 m.
Now the Substitute Force Algorithm extended for manipulation is used instead of the AGPSA. When using a step size of 0.005 s, all components lose their connection. There are two ways of making the simulation feasible:

- Reduce the fixed step size for the *Rkfix2* solver.
- Reduce the stiffness of the substitute contact forces in the model.

The results are as follows. In the first case, the procedure is to reduce the step size until there are no more broken contacts. The step size must be reduced to 0.0007 s in order for the simulation to produce feasible results. This increases the computation time to 3.98 s. The corresponding maximum position error is 0.00130 m. The position error is generally larger for the Substitute Force Algorithm extended for manipulation, because substitute contact forces connected in parallel and substitute contact forces connected in series occur here (see Figure 3.12).
In the second case, the stiffness is reduced in 1 % increments (in relation to the initial value). The same procedure is then carried out for each step with the damping. The result is that the

stiffness (translational spring constant) must be reduced by 59 % and the damping (translational damping constant) by 77 % for all component pairs in the process model. This reduces the computation time to 0.76 s, but increases the position error by 297 % to 0.00386 m. An overview of the results is given in Table 5.2.
This analysis shows that the Assembly Graph Partner Search Algorithm can significantly improve the accuracy and computation time of a simulation, even with few *Part* components and *Part* component pairs in the model. For the simulation with a step size of 0.0007 s, the maximum position error is decreased from 0.00130 m to 0.00031 m (76 %). Additionally, the computation time for a model with given parameters can be reduced significantly. With the SFA extended for manipulation, the computation time is 3.98 s due to the step size limit. This can be reduced to 0.97 s by using the AGPSA (76 %).
Another result of the comparison is that the SFA extended for manipulation requires less computation time than the Assembly Graph Partner Search Algorithm for the same step size. For example, the computation time for a simulation at LoD 1 with a step size of 0.0007 s is 4.96 s when using the Assembly Graph Partner Search Algorithm and 3.98 s when using the Substitute Force Algorithm extended for manipulation (see Table 5.2). The same applies to a simulation at LoD 2 with a step size of 0.0002 s. Here, the computation times are 20.4 s (AGPSA) and 17.1 s (SFA extended for manipulation) (16 % faster).

Table 5.2: A detailed comparison of the Substitute Force Algorithms for the *Block Scenario* assembly simulation (see Section 3.1). Simulations are run at LoD 1 and 2 for both the SFA based on the AGPSA and the SFA extended for manipulation.

LoD	Substitute Alg.	Stiffness $k_{tr,\,B}$	Step size	Comp. time	Error Δr_{max}
1	AGPSA-based	98066 $\frac{\mathrm{N}}{\mathrm{m}}$	0.006 s	contact broken	-
1	AGPSA-based	98066 $\frac{\mathrm{N}}{\mathrm{m}}$	0.005 s	0.97 s	0.00042 m
1	AGPSA-based	98066 $\frac{\mathrm{N}}{\mathrm{m}}$	0.001 s	4.00 s	0.00031 m
1	AGPSA-based	98066 $\frac{\mathrm{N}}{\mathrm{m}}$	0.0007 s	4.96 s	0.00030 m
1	Extended SFA	98066 $\frac{\mathrm{N}}{\mathrm{m}}$	0.005 s	contact broken	-
1	Extended SFA	98066 $\frac{\mathrm{N}}{\mathrm{m}}$	0.0008 s	contact broken	-
1	Extended SFA	98066 $\frac{\mathrm{N}}{\mathrm{m}}$	0.0007 s	3.98 s	0.00130 m
1	Extended SFA	40207 $\frac{\mathrm{N}}{\mathrm{m}}$	0.005 s	0.76 s	0.00386 m
2	AGPSA-based	98066 $\frac{\mathrm{N}}{\mathrm{m}}$	0.0002 s	20.4 s	0.00022 m
2	Extended SFA	98066 $\frac{\mathrm{N}}{\mathrm{m}}$	0.0002 s	17.1 s	0.00099 m

5.2.4 Time analysis

In this section, the computation time for the assembly simulation of the *Block Scenario* example (see Section 3.1) is analyzed. On the one hand, the effects of the Level of Detail (LoD) (see Table 3.4), the optimization of the function calls (see Section 4.5) and the parallelization of the Polygonal Contact Model (PCM) contact calculation (see Section 4.5) on the computation time are considered. On the other hand, the share of the computation time required for the evaluations in the External Environment (EE), including the evaluations of the Minkowski Portal Refinement (MPR) and PCM algorithms, is analyzed.
The time analysis is performed using the `Advanced.GenerateFunctionTimers` function in *Dymola 2025x*. Using this function results in slight overhead in the calculation, so the computation times slightly differ from those in Section 5.1.2.
Table 5.3 shows a detailed overview of the results for the simulations at LoD 1 and LoD 2. When looking at the simulations at LoD 1, it can be seen that the optimization of the function calls does not significantly reduce the computation time. In this simulation, a *Rkfix2* solver with a step size of 0.001 s is used (see Table 5.1). The MPR evaluation for one pair is called $4.9 \cdot 10^6$ times without the optimization. When using the optimization, the MPR evaluation is called $3.1 \cdot 10^6$ times and skipped $1.8 \cdot 10^6$ times. The computation time decreases from 4.6 s (without optimization) to 4.2 s (with optimization) (9 % reduction).
For the simulations at LoD 2, the optimization of the function calls has a similar effect. There are $24.1 \cdot 10^6$ component pair evaluations with the MPR algorithm without the optimization and $14.6 \cdot 10^6$ evaluations and $9.5 \cdot 10^6$ skips when using the optimization. The respective computation times are 23.4 s and 21.3 s (9 % reduction).
The situation is different for simulations at LoD 3. Here, the computation time can be significantly reduced by the optimization of the function calls. If the parallelization of the PCM calculation is used in addition, the computation time can be reduced even further. The simulation is run with a *DASSL* solver and an *Esdirk45a* solver at $2 \cdot 10^{-6}$ tolerance. These solvers

Table 5.3: A detailed analysis of the computation time for the *Block Scenario* assembly simulation (see Section 3.1) at LoD 1 and LoD 2 depending on the usage of the function call optimization (parallelization is not used). The second part shows the time and the share required for the evaluation in the EE and the time and the number of evaluation function calls for the MPR algorithm.

LoD	Opt.	Par.	Comp. time	EE time	EE share	MPR time	MPR calls
1	-	-	4.6 s	2.1 s	45 %	1.6 s	$4.9 \cdot 10^6$
1	x	-	4.2 s	1.7 s	40 %	1.3 s	$3.1 \cdot 10^6$
2	-	-	23.4 s	9.3 s	40 %	7.7 s	$24.1 \cdot 10^6$
2	x	-	21.3 s	7.4 s	35 %	6.1 s	$14.6 \cdot 10^6$

are chosen because the former uses an implicit multi-step method with variable step size and order, while the latter uses an implicit single-step Runge-Kutta scheme with a variable step size. Thus, both a multi-step and a single-step method are tested.
The results are shown in Table 5.4. In all cases, the evaluation in the EE requires between 92 % and 98 % of the computation time. For the usage of the *DASSL* solver, the computation time is 1560 s for the simulation without the function call optimization and the parallelization. The PCM evaluation is called $49.7 \cdot 10^6$ times. By using the parallelization of the PCM evaluation function calls, this time is reduced by 47 % to 831 s. The optimization of the function calls has an even greater impact. It decreases the computation time by 62 % to 597 s. The number of PCM evaluation function calls is reduced to $16.9 \cdot 10^6$.
However, a combination of both improvements is most effective. When both the parallelization and optimization of the PCM evaluation function calls are used, the computation time is reduced by three quarters from 1560 s to 409 s (74 % reduction).
Similar results were achieved using the *Esdirk45a* solver. Here, the computation time is 1273 s for the simulation without improvements. The optimization of the function calls reduces the number of function calls from $41.7 \cdot 10^6$ to $17.0 \cdot 10^6$ and the computation time by 55 % to 573 s. A decrease in the computation time to 688 s is achieved by using the parallelization (46 % reduction). A combination of both improvements works best here, too. The computation time is reduced by 70 % to 384 s.

Table 5.4: A detailed analysis of the computation time for the *Block Scenario* assembly simulation (see Section 3.1) at LoD 3 depending on the usage of the function call optimization and the parallelization of the PCM evaluations. The second part shows the time and the share required for the evaluation in the EE and the time and the number of evaluation function calls for the PCM algorithm. The simulation is run with a *DASSL* solver and an *Esdirk45a* solver at $2 \cdot 10^{-6}$ tolerance.

Solv.	Opt.	Par.	Comp. time	EE time	EE share	PCM time	PCM calls
DASSL	-	-	1560 s	1527 s	98 %	1524 s	$49.7 \cdot 10^6$
DASSL	-	x	831 s	796 s	96 %	792 s	$49.7 \cdot 10^6$
DASSL	x	-	597 s	566 s	95 %	564 s	$16.9 \cdot 10^6$
DASSL	x	x	408 s	379 s	93 %	377 s	$16.9 \cdot 10^6$
Esdirk45a	-	-	1273 s	1242 s	98 %	1240 s	$41.7 \cdot 10^6$
Esdirk45a	-	x	688 s	654 s	95 %	651 s	$41.7 \cdot 10^6$
Esdirk45a	x	-	573 s	546 s	95 %	544 s	$17.0 \cdot 10^6$
Esdirk45a	x	x	384 s	354 s	92 %	352 s	$17.0 \cdot 10^6$

5.3 Simulation of the block scenario example using detailed drivetrain models

The block scenario assembly process is simulated here using other robots, with detailed models of the drivetrain. The scenario shows how the developed solution presented in Chapter 3 enables detailed, multi-domain system simulations of assembly processes.

5.3.1 Overview of the models used

The *Block Scenario* example is introduced in Section 3.1 and simulated in Section 5.1. The process is simulated again, but with different robots. Two *KUKA KR16 C2* robots are used because detailed models of the drivetrain are available for them (Reiner 2011). The remaining parts of the robot models are again part of the *DLR Robots library* and the visualization is based on the *DLR Visualization 2 Library*. The visualization of the modified scenario is presented in Figure 5.9.
Because the workspace of the *KUKA KR16 C2* robots is smaller, Robot1 must be moved 50 cm in the y-direction (see Figure 5.3 and Figure 5.9). The program for Robot1 is modified accordingly. Models of the drivetrain include the control, the electric motors with friction, and gears with friction. They are based on the work of Reiner (2011).

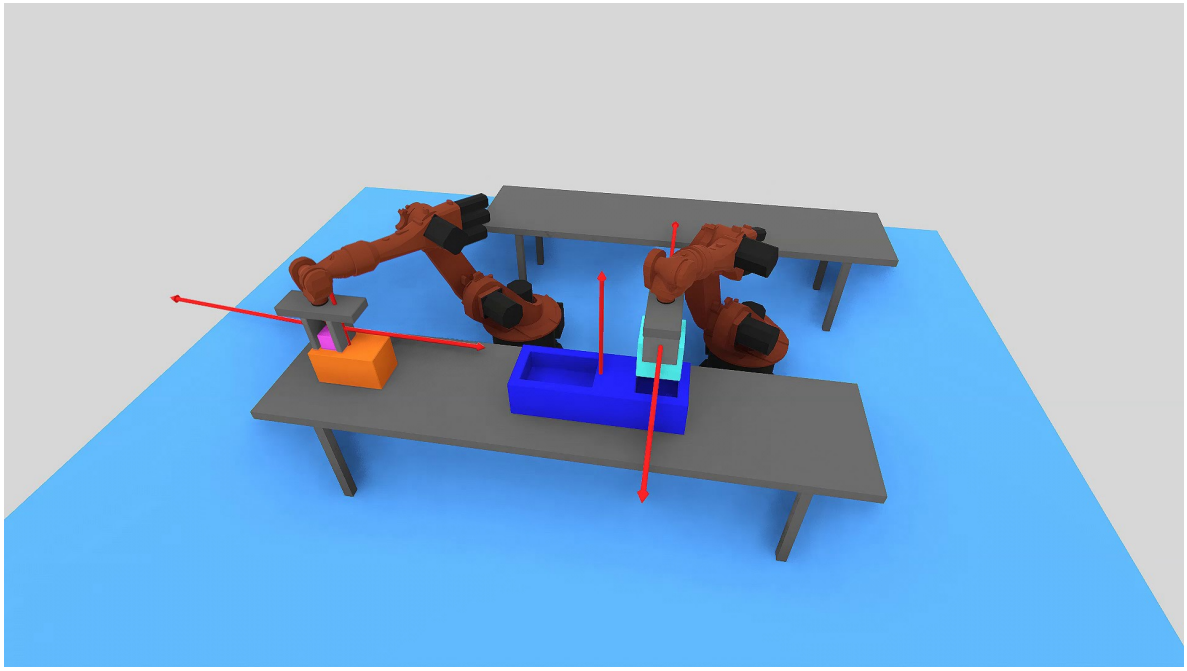

Figure 5.9: Visualization of the modified *Block Scenario* examples with *KUKA KR16 C2* robots and detailed drivetrain models. The image shows the assembly process at time 9.8 s from the simulation at LoD 3 with contact forces (represented by red arrows).

5.3.2 Resulting torque curves from the simulation

The simulation ran successfully at all Levels of Detail. For LoD 1, a *Rkfix2* solver with a fixed step size of 0.001 s was used and the computation time was 4.09 s (average real-time factor 0.097). The simulation at LoD 2 can be run with a *Rkfix2* solver with 0.0002 s step size. The computation time is 21.2 s (average real-time factor 0.50). It is also possible to run the simulation at LoD 2 with a variable-step solver such as *DASSL* (computation time 9.39 s for $1 \cdot 10^{-6}$ tolerance). Simulations of the scenario at LoD 3 are not real-time capable. The comptation times are 1292 s (*DASSL* solver), 277 s (*Cvode* solver), and 905 s (*Esdirk45a* solver) for a tolerance of $1 \cdot 10^{-6}$. The resulting torque curves (normalized to the maximum torque for each axis) for all axes of Robot1 are shown in Figure 5.10.
This application shows how the solution developed in Chapter 3 enables detailed simulations of assembly processes. By combining the developed component models with additional Modelica libraries, multi-domain system simulations become possible.

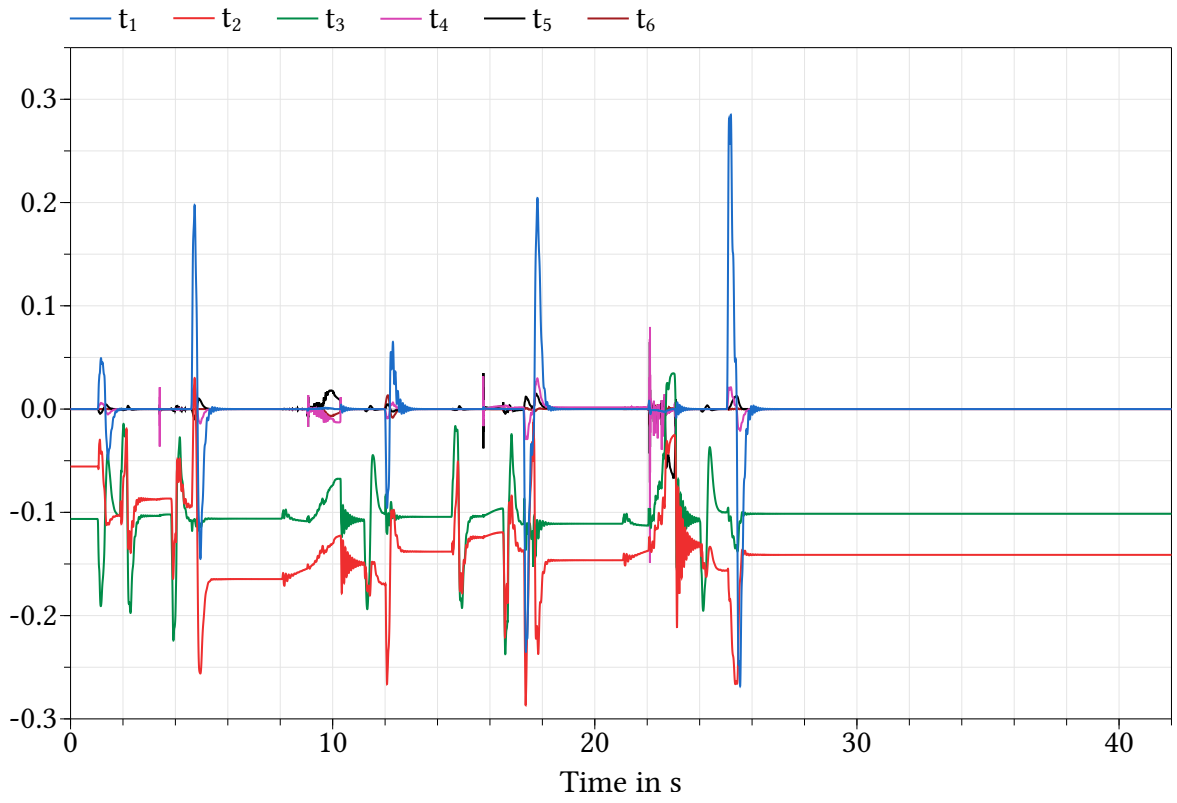

Figure 5.10: Torque curves for all axes of Robot1 (type *KUKA KR16 C2*) for the simulation at LoD 3 with contact forces. The torques are normalized to the maximum for each axis. Here, detailed models of the drivetrain are used, including the control, electric motors with friction, and gears with friction.

5.4 Assembly of a motor in a chainsaw housing

The second application involves the assembly of an electric motor into the housing of a chainsaw and is part of the German Aerospace Center (DLR) project *Factory of the Future (FoF)*[1] First, an overview is given and the process models are shown. Finally, the simulation results are presented.

5.4.1 Overview of the process

In this process, an electric motor is assembled into the housing of a chainsaw. Figure 5.11 shows the related simulation visualization. The scenario consists of a table, the chainsaw housing attached to a mount, a box containing the electric motor, and the *DLR SARA Robot* (Iskandar et al. 2021) with a gripper. The box is on the table, and the motor is inside the box. The chainsaw housing and its mount are considered a single component on the table.
First, the motor is picked from the box. The motor must be removed from the box perpendicular to the table and at a low velocity, because it will get stuck otherwise and the box will be

[1] `https://www.dlr.de/en/rm/research/projects/completed-projects/fof-x`

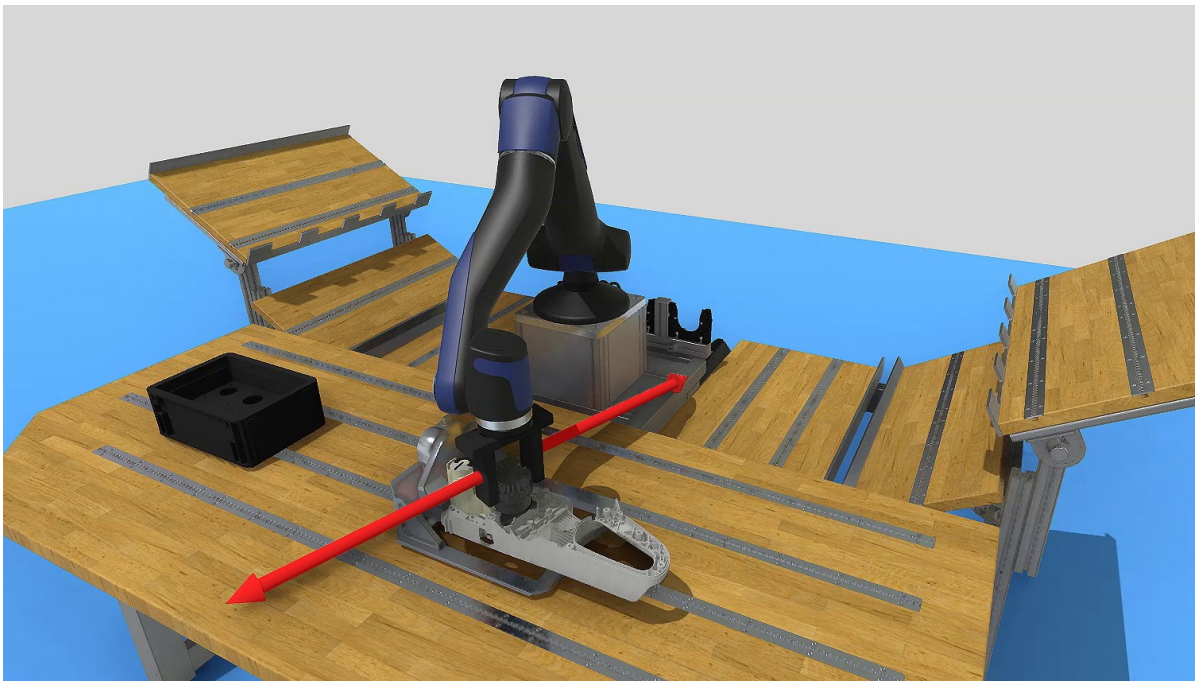

Figure 5.11: Visualization of the assembly simulation of an electric motor into the housing of an electric chainsaw. The geometry of the housing is complex and non-convex. Contact forces and dynamics models for the robot are used for the simulation (LoD 3). The arrows represent the contact forces for the gripper jaws.

lifted as well. Then, the motor is moved to a position above the housing. Finally, the motor is moved into the housing, rotated, and assembled.
The process model is built in Modelica using component models developed in this work in combination with other existing Modelica libraries. This demonstrates the flexibility of the Modelica environment. The corresponding object diagram is shown in Figure 5.12. The component models presented in Chapter 3 are used for the interaction between the components box, motor, mountHousing, gripper, and table. The manipulationWorld object is used for global settings regarding manipulation and assembly. In addition, the *DLR Robots library* is used for the robot and its controller and the *DLR Visualization 2 Library* for the visualization (including freeCamera, visualizationSetup, and skyBoxEnvironment). A simplified model of the drivetrain and the control is used for the robot. The remaining parts use multibody components from the MSL (including body and world).
The masses for box, motor, and mountHousing are as follows: $m_{Box/MH} = 0.5$ kg and $m_{Motor} = 0.3$ kg. As in the *Block Scenario* example, a maximum position error of $\Delta r_{max} = 0.0001$ m is specified. This leads to the following translational spring and damping constants for the SFA: $k_{tr,\ Box/MH} \approx 49033\ \frac{\mathrm{N}}{\mathrm{m}}$, $d_{tr,\ Box/MH} \approx 1.2 \cdot 314\ \frac{\mathrm{Ns}}{\mathrm{m}}$, $k_{tr,\ Motor} \approx 29420\ \frac{\mathrm{N}}{\mathrm{m}}$, $d_{tr,\ Motor} \approx 1.2 \cdot 188\ \frac{\mathrm{Ns}}{\mathrm{m}}$.

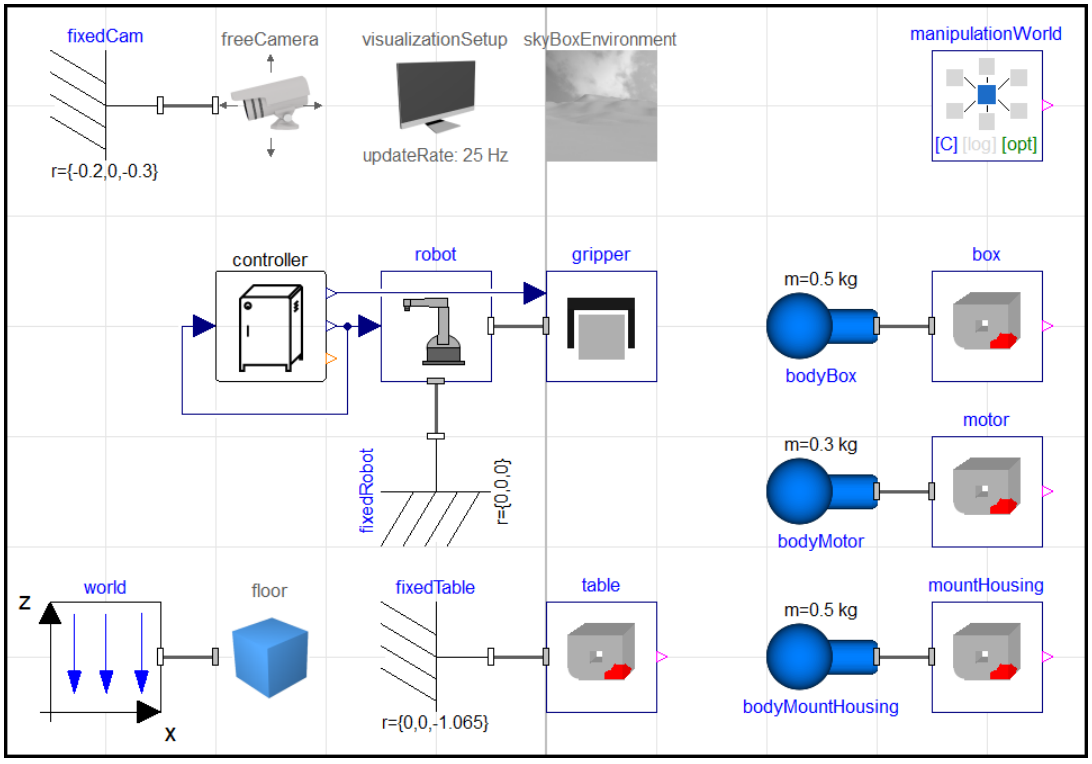

Figure 5.12: Object diagram of the process model in *Dymola* for the assembly process of an electric motor into the housing of a chainsaw. The diagram consists of component models from this work for the object interaction and components from the *DLR Robots library*, the *DLR Visualization 2 Library*, and the MSL.

They are specified by the user in the same way as in the *Block Scenario* example in Section 5.1. The damping constants are increased by 20 % to make the model more numerically stable.
For the PCM algorithm used as contact force model, the following parameters are used: Young's modulus $E = 3 \cdot 10^7$ Pa, Poisson's ratio $\nu = 0.3$, coefficient of friction $\mu = 0.5$, friction regularization velocity $v_\epsilon = 0.001 \frac{\text{m}}{\text{s}}$, and layer thickness $d = 0.002$ m. The coefficient of friction differs for the gripper: $\mu_{Gripper} = 0.90$.
For both the substitute force models and the contact force models, the 3D models are modified. All components in the scenario are non-convex geometries. Since the MPR collision detection algorithm used for the SFA is limited to convex geometries, decomposing the 3D models is necessary. Without convex decomposition, errors would occur, such as grasping the box when the motor is picked up from the box, because the convex hull of the box would be used. The convex decompositions for the components box and mountHousing are performed using the open source library *V-HACD* (Mammou 2016) (see Section 2.1.1). The box is decomposed into 24 convex parts and the mountHousing component into 40 convex parts. The resulting geometries are shown in Figure 5.13.
The 3D models must also be modified for the PCM algorithm. In some areas, the resolution of the polygonal mesh of the components is changed. Specifically, the resolution is increased

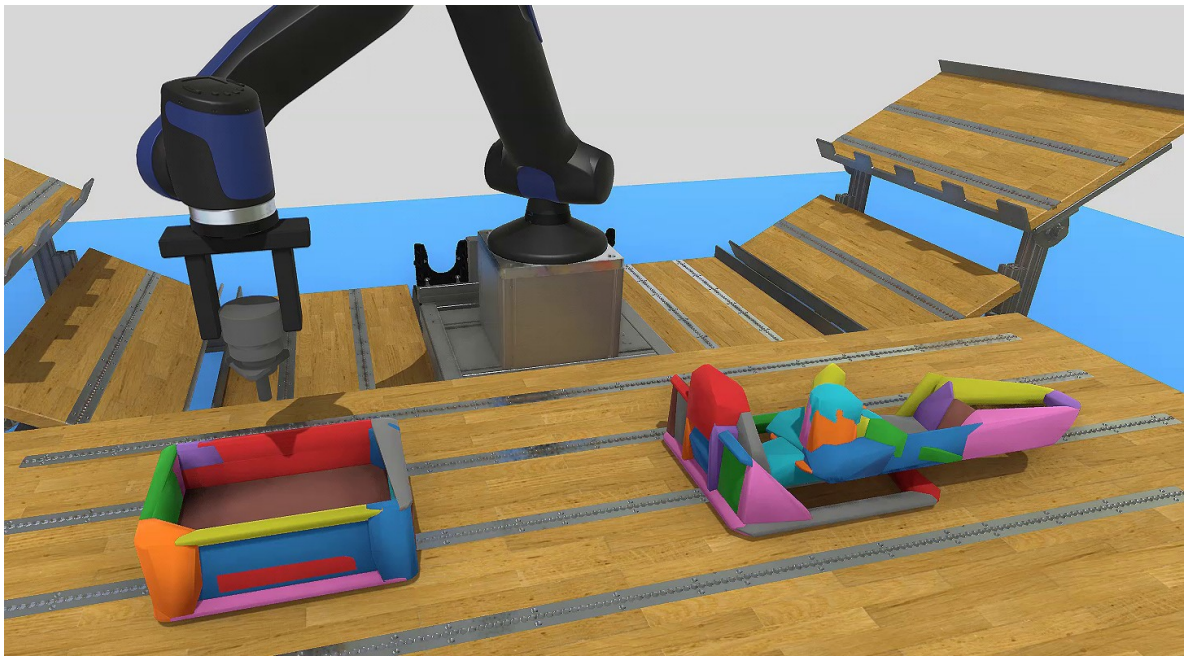

Figure 5.13: Visualization of the convex decompositions for the components mountHousing and box. The former is decomposed into 40 and the latter into 24 convex parts. These geometries are used for collision detection in the substitute contact force calculation. For the motor, a rebuild is used both for the substitute models and the contact force models.

in the contact areas between the motor and the box, as well as between the motor and the housing. For the press fit between motor and housing, the geometry influences the force during the assembly process and is therefore manually adjusted. Furthermore, chamfering is applied to the contact areas of all components because sharp edges can cause the PCM algorithm to behave in an unphysical manner (Hippmann 2024).
For the motor component, a rebuild is created that does not have the sharp cooling fins. One advantage of the rebuild is that convex decomposition is unnecessary. The five convex shapes of the rebuilt motor can be used directly for collision detection instead. For the PCM algorithm, these shapes are joined into one and given a detailed mesh with 4038 triangles. The box is described by 14728 triangles and the mountHousing by 158572 triangles. Additionally, the gripper jaws are described by 1224 triangles. The meshes of the motor, the gripper jaws, and the housing are shown in Figure 5.14.
The application shows the advantages of using the PCM algorithm as the contact force model. The geometries in this application are complex and non-convex. As previously demonstrated, collision algorithms that are limited to convex bodies require a convex decomposition. This can result in unphysical behavior such as sudden changes in the contact force at the transitions between the convex parts of a body. The PCM algorithm, however, is suitable for non-convex geometries and calculates contact forces across the entire contact surface.

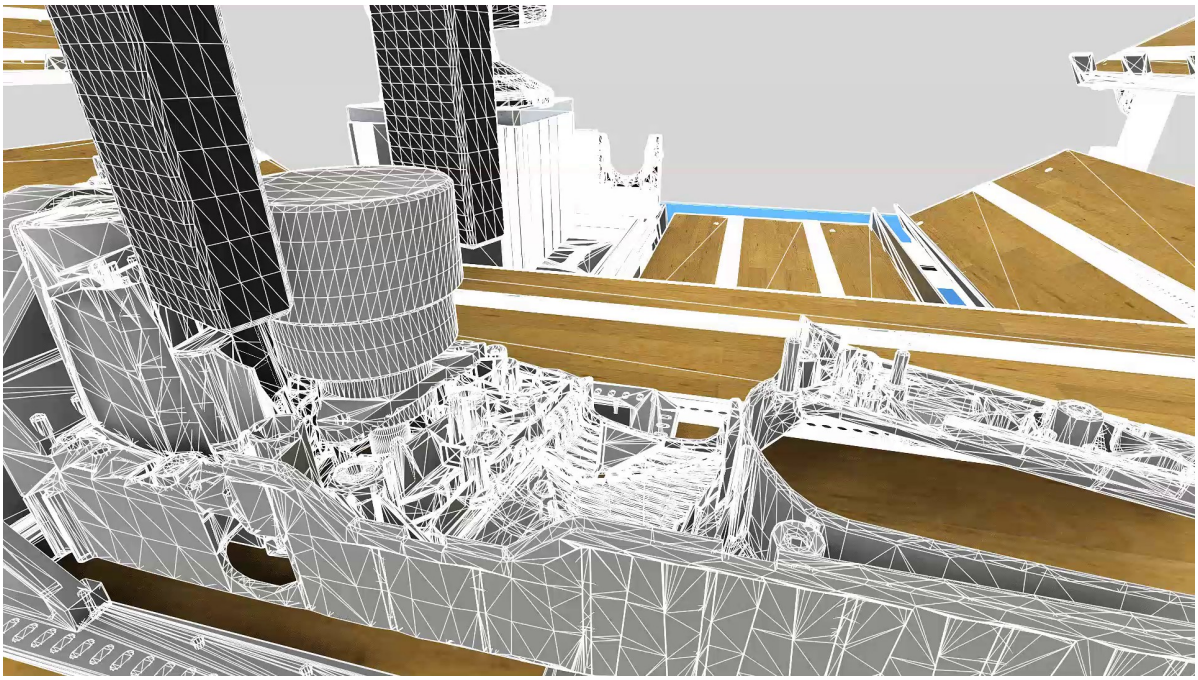

Figure 5.14: Visualization of the meshes of the components motor (4038 triangles), housing with mount (158572 triangles), and gripper jaws (1224 triangles). These polygonal representations of the components are the basis for the PCM algorithm.

5.4.2 Results and analysis

The simulation time for the assembly process is 30 s. The simulation was run successfully with a computation time of 6.78 s at LoD 1 and 359 s at LoD 3. A *Rkfix2* solver with a fixed step size of 0.001 s was used for the simulation at LoD 1 based on substitute contact forces. The simulation with contact forces at LoD 3 was run with the variable-step solver *DASSL* using a tolerance of $5 \cdot 10^{-7}$. The model also runs successfully with the same tolerance using the *Cvode* solver (computation time 115 s) or the *Esdirk45a* solver (computation time 256 s). The torque curves for all robot axes are shown in Figure 5.15. They are normalized to the maximum torque for each axis and based on the simulation at LoD 3. At time 23.8 s, the motor is assembled into the housing. This results in contact forces from the press fit and creates peaks in the torque curves, especially for axes 2 and 4. Since the robot in this model is position-controlled, small positional deviations can lead to high contact forces. This can be tested in the simulation and allows high contact forces to be identified in advance.
In addition, the simulation at LoD 2 was run with a computation time of 35.8 s with a *Rkfix2* solver with a fixed step size of 0.0002 s. This simulation is nearly real-time, but only on average. However, the simulation at LoD 2 is also beneficial for the development when using variable-step solvers and without real-time capability. For collaborative robots like the one used in this scenario, load is a more relevant factor than it is for industrial robots, even without contact forces. This is because collaborative robots typically have a significantly smaller payload capacity of about 5 to 10 kg. Even though there are only three components

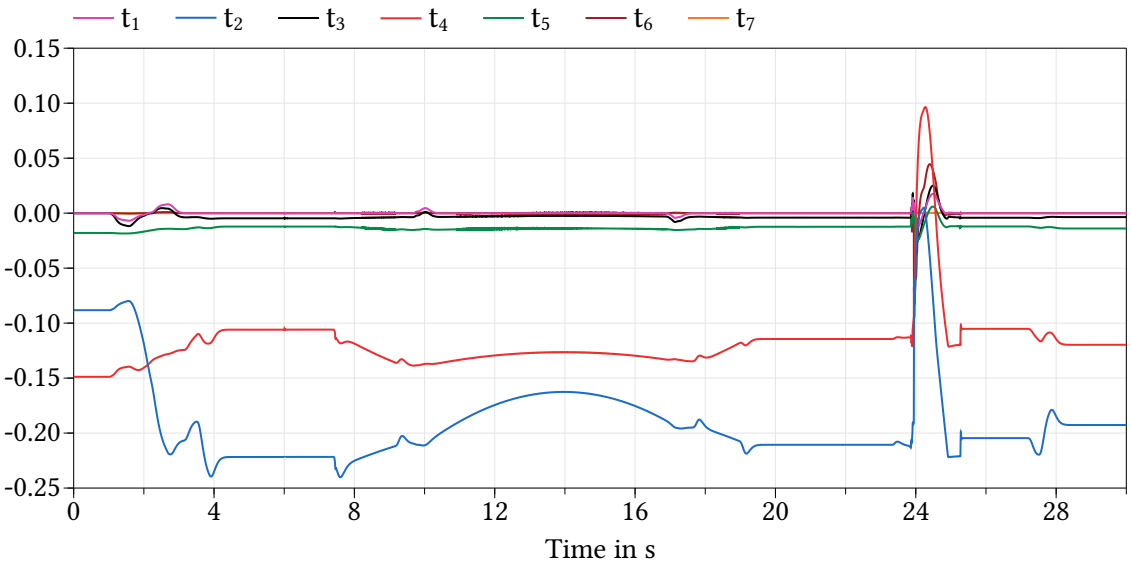

Figure 5.15: Torque curves for all robot axes (normalized to the maximum torque for each axis) during the assembly of the electric motor into the chainsaw housing. Contact forces resulting from the assembly process create torque peaks at time 23.8 s, particularly for axes 2 and 4.

in this example, the computation time for simulations at LoD 1 and LoD 2 is relatively high. This is due to the convex decompositions used for collision detection. For each component pair, all combinations of convex parts must be analyzed. This is time consuming. However, the simulation at LoD 1 is real-time capable (average real-time factor 0.23).
A detailed overview of computation times from different solvers is shown in Table 5.5. The results show that increasing the tolerance does not necessarily lead to a reduction in computing time with multi-step methods with variable step sizes, such as *DASSL* or *Cvode*. In contrast, the computing time for solvers such as *Esdirk45a* is reduced at higher tolerances because it is a single-step method with a variable step size and an implicit Runge-Kutta scheme. In both cases, computation time increases when step sizes are too large because then longer iterations are required to achieve convergence.

Table 5.5: Computation times for selected solvers and step sizes or tolerances for the motor housing assembly simulation at multiple LsoD. The visualization is disabled, so the results differ slightly from the previous ones. The superscript indicates: [1]*position error* (i.e. the final positions of the components differ by more than 1 mm).

		Computation time depending on the LoD		
Solver	Step size / tolerance	LoD 1 (kinematics)	LoD 2 (dynamics)	LoD 3 (dyn. contacts)
Rkfix2	$2 \cdot 10^{-6}$ s	105 min	108 min	(8 h)[1]
Rkfix2	$1 \cdot 10^{-5}$ s	21 min	21 min	*instable*
Rkfix2	0.0002 s	32.7 s	29.2 s	*instable*
Rkfix2	0.001 s	6.4 s	*instable*	*instable*
Rkfix2	0.01 s	(0.14 s)[1]	*instable*	*instable*
DASSL	$1 \cdot 10^{-7}$	22.3 s	14.7 s	734 s
DASSL	$5 \cdot 10^{-7}$	12.8 s	14.4 s	334 s
DASSL	$1 \cdot 10^{-6}$	11.5 s	12.7 s	225 s
Cvode	$1 \cdot 10^{-7}$	9.6 s	9.0 s	154 s
Cvode	$5 \cdot 10^{-7}$	8.2 s	8.8 s	112 s
Cvode	$1 \cdot 10^{-6}$	7.6 s	9.7 s	104 s
Esdirk45a	$1 \cdot 10^{-7}$	15.4 s	15.1 s	370 s
Esdirk45a	$5 \cdot 10^{-7}$	13.7 s	14.0 s	236 s
Esdirk45a	$1 \cdot 10^{-6}$	11.5 s	12.2 s	197 s

5.5 Multi-domain simulation of the disassembly of an electric motor

In this application, the disassembly process of an electric motor (used in Battery Electric Vehicles (BEVs)) is analyzed. The use case is part of the DLR project *MaTiC-M (Methods and Technologies for an intelligent Circularity of Materials)*[2]. First, a motivation and overview of the application is given. Then, the multi-domain simulation models are shown. Finally, the results are presented and the insights are discussed.

5.5.1 Motivation and overview

In this use case, the recycling of an electric motor used in BEVs is analyzed (Wachter 2023). The aim is to establish whether there are advantages to recycling in terms of criteria such as energy consumption and resource usage.
However, in this section the focus is on the disassembly process and its modeling and simulation. Figure 5.16 shows the components of the motor in multiple steps of the disassembly process. In this work, only a part of the disassembly process is considered for simulation. Therefore, the focus is on the rotor of the motor, excluding the shaft. The aim is to retrieve the magnets via a robotic disassembly process.

[2] `https://www.dlr.de/en/rm/research/projects/matic-m`

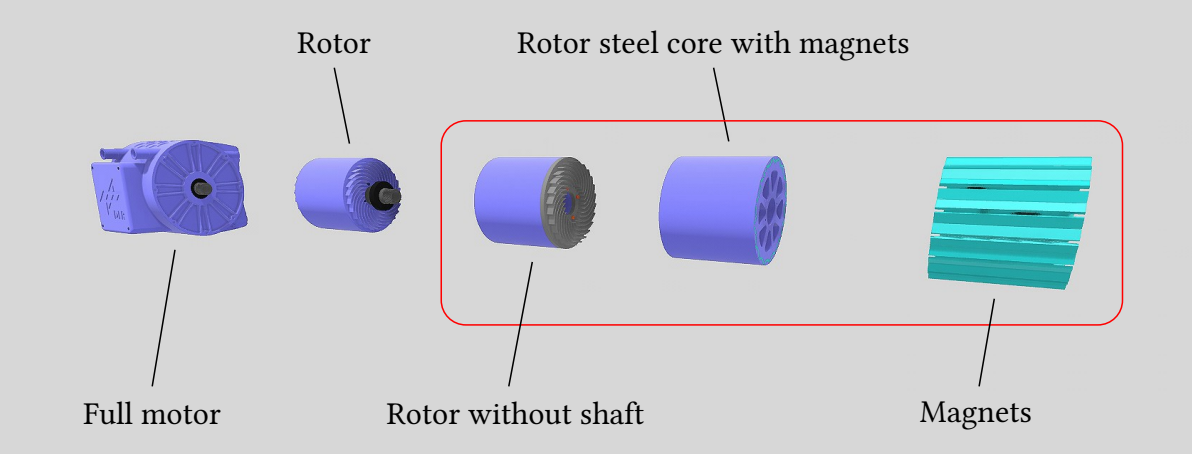

Figure 5.16: The electric motor in multiple steps during the assembly process. The full motor is shown on the left. Then, everything except the rotor is removed. The rotor without the shaft, the steel core with magnets, and the magnets themselves are marked in red. This marked part of the disassembly process is considered for modeling and simulation below. The 3D models are based on Wachter (2023).

The following steps are necessary in order to disassemble the magnets from the rotor core:

1. Disassembly of the rotor cover from the rotor steel core.
2. Heating of the rotor steel core to loosen the adhesive on the magnets.
3. Pressing out the magnets from the rotor steel core.

This application effectively demonstrates how the component models developed in Chapter 3 enable detailed multi-domain system simulations of assembly processes. For the first step, a disassembly process including screws must be modeled. Thermodynamic models, in the form of heat transfer models, are necessary for the second step. And a contact force model is required to model the third step.
A concept study will examine the power and energy required by each device in this process, as well as how the processes can be combined in terms of timing. For this purpose, a simulation model will be developed. The related visualization is illustrated in Figure 5.17. It should be noted that conceptual models are used, for example for the furnaces and the press.

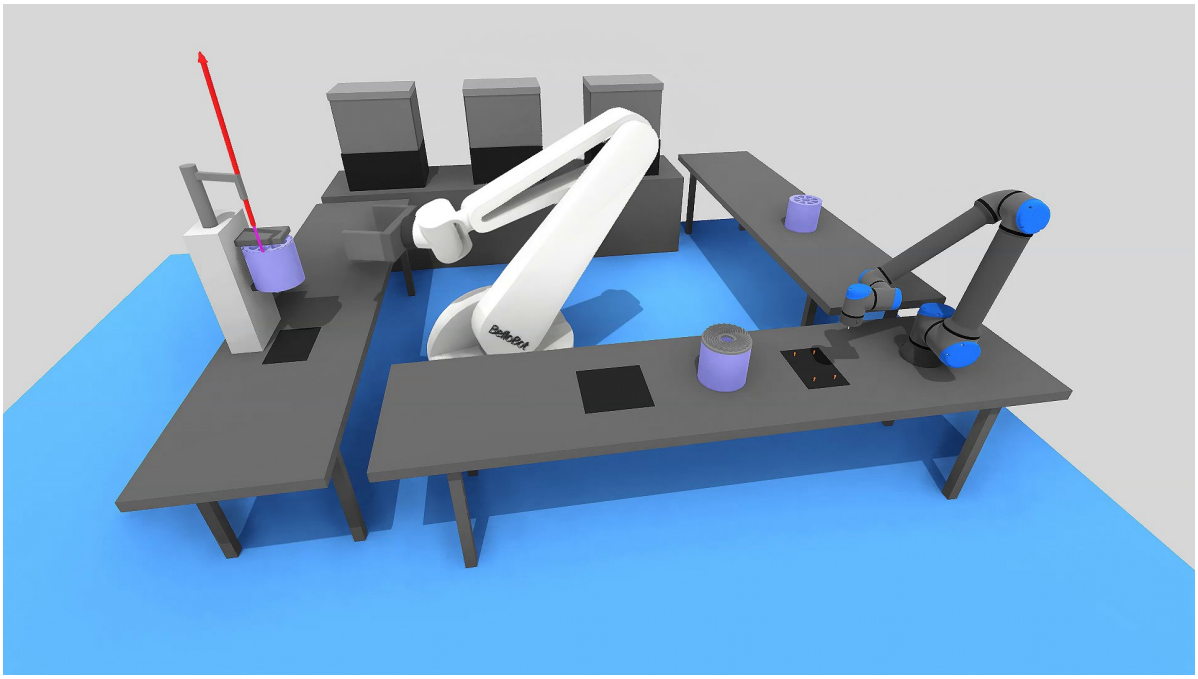

Figure 5.17: Visualization of the concept study of the disassembly process based on a multi-domain system simulation. The rotor cover is disassembled in the front. Then, the rotor steel core is heated in one of the furnaces in the back. Finally, the magnets are pressed out by a stamping press (on the left). A red arrow represents the contact force acting on the stamping press.

5.5.2 Multi-domain models

Component models from multiple domains are necessary to build the model for the disassembly process mentioned in Section 5.5.1: mechanics, thermodynamics, and electrical. The manipulation and assembly part is modeled using the component models developed in Chapter 3. As with the other applications, the robots are built with components from the *DLR Robots library* and the visualization is based on the *DLR Visualization 2 Library*. The remaining mechanical parts are multibody components from the MSL (including body and world). Additionally, the process of warming up the rotor steel core in the furnace is modeled using components from the thermal part of the MSL.
Because Modelica is an object-oriented modeling language for system simulations, it is possible to create a multi-domain model that covers all aspects of the process. Thus, components from the aforementioned libraries are combined into a single multi-domain model.
The **thermal model of the furnace heating process** is shown in Figure 5.18. It consists of heat capacities for the furnace, the rotor steel core (abbreviated as "rotor"), and the air inside of the furnace. The latter is used to map air exchange when the furnace door is opened. The furnace is heated by a heat flow, controlled by a PI controller. Heat transfer based on convection and radiation is assumed between the furnace and the rotor. For the heat transfer between the furnace and the air as well as between the furnace and the environment (heat loss), only convection is considered.
Since this is a concept study, the following parameters are only approximate. However, it is important that the values are within the correct order of magnitude. The following boundary conditions apply. It is assumed that the furnace is lined with stone on the inside and that heat conduction between the furnace and the rotor is negligible. Therefore, convection and radiation predominate between the furnace and the rotor. For other furnace models, the heat conduction between the furnace and the component may also need to be considered (depending on whether the component is placed directly on the furnace floor or on a wire rack). Radiation is only considered between the furnace and the rotor.
The furnace temperature is 300 °C and the environment temperature is 20 °C. The furnace is a cube with 0.70 m edges and a mass of 200 kg. The rotor is a cylinder with 0.10 m radius and 0.20 m height and a mass of 30 kg. The heat transfer coefficient from the furnace to the rotor and to the air is 15 $\frac{\mathrm{W}}{\mathrm{m}^2 \cdot \mathrm{K}}$, similar to the furnace heating example in Scherf (2011, p. 85). For the heat transfer from the furnace to the environment (heat loss), a heat transfer coefficient of 0.80 $\frac{\mathrm{W}}{\mathrm{m}^2 \cdot \mathrm{K}}$ is assumed. An emissivity of 0.65 is used for the radiation between the furnace and the rotor.
The furnace heating model is initialized when the furnace and air are already at 300 °C. When the furnace door is opened to add or remove the rotor, the temperature of the air inside the furnace is reinitialized to 20 °C. This is because it is assumed that air is fully exchanged with the environment. In Modelica, the `reinit()` command is used for this. Both opening times are predefined in the model by the user. For the rotor, the *HeatCapacitor* component from thermal part of the MSL is modified so that its capacity can be changed during runtime. The capacity of the rotor is based on its mass. The mass is 30 kg when the rotor is inside the

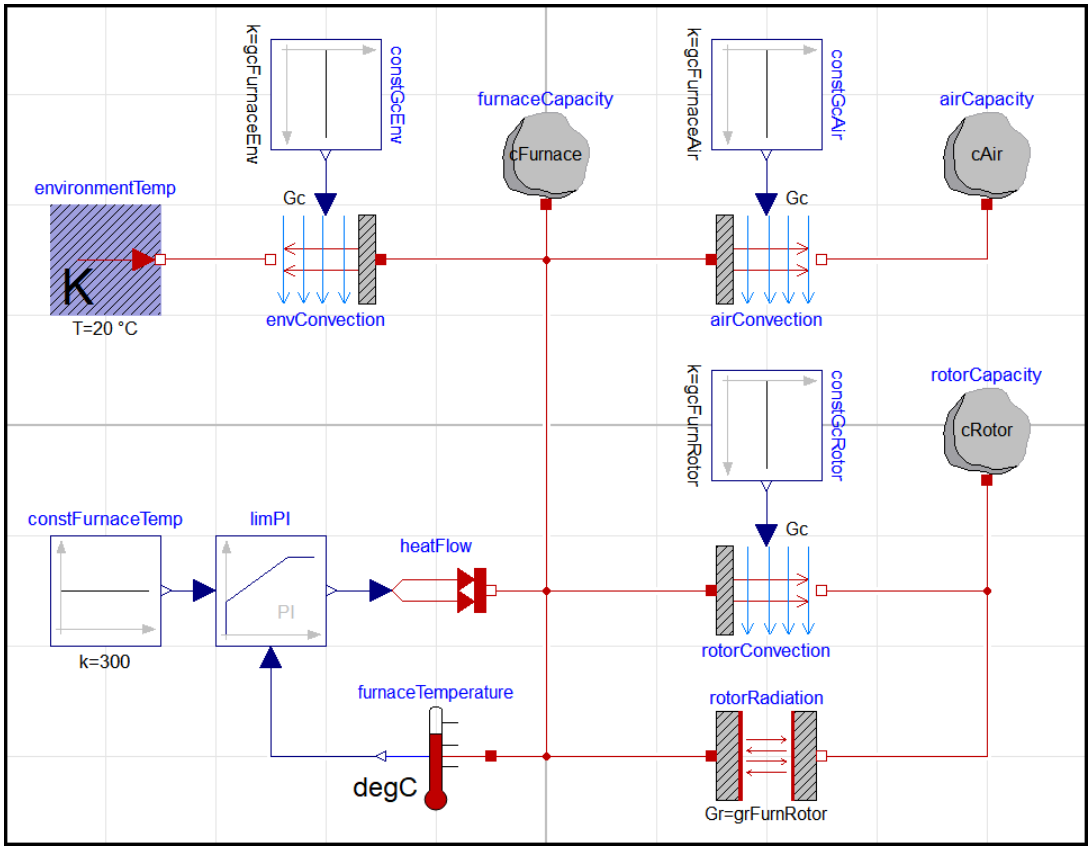

Figure 5.18: Object diagram of the furnace heating process model in *Dymola* for the electric motor disassembly use case. The model is based on components of the thermal part of the MSL. The furnace is modeled with a PI controller and heat losses to the environment. Heat transfer via convection and radiation is considered for the rotor, while for the air, only convection is considered.

furnace and almost zero otherwise. This is also predefined by the user. In addition, the rotor temperature is reinitialized when it is added to the furnace.
Figure 5.19 shows the temperature curves for a simulation time of 10000 s. In this simulation, the rotor is added to the furnace at time 1000 s and removed at time 8200 s. During its two hours inside the furnace, the rotor reaches a temperature of approximately 280 °C. The air temperature drops to 20 °C when the rotor is added and removed.
The corresponding power consumption of the furnace is presented in Figure 5.20. It takes approximately 660 W of power to maintain a temperature of 300 °C in the furnace. This is due to heat loss from the furnace to the surrounding environment.

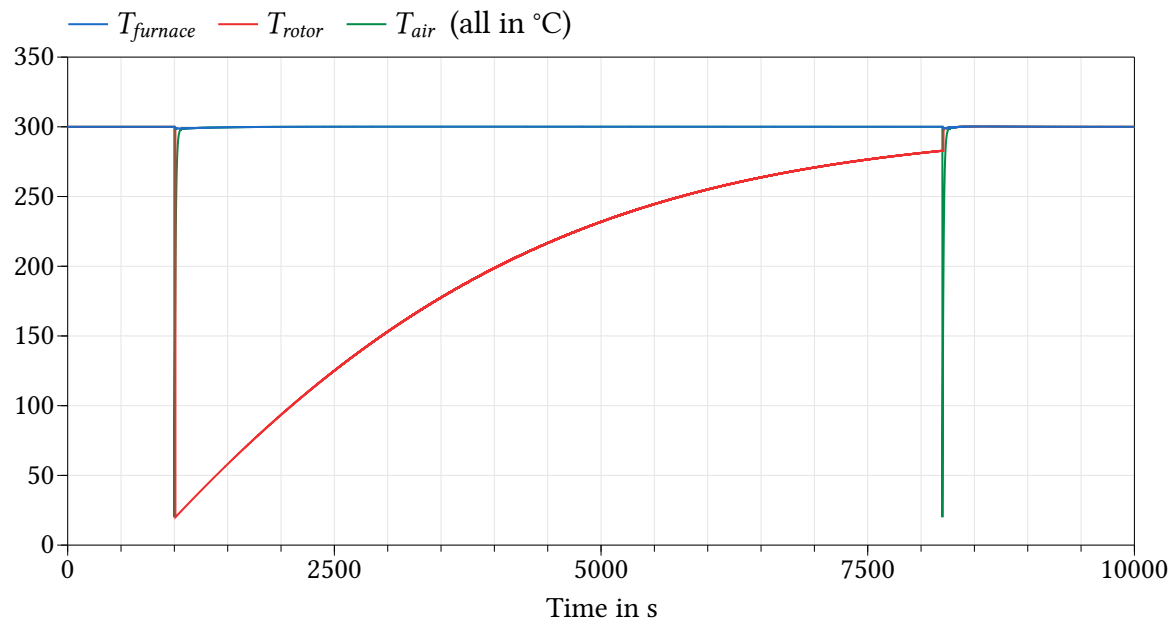

Figure 5.19: Temperature curves in the furnace heating process for the furnace, the rotor, and the air in the furnace. The rotor is added to the furnace at time 1000 s and removed at time 8200 s. After two hours, the rotor temperature reaches approximately 280 °C. The air temperature drops when the door is opened.

The **electrical model of the power consumption** for the robot is another aspect of the multi-domain model. According to Eggers (2019, p. 39), the idle power consumption of an industrial robot can be assumed to be 200 W. In this context, "idle" means the robot is not moving and the brakes are closed. The torque required to hold the axis in place (holding torque) is neglected. For the power consumption during the robot movement, a simplified model based on the mechanical power of each axis multiplied with a loss factor is used:

$$P_{robot,\ electrical} = k_{loss} \cdot P_{robot,\ mechanical} \tag{5.24}$$

$$\Rightarrow P_{robot,\ electrical} = k_{loss} \cdot t \cdot \omega \tag{5.25}$$

with k_{loss} being the loss factor, t being the torque, and ω being the angular velocity of the axis. The power consumption of the robot is then the sum of the power consumption of all axes. This model does not take recuperation and other energy-saving effects into account. For the present application, a loss factor of $k_{loss} = 1.3$ is assumed. The amount of electrical power required by the furnace is determined by the heat flow input. As a boundary condition, it is assumed that electric power is converted into heat flow at 100 % efficiency.
The **mechanical models** of the robots are developed in the same way as those for the other applications in this work. The same type of gripper that is mounted to the robot serves as the mount for the stamping press. A PI controller is used for the position control of the stamp.

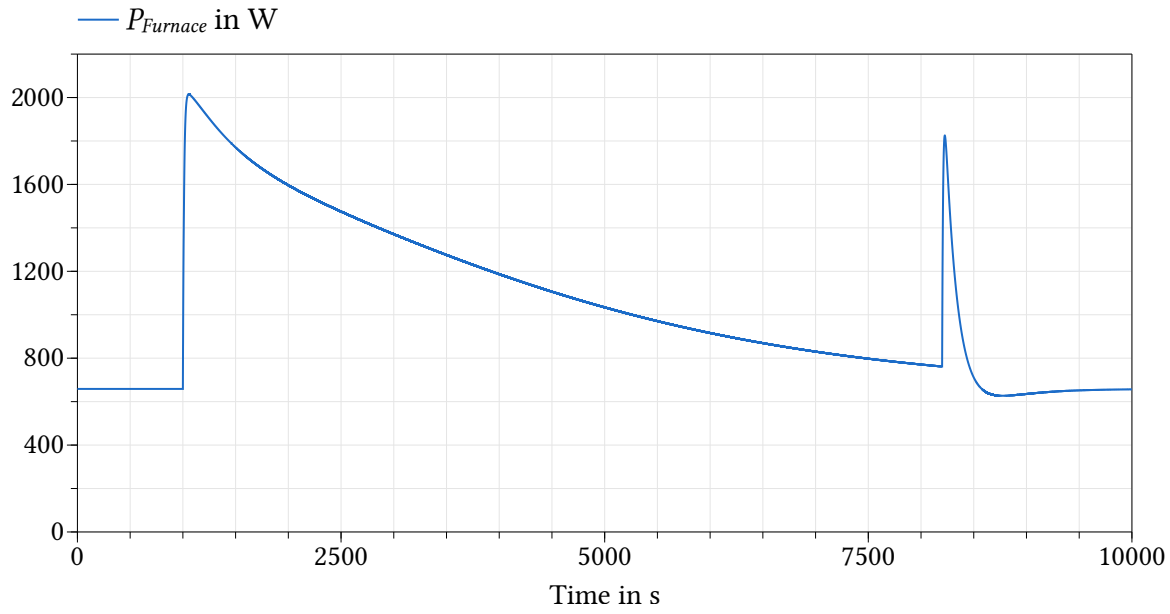

Figure 5.20: Furnace power consumption curve for the furnace heating process. The first peak in power is when the rotor is added to the furnace. The second peak results from the exchange of air in the furnace when the rotor is removed.

The start time of the stamp is set by the user. However, the 3D models must be modified for the contacts between the magnets and the rotor steel core. In contrast to the press fit between the motor and housing in Section 5.4, the magnets are attached to the rotor with an adhesive in a clearance fit. Because the PCM is a compliant contact model, it is not possible to attach the magnets in their original geometry because there is no intersection. Without an intersection, there is no contact force. The dimensions of the magnets can be increased slightly, though. Together with appropriate parameterization, this results in a holding force similar to that of the adhesive. This holding force is parameterized to correspond to the adhesive that has already been heated in the furnace. Simplified models are used for the screws. They are assembled to the assembly in the same way as the other components.
The masses are 30 kg for the rotor steel core, 1 kg for the rotor cover, and 0.1 kg for the magnets and screws. The dimension of the magnets is increased by the factor 1.0016. For the SFA, the translational spring constants are $6 \cdot 10^5 \frac{\mathrm{N}}{\mathrm{m}}$ for the rotor steel cores, $1 \cdot 10^5 \frac{\mathrm{N}}{\mathrm{m}}$ for the rotor cover, and $1 \cdot 10^4 \frac{\mathrm{N}}{\mathrm{m}}$ for the screws and magnets. The corresponding translational damping constants are 10000 $\frac{\mathrm{Ns}}{\mathrm{m}}$, 700 $\frac{\mathrm{Ns}}{\mathrm{m}}$, and 70 $\frac{\mathrm{Ns}}{\mathrm{m}}$. The following parameters are used for the PCM algorithm used for the contact between the magnet, the rotor steel core, and the stamping press: Young's modulus $E = 1 \cdot 10^6$ Pa, Poisson's ratio $\nu = 0.3$, coefficient of friction $\mu = 0.6$, friction regularization velocity $v_\epsilon = 0.001 \frac{\mathrm{m}}{\mathrm{s}}$, and layer thickness $d = 0.001$ m.
The object diagram of the **overall multi-domain model in Modelica** is shown in Figure 5.21. It incorporates models from three different domains.

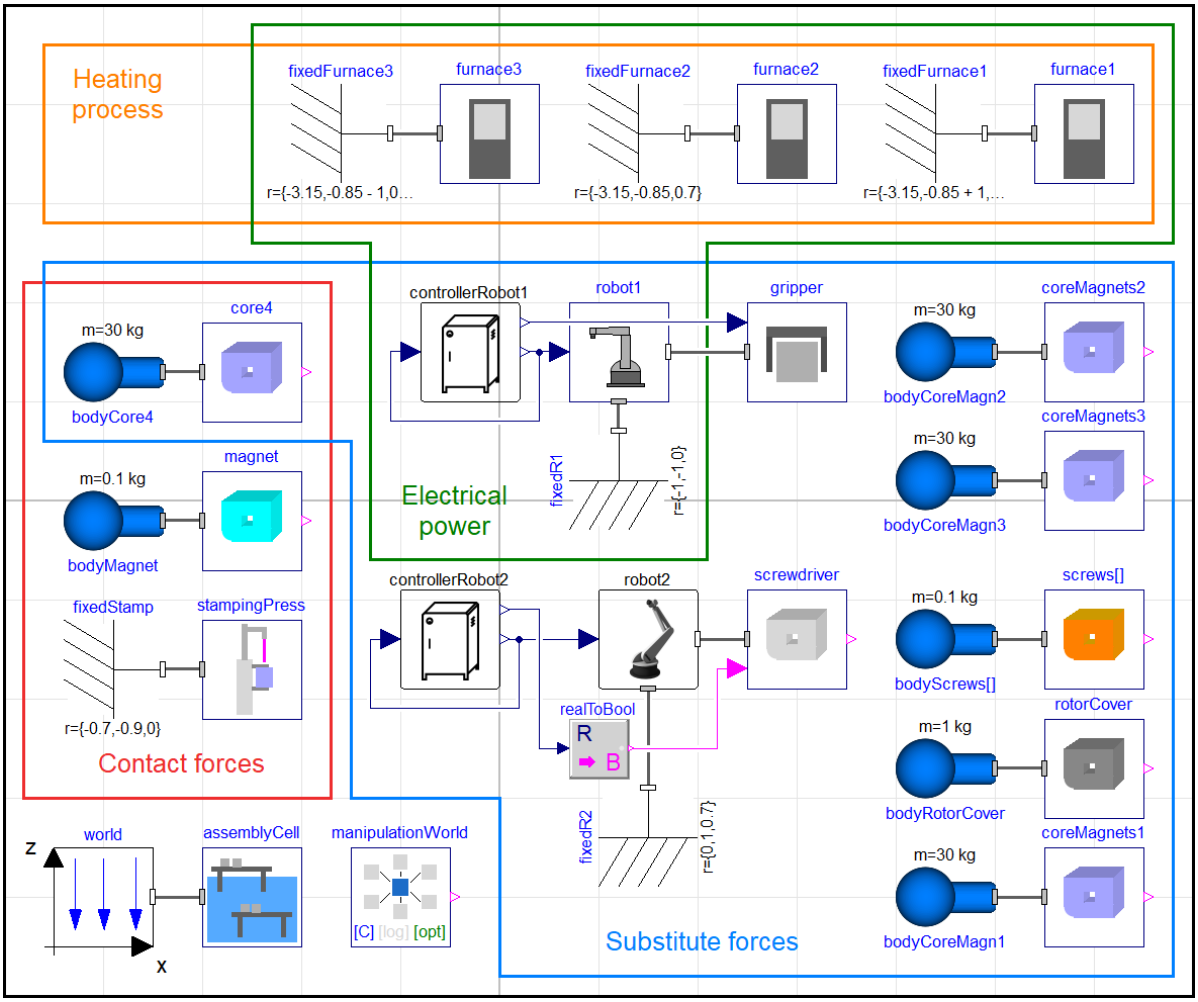

Figure 5.21: Object diagram of the multi-domain model in *Dymola* for the electric motor disassembly process. In the mechanical domain (marked in red and blue), robot kinematics (robot2) and dynamics (robot1) models are used, as well as substitute force (blue) and contact force (red) models. Component core4 interacts with both substitute forces (manipulation by robot1) and contact forces (pressing out the magnet). In addition, there are heat transfer models for the furnace heating process (orange) and electrical models for the power consumption (green).

The mechanical part consists of kinematics and dynamics models of the robots. For the collaborative robot (robot2), a kinematics model is used. The industrial robot (robot1) is described with a rigid body dynamics model, including a simplified model of the drivetrain and control. The interaction between components in terms of manipulation and assembly is modeled by both fast substitute models and detailed contact force models. The multi-domain model is therefore a hybrid model as described in Section 3.6.3. In the present model, the

contact forces are only relevant for the contacts in the stamping press (between the magnet, the stamping press, the rotor steel core, and the mount). Therefore, using all LsoD in the model results in faster computation times. Additionally, both substitute and contact forces are used for the component core4. The former are used for picking the component from furnace3 and placing it in the stamping press by robot1 and the latter are used for fixing and pressing out the magnet in the stamping press. Another advantage of hybrid models is that they can be created even if not all models are available. For instance, the present model does not require a dynamics model for robot2.
The model also includes a heat transfer part. The furnace heating process model (see Figure 5.18) represents the heating process of the rotor steel core in the furnace. It is part of the components furnace1, furnace2, and furnace3. The opening and closing times for all furnaces are set by the user. The same applies to the time when the rotor is added to or removed from a furnace. This is because the heat transfer model does not offer structural variability. However, these times can also be derived automatically from the robot program.
In addition, the model includes an electrical part that describes the power consumption of the industrial robot (robot1) and the three furnaces. The models for the power consumption are parts of the related component models.
Modeling all domains in one model has advantages. The resulting model is more flexible for changes and there is less work for integration.

5.5.3 Results from the simulation

A conceptual disassembly process is described in the following. A visualization of multiple steps in the disassembly process is shown in Figure 5.22. First, robot2 removes the screws from the rotor with cover on the front table (15 s). Then, robot1 picks a rotor from the right table and places it in furnace1 (35 s). Furthermore, robot1 removes a rotor that has already been heated from furnace3 and places it in the stamp press (56 s). Here, the magnet is pressed out using a contact force (66 s to 71 s). Finally, robot1 removes the rotor cover from the rotor on the front table (82 s). The multi-domain simulation for this process has a simulation time of 90 s and is based on the model presented in Figure 5.21 and Figure 5.18.
The simulation was successfully run for both LoD 1 and LoD 3. A *Rkfix2* solver with a fixed step size of 0.002 s was used for the former and a *Cvode* solver with $1 \cdot 10^{-5}$ tolerance for the latter. The computation times are 41.1 s for the simulation at LoD 1 and 1099 s for the

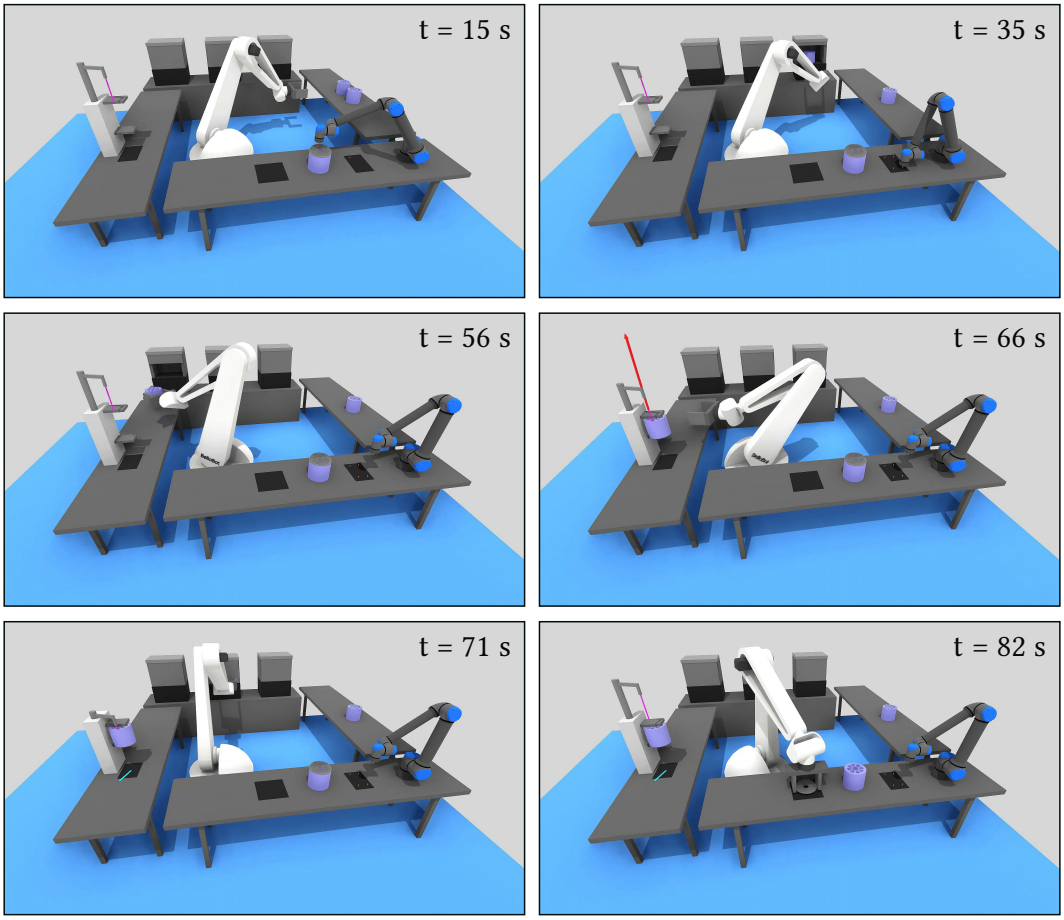

Figure 5.22: Visualization of the simulation of the electric motor disassembly process for multiple steps. The simulation is a concept study and based on a multi-domain model, including mechanics, thermodynamics, and electrical.

simulation at LoD 3. Additionally, the simulation at LoD 2 was run with a computation time of 77.6 s (same settings as the simulation at LoD 3). The simulations at LoD 2 and at LoD 3 can be used to determine the power consumption of the industrial robot and the furnaces. With the simulation at LoD 3, the force acting on the stamping press for pressing out the magnet can be calculated in addition.
The power consumption curves for the industrial robot (robot1), furnace1, and furnace3 are presented in Figure 5.23. More power is required for furnace 1 because a rotor steel core at room temperature is placed inside it during the simulation. In contrast, furnace 3 only needs to heat the replaced air because a preheated rotor steel core is removed from it.

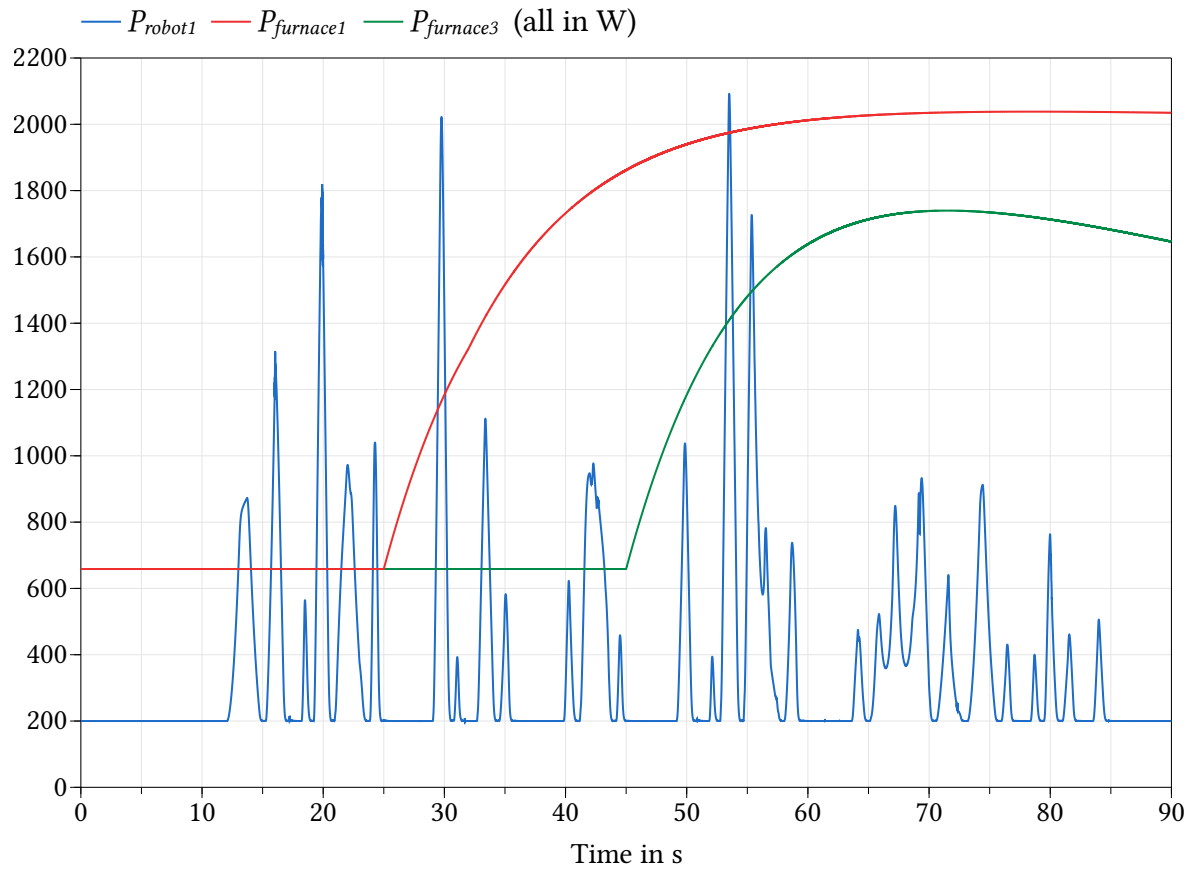

Figure 5.23: Power consumption for the industrial robot and two furnaces in the simulation of the disassembly process of an electric motor. This simulation is based on a multi-domain model and serves as a concept study. For furnace1, the peak in power consumption occurs when the cold rotor steel core is added to the furnace. In contrast, a rotor that has already been heated is removed from furnace3, and only the air is exchanged there.

6 Conclusion and outlook

In this final chapter, the work is summed up and a conclusion is given. In addition, an outlook is provided in the form of potential future developments.

6.1 Conclusion

This work is a contribution to the field of the simulation of assembly and disassembly processes at the system level. Detailed simulations require the consideration of several domains, such as models of the robot drivetrains, including the electric motors and their energy consumption. Multidomain models are particularly relevant for automatic disassembly processes, as in some cases parts must be heated in order to be disassembled.
An analysis of the state of the art revealed that there are many simulation tools available for various tasks in robotic assembly simulation. However, most of these robot simulation tools are limited to the mechanical domain and use physics engines with non-smooth contact models and specialized solvers based, for example, on time-stepping methods. These approaches are not well suited for stiff components in multi-domain models, such as those found in detailed drivetrain models. For detailed simulations of assembly and disassembly processes with multiple domains, system simulation tools such as *Modelica* environments or *Simscape* are required. These tools are based on object-oriented modeling languages and event-based solvers. However, system simulation tools are typically structure-invariant and do not support dynamic structural changes of physical components during runtime. However, this capability is required for assembly and disassembly. In addition, these tools usually do not support the modeling of contacts between complex, non-convex, rigid bodies.
In this work, a novel solution for the modeling and simulation of manipulation and assembly processes in system simulation tools was developed. The objective is to enable detailed, multi-domain simulations of these processes at the system level in a stable, efficient, and highly-flexible manner. The solution must be able to handle complex, non-convex geometries and should require minimal modeling effort.
The developed solution is based on compliant contact models and the automatic generation and management of component pairs. It extends the object-oriented modeling language of a system simulation tool with specialized models for assembly and disassembly at two Levels of Detail: fast substitute models and detailed contact force models. Both models are integrated into reusable component models, reducing the effort required for modeling assembly and disassembly processes. Additionally, the developed extended modeling framework provides an intuitive description of assembly processes and assembly hierarchies, eliminating the need to manually model all potential contact pairs.
Several contributions were presented in this work to enable the stable and efficient simulation of manipulation and assembly processes in system simulation tools. First, a general

concept was developed and component types and their interactions were defined. The *Block Scenario* example was introduced as a basis to describe the developed algorithms and models. Moreover, the Polygonal Contact Model was selected as the most suitable contact force algorithm for the simulation of manipulation and assembly processes in system simulation tools. It is based on compliant, discretized contact surfaces.
Then, the main contributions were presented. The Algorithm for Component Interaction serves as the basis for the interactions between components in manipulation and assembly processes. It is used for both the substitute models and the contact force models. The Substitute Force Algorithm was developed to overcome the limitations of structure-invariant environments. This algorithm allows physical components to be connected or disconnected during runtime and is based on contact forces that hold the components in place.
The Substitute Force Algorithm is sufficient for individual connections between components. However, in order to simulate assembly processes, the algorithm had to be extended to include hierarchy levels and connection prioritization. Even with an extension for manipulation and assembly, the algorithm was insufficient for simulating complex assemblies. Therefore, an additional algorithm for the robust handling of hierarchical contacts was developed. The Assembly Graph Partner Search Algorithm enables the stable simulation of assemblies. Using the Substitute Force Algorithm in assemblies can result in contact forces connected in series or contact forces connected in parallel. In contrast, the Assembly Graph Partner Search Algorithm searches for the related gripper or table for each component of an assembly and applies a single (virtual) contact force to that component approximating the various contact forces present in the assembly.
To reduce the effort generally required to create new models, both the fast substitute models and the detailed contact models were combined into component models. This way, a model of a specific assembly process needs to be created only once using these components. The so built process model can then be used for multiple tasks, including Virtual Commissioning in real-time and process optimization with detailed contact forces. This significantly reduces the modeling effort. Multiple Levels of Detail were defined for this purpose.
Moreover, the developed solution was realized in *Modelica* and *C++*. The object-oriented, multi-domain modeling language *Modelica* is used in the system simulation environment *Dymola* because its acausal component models offer a high degree of flexibility with full access to the equations describing the physical behavior. The solution developed in this work is provided as *Modelica* component models extended by an External Environment containing the presented algorithms. Additional software packages, such as *libccd* for the collision detection required by the Substitute Force Algorithm, and the Polygonal Contact Model, were integrated into the External Environment as well. The correct interaction between the *Modelica* model and the External Environment was ensured so that the algorithms are only evaluated once the positions and orientations have been updated for all components. Additionally, optimizations were made to reduce the computation time.
The developed solution has been successfully applied to several applications. First, the *Block Scenario* example was successfully simulated at all Levels of Detail using multiple fixed-step and variable-step solvers. Using this example, the suitability of the method for simulating

disassembly processes was also demonstrated.
Then, a comprehensive analysis was performed based on the *Block Scenario* example. A comparison between the fast substitute models and the contact models showed that the former reduced the calculation time in this example by 95 % while resulting in a position error of less than 0.00025 m. In another comparison, the advantages of the Assembly Graph Partner Search Algorithm over the Substitute Force Algorithm were demonstrated. Using the former can either reduce the position error by 76 % with the same step size or reduce the computation time by 76 % with an increased step size, which is impossible with the latter.
The analysis also showed that the optimizations significantly reduced computation time. Using the parallelization of the Polygonal Contact Model evaluation function calls reduced the computation time by 47 %. Reducing the number of function calls reduced the computation time by 62 %. Both optimizations combined reduced the computation time by 74 %.
Additionally, the *Block Scenario* example was successfully simulated using two KUKA KR16 robots. Here, detailed drivetrain models were used, including controls, gears with friction, and electric motors. This application showed that the models developed in this work can be combined with detailed multi-domain models such as those of the robot drivetrain.
Furthermore, the process of assembling an electric motor into a chainsaw housing was successfully simulated. This application involves complex geometries, such as those of the housing and box. The meshes had to be modified slightly for the contact model and convex decompositions had to be generated for collision detection.
Finally, the component models were applied to a comprehensive multi-domain model. For this, a concept study of the disassembly process of an electric motor from a Battery Electric Vehicle was used. This study aims to investigate the recycling of electric motors. Part of this process involves removing the magnets from the rotor. In order to disassemble the magnets, the rotor must first be heated in a furnace to weaken the adhesive. Therefore, the overall model contains a thermodynamic model describing the furnace heating process. In addition, the power consumption of the robot and furnaces is modeled with additional electrical models. The resulting multi-domain model covers the manipulation and assembly processes at two Levels of Detail as well as the furnace heating process and the power consumption of the involved machines. The model can be used to estimate the overall power consumption and process times of the combined disassembly and heating processes.
These applications have demonstrated the suitability of the developed solution for simulation of manipulation and assembly processes in detail at the system level across multiple domains.

6.2 Future work

The Substitute Force Algorithm uses the Minkowski Portal Refinement collision detection algorithm. The computation time of the latter could be reduced by using Bounding Volume Hierarchies, similar to those used in the Polygonal Contact Model. This would significantly reduce the computing time, especially for complex geometries. When using convex decomposition, it may also be helpful to preselect which convex parts could collide.

In addition, for each component pair, a Minkowski Portal Refinement collision detection evaluation is performed. These evaluations could be calculated in parallel, similar to the parallelization of the Polygonal Contact Model contact evaluations. This would further reduce the computing time. This is especially relevant when using convex decompositions because all combinations of convex parts must be evaluated for each component pair.
In the Polygonal Contact Model algorithm, the geometric description of one component is converted into the frame of the other component for each component pair. For a large number of components and therefore a large number of component pairs, it could be more efficient to convert the geometric descriptions of all components into the world frame.
Finally, the Pressure Field Contact model could be investigated as an alternative to the Polygonal Contact Model and a (semi) automatic generation of the process models from existing CAD models could reduce the modeling effort even further.

Bibliography

Andersson, Sören, Anders Söderberg, and Stefan Björklund (2007). "Friction models for sliding dry, boundary and mixed lubricated contacts". In: *Tribology International* 40.4, pp. 580–587. DOI: 10.1016/j.triboint.2005.11.014.

Bardaro, Gianluca, Luca Bascetta, Francesco Casella, and Matteo Matteucci (2017). "Using Modelica for advanced Multi-Body modelling in 3D graphical robotic simulators". In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 887–894. DOI: 10.3384/ecp17132887.

Bellmann, Tobias, Andreas Seefried, and Bernhard Thiele (2020). "The DLR Robots library - Using replaceable packages to simulate various serial robots". In: *Proceedings of Asian Modelica Conference 2020, Tokyo, Japan, October 08-09, 2020*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 153–161. DOI: 10.3384/ecp2020174153.

Bender, Edward A. and S. Gill Williamson (2010). *Lists, Decisions and Graphs. With an Introduction to Probability*. San Diego: University of California.

Bender, Jan, Kenny Erleben, and Jeff Trinkle (2012). "Interactive Simulation of Rigid Body Dynamics in Computer Graphics". In: *Proceedings of EG 2012 - State of the Art Reports (2012), Eurographics Association*. Eurographics Association, pp. 95–134.

Bender, Jan, Kenny Erleben, and Jeff Trinkle (2014). "Interactive Simulation of Rigid Body Dynamics in Computer Graphics". In: *Computer Graphics Forum* 33.1, pp. 246–270. DOI: 10.1111/cgf.12272.

Bezanson, Jeff, Alan Edelman, Stefan Karpinski, and Viral B. Shah (2017). "Julia: A Fresh Approach to Numerical Computing". In: *SIAM Review* 59.1, pp. 65–98. DOI: 10.1137/141000671.

Blochwitz, Torsten, Martin Otter, Johan Akesson, Martin Arnold, Christoph Clauß, et al. (2012). "Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models". In: *Proceedings of the 9th International Modelica Conference, September 3-5, 2012, Munich, Germany*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 173–184. DOI: 10.3384/ecp12076173.

Blochwitz, Torsten, Martin Otter, Martin Arnold, Constanze Bausch, Christoph Clauß, et al. (2011). "The Functional Mockup Interface for Tool independent Exchange of Simulation Models". In: *Proceedings of the 8th International Modelica Conference, March 20-22, Dresden, Germany*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 105–114. DOI: 10.3384/ecp11063105.

Bortoff, Scott A. (2020). "Modeling Contact and Collisions for Robotic Assembly Control". In: *Proceedings of the American Modelica Conference 2020, Boulder, Colorado, USA, March 23-25, 2020*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 54–63. DOI: 10.3384/ecp2016954.

Buse, Fabian and Tobias Bellmann (2021). "General Purpose Lua Interpreter for Modelica". In: *Proceedings of the 14th Modelica Conference 2021, Linköping, Sweden, September 20-24, 2021*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 425–431. DOI: 10.3384/ecp21181425.

Buse, Fabian, Antoine Pignède, and Stefan Barthelmes (2023). "A Modelica Library to Add Contact Dynamics and Terramechanics to Multi-Body Mechanics". In: *Proceedings of the 15th International Modelica Conference, Aachen, Germany, October 09-11, 2023*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 433–442. DOI: 10.3384/ecp204433.

Clauberg, Patrick Jan (2013). "Methoden zur parallelen Berechnung von nicht-glatten dynamischen Systemen". PhD thesis. Technische Universität München. URL: https://mediatum.ub.tum.de/?id=1111045.

Cormen, Thomas H., Charles Eric Leiserson, Ronald Linn Rivest, and Clifford Stein (2022). *Introduction to Algorithms*. 4th ed. Cambridge: The MIT Press. ISBN: 9780262367509.

Corral, Eduardo, Raúl Gismeros Moreno, M. J. Gómez García, and Cristina Castejón (2021). "Nonlinear phenomena of contact in multibody systems dynamics: a review". In: *Nonlinear Dynamics* 104.2, pp. 1269–1295. DOI: 10.1007/s11071-021-06344-z.

Coumans, Erwin (2024). *Bullet Real-Time Physics Simulation*. URL: https://pybullet.org/wordpress/ (visited on 2025-03-26).

Dassault Systèmes (2024). *DELMIA. Make It Happen. DELMIA empowers manufacturing, supply chain and service providers to efficiently plan, manage, optimize, and execute their operations.* URL: https://www.3ds.com/products/delmia (visited on 2025-03-26).

Dassault Systèmes (2025). *Dymola. Multi-Engineering Modeling and Simulation based on Modelica and FMI*. URL: https://www.3ds.com/products/catia/dymola (visited on 2025-03-26).

De Marco, Francesco (2023). "Comparative study of multi-physics modelling and simulation software for lifetime performance evaluation of battery systems". MA thesis. Politecnico di Torino. URL: https://webthesis.biblio.polito.it/27416/.

Diankow, Rosen (2010). "Automated Construction of Robotic Manipulation Programs". PhD thesis. Carnegie Mellon University. DOI: 10.1184/R1/6714905.v1.

Drumwright, Evan and Jeffrey C. Trinkle (2017). "Contact Simulation". In: *Humanoid Robotics: A Reference*. Ed. by Ambarish Goswami and Prahlad Vadakkepat. Dordrecht: Springer Netherlands, pp. 1–55. DOI: 10.1007/978-94-007-7194-9_25-1.

Eggers, Kai Benjamin (2019). "Energieeffizienter Betrieb von Industrierobotern". PhD thesis. Universität Hannover. DOI: 10.15488/5332.

Elandt, Ryan, Evan Drumwright, Michael Sherman, and Andy Ruina (2019). "A pressure field model for fast, robust approximation of net contact force and moment between nominally rigid objects". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Macau, China). IEEE, pp. 8238–8245. DOI: 10.1109/iros40897.2019.8968548.

Ellekilde, Lars-Peter and Jimmy A. Jorgensen (2010). "RobWork: A Flexible Toolbox for Robotics Research and Education". In: *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*. VDE. Berlin: VDE Verlag, pp. 800–806. ISBN: 9783800732739.

Elmqvist, Hilding, Axel Goteman, Vilhelm Roxling, and Toheed Ghandriz (2015). "Generic Modelica Framework for MultiBody Contacts and Discrete Element Method". In: *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 427–440. DOI: 10.3384/ecp15118427.

Elmqvist, Hilding, Martin Otter, and François E. Cellier (1995). "Inline Integration: A New Mixed Symbolic/Numeric Approach for Solving Differential-Algebraic Equations Systems". In: *ESM'95, European Simulation Multiconference, Prag, 5.-8. Juni 1995*.

Erez, Tom, Yuval Tassa, and Emanuel Todorov (2015). "Simulation Tools for Model-Based Robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)* (Seattle, WA, USA). IEEE, pp. 4397–4404. DOI: 10.1109/icra.2015.7139807.

Ericson, Christer (2004). *Real-Time Collision Detection*. Morgan Kaufmann Series in Interactive 3D Technology. San Francisco: Elsevier. ISBN: 1558607323.

ESI Group (2024). *SimulationX. Multi-domain System Simulation Software*. URL: https://www.esi-group.com/products/simulationx (visited on 2025-03-26).

Fiser, Daniel (2018). *libccd. Library for collision detection between two convex shapes*. URL: https://github.com/danfis/libccd (visited on 2025-03-26).

Flores, Paulo (2022). "Contact mechanics for dynamical systems: a comprehensive review". In: *Multibody System Dynamics* 54.2, pp. 127–177. DOI: 10.1007/s11044-021-09803-y.

Flores, Paulo and Hamid M. Lankarani (2016). *Contact Force Models for Multibody Dynamics*. Solid Mechanics and Its Applications. Cham: Springer. ISBN: 9783319308975.

Frieben, Tanja, Marcel Schneider, Jürgen Gausemeier, and Ansgar Trächtler (2015). "Virtuelle Inbetriebnahme mit wählbarer Modellierungstiefe: Verkürzung der Entwicklungszeit bis zum Start-of-Production". In: *Zeitschrift für wirtschaftlichen Fabrikbetrieb* 110.4, pp. 227–232. DOI: 10.3139/104.111316.

Fritzson, Peter, Adrian Pop, Karim Abdelhak, Adeel Ashgar, Bernhard Bachmann, et al. (2020). "The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development". In: *Modeling, Identification and Control* 41.4, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

Gilbert, Elmer G., Daniel W. Johnson, and S. Sathiya Keerthi (1988). "A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space". In: *IEEE Journal on Robotics and Automation* 4.2, pp. 193–203. DOI: 10.1109/56.2083.

Gross, Dietmar, Werner Hauger, Jörg Schröder, and Wolfgang A. Wall (2015). *Technische Mechanik 3. Kinetik*. 13th ed. Springer-Lehrbuch. Berlin, Heidelberg: Springer. ISBN: 9783642539541.

Gulan, Stefan, Ulrich Odefey, Thomas Bär, Herbert Beesten, Holger Hämmerle, Bernd Kärcher, and Matthias Riedl (2014). "Physikbasierte Simulation im Anlagenbau". In: *ASIM-Workshop STS/GMMS 2014 Tagungsband*, pp. 57–62. ISBN: 9783901608421.

Heckmann, Andreas, Martin Otter, Stefan Dietz, and José Díaz López (2006). "The DLR FlexibleBody library to model large motions of beams and of flexible bodies exported from finite element programs". In: *Proceedings of the 5th International Modelica Conference, September 4-5, 2006, Vienna, Austria*, pp. 85–95. URL: https://elib.dlr.de/47219/.

Hellerer, Matthias and Fabian Buse (2017). "Compile-time dynamic and recursive data structures in Modelica". In: *Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT'17)*. New York: ACM, pp. 81–86. DOI: 10.1145/3158191.3158205.

Hertz, Heinrich (1882). "Ueber die Berührung fester elastischer Körper". In: *Journal für die reine und angewandte Mathematik* 1882.92, pp. 156–171. DOI: 10.1515/crll.1882.92.156.

Hindmarsh, Alan C., Peter N. Brown, Keith E. Grant, Steven L. Lee, Radu Serban, Dan E. Shumaker, and Carol S. Woodward (2005). "SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers". In: *ACM Transactions on Mathematical Software* 31.3, pp. 363–396. DOI: 10.1145/1089014.1089020.

Hippmann, Gerhard (2004a). "An Algorithm for Compliant Contact Between Complexly Shaped Bodies". In: *Multibody System Dynamics* 12.4, pp. 345–362. DOI: 10.1007/s11044-004-2513-4.

Hippmann, Gerhard (2004b). "Modellierung von Kontakten komplex geformter Körper in der Mehrkörperdynamik". PhD thesis. Technische Universität Wien.

Hippmann, Gerhard (2024). "Polygonal contact model revisited: notes on usage and improved implementation". In: *Multibody System Dynamics* 60.2, pp. 219–231. DOI: 10.1007/s11044-023-09895-8.

Hofmann, Andreas, Lars Mikelsons, Ines Gubsch, and Christian Schubert (2014). "Simulating Collisions within the Modelica MultiBody library". In: *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 949–957. DOI: 10.3384/ecp14096949.

Hoher, Simon (2016). *Ein gekoppeltes Materialflussmodell zur durchgängigen Entwicklungsunterstützung von Materialflusssteuerungen*. Dissertation, Universität Stuttgart. Stuttgarter Beiträge zur Produktionsforschung. Stuttgart: Fraunhofer Verlag. ISBN: 9783839611975.

Hu, Zeguang and Dingxuan Zhao (2024). "Dynamics and simulation of walking excavator chassis based on the virtual work principle". In: *Multibody System Dynamics*. DOI: 10.1007/s11044-024-10016-2.

Hummel, Johannes, Robin Wolff, Tobias Stein, Andreas Gerndt, and Torsten Kuhlen (2012). "An Evaluation of Open Source Physics Engines for Use in Virtual Reality Assembly Simulations". In: *Advances in Visual Computing. 8th International Symposium, ISVC 2012*. Berlin, Heidelberg: Springer, pp. 346–357. DOI: 10.1007/978-3-642-33191-6_34.

ISG Industrielle Steuerungstechnik (2024). *ISG-virtuos. Die Simulationsplattform für digitale Zwillinge zur virtuellen Inbetriebnahme*. URL: https://www.isg-stuttgart.de/produkte/softwareprodukte/isg-virtuos (visited on 2025-03-26).

Iskandar, Maged, Oliver Eiberger, Alin Albu-Schaffer, Alessandro De Luca, and Alexander Dietrich (2021). "Collision Detection, Identification, and Localization on the DLR SARA Robot with Sensing Redundancy". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3111–3117. DOI: 10.1109/icra48506.2021.9561677.

Johnson, K. L. (1985). *Contact mechanics*. Cambridge: Cambridge University Press. DOI: 10.1017/cbo9781139171731.

Junghanns, Andreas, Torsten Blochwitz, Christian Bertsch, Torsten Sommer, Karl Wernersson, et al. (2021). "The Functional Mock-up Interface 3.0 - New Features Enabling New Applications". In: *Proceedings of the 14th Modelica Conference 2021, Linköping, Sweden, September 20-24, 2021*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 17–26. DOI: 10.3384/ecp2118117.

Kennedy, Christopher A. and Mark H. Carpenter (2019). "Diagonally implicit Runge-Kutta methods for stiff ODEs". In: *Applied Numerical Mathematics* 146, pp. 221–244. DOI: 10.1016/j.apnum.2019.07.008.

Koch, Philip, Nico Töpfer, Sebastian Albrecht, Bernhard Thiele, and Robert Reiser (2025). "Modular production using hierarchical planning and a generic OPC UA skills concept". In: *2025 IEEE 21st International Conference on Automation Science and Engineering (CASE)*. IEEE, pp. 2347–2354. DOI: 10.1109/case58245.2025.11164145.

Koenig, Nathan and Andrew Howard (2004). "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Sendai, Japan). IEEE, pp. 2149–2154. DOI: 10.1109/iros.2004.1389727.

Köslin, Fabian, Philipp Puntel-Schmidt, Willi Riediger, and Alexander Fay (2014). "Entwurf einer Modelica Simulationsbibliothek für die virtuelle Inbetriebnahme fertigungstechnischer Anlagen". In: *7th International Symposium on Automatic Control, AUTSYM 2014, 25.-26.09.2014, Wismar*.

Kübler, Karl, Alexander Verl, Oliver Riedel, and Michael Oberle (2017). "Simulation-assisted run-to-run control for battery manufacturing in a cloud environment". In: *2017 24th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)* (Auckland, New Zealand). IEEE, pp. 1–6. DOI: 10.1109/m2vip.2017.8211450.

Kümper, Sebastian, Matthias Hellerer, and Tobias Bellmann (2021). "DLR Visualization 2 Library - Real-Time Graphical Environments for Virtual Commissioning". In: *Proceedings of the 14th Modelica Conference 2021, Linköping, Sweden, September 20-24, 2021*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 197–204. DOI: 10.3384/ecp21181197.

Lacour, Frédéric-Felix (2012). *Modellbildung für die physikbasierte Virtuelle Inbetriebnahme materialflussintensiver Produktionsanlagen*. Dissertation, Technische Universität München, 2011. Forschungsberichte IWB. München: Utz. ISBN: 9783831641628.

Lechler, Tobias, Eva Fischer, Maximilian Metzner, Andreas Mayr, and Jörg Franke (2019). "Virtual Commissioning - Scientific review and exploratory use cases in advanced production systems". In: *Procedia CIRP* 81.3, pp. 1125–1130. DOI: 10.1016/j.procir.2019.03.278.

León, Beatriz, Stefan Ulbrich, Rosen Diankov, Gustavo Puche, Markus Przybylski, et al. (2010). "OpenGRASP: A Toolkit for Robot Grasping Simulation". In: *Simulation, Modeling, and Programming for Autonomous Robots. Second International Conference, SIMPAR 2010*. Berlin, Heidelberg: Springer, pp. 109–120. DOI: 10.1007/978-3-642-17319-6_13.

Lotter, Bruno and Hans-Peter Wiendahl (2012). *Montage in der industriellen Produktion. Ein Handbuch Für Die Praxis*. 2nd ed. Berlin, Heidelberg: Springer. ISBN: 9783642290602.

Lüder, Arndt and Nicole Schmidt (2024). "AutomationML in a Nutshell". In: *Handbuch Industrie 4.0. Band 2: Automatisierung*. Ed. by Birgit Vogel-Heuser, Michael ten Hompel, and

Thomas Bauernhansl. Berlin, Heidelberg: Springer, pp. 827–874. DOI: 10.1007/978-3-662-58528-3_61.

Machado, Margarida, Pedro Moreira, Paulo Flores, and Hamid M. Lankarani (2012). "Compliant contact force models in multibody dynamics: Evolution of the Hertz contact theory". In: *Mechanism and Machine Theory* 53, pp. 99–121. DOI: 10.1016/j.mechmachtheory.2012.02.010.

Mainzer, David (2015). "New geometric algorithms and data structures for collision detection of dynamically deforming objects". PhD thesis. Technische Universität Clausthal. DOI: 10.21268/20151117-122639.

Mammou, Khaled (2016). "Volumetric Hierarchical Approximate Convex Decomposition". In: *Game Engine Gems 3*. Ed. by Eric Lengyel. Boca Raton: CRC Press, pp. 141–158. ISBN: 9781498755665. URL: https://github.com/kmammou/v-hacd.

Masterjohn, Joseph, Damrong Guoy, John Shepherd, and Alejandro Castro (2022). "Velocity Level Approximation of Pressure Field Contact Patches". In: *IEEE Robotics and Automation Letters* 7.4, pp. 11593–11600. DOI: 10.1109/lra.2022.3203845.

MathWorks (2023). *Spatial Contact Force. Simscape Multibody Documentation*. URL: https://www.mathworks.com/help/sm/ref/spatialcontactforce.html (visited on 2025-03-26).

MathWorks (2024). *Simscape. Model and simulate multidomain physical systems*. URL: https://www.mathworks.com/products/simscape.html (visited on 2025-03-26).

McNeely, William A., Kevin D. Puterbaugh, and James J. Troy (2005). "Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling". In: *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses* (Los Angeles, California). New York: ACM, pp. 42–49. DOI: 10.1145/1198555.1198605.

Miller, Andrew T. and Peter K. Allen (2004). "Graspit! A Versatile Simulator for Robotic Grasping". In: *IEEE Robotics & Automation Magazine* 11.4, pp. 110–122. DOI: 10.1109/mra.2004.1371616.

Modelica Association (2017). *Modelica - A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.4. Tech. Rep.* Modelica Association. URL: https://modelica.org/documents/ModelicaSpec34.pdf.

Modelica Association (2020). *Modelica Standard Library. Version 4.0.0*. Modelica Association. URL: https://github.com/modelica/ModelicaStandardLibrary (visited on 2025-03-26).

Moreau, Jean Jacques (1988). "Unilateral Contact and Dry Friction in Finite Freedom Dynamics". In: *Nonsmooth Mechanics and Applications*. Ed. by Jean Jacques Moreau and P. D. Panagiōtopulos. Vienna: Springer, pp. 1–82. DOI: 10.1007/978-3-7091-2624-0_1.

Müller, Matthias, Miles Macklin, Nuttapong Chentanez, Stefan Jeschke, and Tae-Yong Kim (2020). "Detailed Rigid Body Simulation with Extended Position Based Dynamics". In: *Computer Graphics Forum* 39.8, pp. 101–112. DOI: 10.1111/cgf.14105.

Narang, Yashraj, Kier Storey, Iretiayo Akinola, Miles Macklin, Philipp Reist, et al. (2022). "Factory: Fast Contact for Robotic Assembly". In: *Robotics: Science and Systems (RSS 2022)* (New York, NY, USA).

Neumayr, Andrea (2025). "Modeling and Simulation of Physical Systems with Variable Structure". PhD thesis. Technische Universität München. URL: https://mediatum.ub.tum.de/?id=1767266.

Neumayr, Andrea and Martin Otter (2017). "Collision Handling with Variable-Step Integrators". In: *Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT'17)*. New York: ACM. DOI: 10.1145/3158191.3158193.

Neumayr, Andrea and Martin Otter (2023). "Modelling and Simulation of Physical Systems with Dynamically Changing Degrees of Freedom". In: *Electronics* 12.3. DOI: 10.3390/electronics12030500.

Nvidia (2024). *NVIDIA PhysX. NVIDIA PhysX is a scalable, multi-physics SDK for simulating and modeling physics in Robotics, Autonomous Vehicles, and VFX workflows.* URL: https://developer.nvidia.com/physx-sdk (visited on 2025-03-26).

Oestersötebier, Felix, Peng Wang, and Ansgar Trächtler (2014). "A Modelica Contact Library for Idealized Simulation of Independently Defined Contact Surfaces". In: *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 929–937. DOI: 10.3384/ecp14096929.

Otter, Martin and Jonathan Brembeck (2022). *Simulation mechatronischer Systeme. Vorlesung, Technische Universität München.*

Otter, Martin, Hilding Elmqvist, and José Díaz López (2005). "Collision Handling for the Modelica Multibody Library". In: *Proceedings of the 4th International Modelica Conference, March 7-8, 2005, Hamburg, Germany*. URL: https://elib.dlr.de/12299/.

Otter, Martin, Hilding Elmqvist, and Sven Erik Mattsson (2003). "The New Modelica Multi-Body Library". In: *Proceedings of the 3rd International Modelica Conference, November 3-4, 2003, Linköping, Sweden*. URL: https://modelica.org/events/Conference2003/papers/h37_Otter_multibody.pdf.

Petzold, Linda R. (1982). "A Description of DASSL: A Differential/Algebraic System Solver". In: *SAND-82-8637 CONF-820810-21* (Livermore, CA, USA). Sandia National Labs.

Pfeiffer, Andreas (2012). "Optimization Library for Interactive Multi-Criteria Optimization Tasks". In: *Proceedings of the 9th International Modelica Conference, September 3-5, 2012, Munich, Germany*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 669–680. DOI: 10.3384/ecp12076669.

Pfeiffer, Friedrich (2008). "On non-smooth dynamics". In: *Meccanica* 43.5, pp. 533–554. DOI: 10.1007/s11012-008-9139-1.

Pfeiffer, Friedrich and Christoph Glocker (2004). *Multibody dynamics with unilateral contacts*. Wiley series in nonlinear science. Weinheim: Wiley-VCH. ISBN: 9780471155652.

Pozzi, Maria, Gabriele Maria Achilli, Maria Cristina Valigi, and Monica Malvezzi (2022). "Modeling and Simulation of Robotic Grasping in Simulink Through Simscape Multibody". In: *Frontiers in Robotics and AI* 9. DOI: 10.3389/frobt.2022.873558.

Puntel-Schmidt, Philipp and Alexander Fay (2015). "Levels of Detail and Appropriate Model Types for Virtual Commissioning in Manufacturing Engineering". In: *IFAC-PapersOnLine* 48.1, pp. 922–927. DOI: 10.1016/j.ifacol.2015.05.027.

Reiner, Matthias J. (2011). "Modellierung und Steuerung von strukturelastischen Robotern". PhD thesis. Technische Universität München. URL: https://mediatum.ub.tum.de/?id=967161.

Reinhart, Gunther and Georg Wünsch (2007). "Economic application of virtual commissioning to mechatronic production systems". In: *Production Engineering* 1.4, pp. 371–379. DOI: 10.1007/s11740-007-0066-0.

Reiser, Robert (2021). "Object Manipulation and Assembly in Modelica". In: *Proceedings of the 14th Modelica Conference 2021, Linköping, Sweden, September 20-24, 2021*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 433–441. DOI: 10.3384/ecp21181433.

Reiser, Robert (2025). "A Drilling Force Model for Use in Multibody System Simulation Environments". In: *SNE Simulation Notes Europe* 35.4, pp. 179–186. DOI: 10.11128/sne.35.tn.10752.

Reiser, Robert and Matthias J. Reiner (2023). "Modeling and simulation of dynamically constrained objects for limited structurally variable systems in Modelica". In: *Proceedings of the 15th International Modelica Conference, Aachen, Germany, October 09-11, 2023*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 151–158. DOI: 10.3384/ecp204151.

Reiser, Robert, Bernhard Thiele, Tobias Bellmann, Philip Koch, and Christoph Walter (2022). "Real-time simulation and virtual commissioning of a modular robot system with OPC UA". In: *ISR Europe 2022. 54th International Symposium on Robotics*. VDE ITG. Munich: VDE Verlag. ISBN: 9783800758913.

Rohmer, Eric, Surya P. N. Singh, and Marc Freese (2013). "V-REP: a Versatile and Scalable Robot Simulation Framework". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 1321–1326. DOI: 10.1109/iros.2013.6696520.

Rulka, Wolfgang (1990). "SIMPACK - A Computer Program for Simulation of Large-motion Multibody Systems". In: *Multibody Systems Handbook*. Ed. by Werner Schiehlen. Berlin, Heidelberg: Springer, pp. 265–284. DOI: 10.1007/978-3-642-50995-7_16.

Ryan, R. R. (1990). "ADAMS - Multibody System Analysis Software". In: *Multibody Systems Handbook*. Ed. by Werner Schiehlen. Berlin, Heidelberg: Springer, pp. 361–402. DOI: 10.1007/978-3-642-50995-7_21.

Sagardia, Mikel, Theodoros Stouraitis, and Joao Lopes e Silva (2014). "A New Fast and Robust Collision Detection and Force Computation Algorithm Applied to the Physics Engine Bullet: Method, Integration, and Evaluation". In: *Conference and Exhibition of the European Association of Virtual and Augmented Reality* (Bremen). University of Bremen.

Scherf, Helmut E. (2011). *Modellbildung und Simulation dynamischer Systeme. Eine Sammlung von Simulink-Beispielen*. 4th ed. München: Oldenbourg. ISBN: 9783486711349.

Smith, Russ (2024). *Open Dynamics Engine*. URL: https://www.ode.org/ (visited on 2025-03-26).

Snethen, Gary (2008). "XenoCollide: Complex Collision Made Simple". In: *Game programming gems 7*. Ed. by Scott Jacobs. Boston: Charles River Media, pp. 165–178. ISBN: 9781584505273.

Striffler, Nikolai and Tobias Voigt (2023). "Concepts and trends of virtual commissioning - A comprehensive review". In: *Journal of Manufacturing Systems* 71, pp. 664–680. DOI: 10.1016/j.jmsy.2023.10.013.

Stüber, Moritz (2017). "Simulating a Variable-structure Model of an Electric Vehicle for Battery Life Estimation Using Modelica/Dymola and Python". In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 291–298. DOI: 10.3384/ecp17132291.

Süß, Sebastian, Anton Strahilov, and Christian Diedrich (2015). "Behaviour Simulation for Virtual Commissioning using Co-Simulation". In: *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, pp. 1–8. DOI: 10.1109/etfa.2015.7301427.

Tasora, Alessandro, Radu Serban, Hammad Mazhar, Arman Pazouki, Daniel Melanz, Jonathan Fleischmann, Michael Taylor, Hiroyuki Sugiyama, and Dan Negrut (2016). "Chrono: An Open Source Multi-physics Dynamics Engine". In: *High Performance Computing in Science and Engineering. Second International Conference, HPCSE 2015, Soláň, Czech Republic*. Cham: Springer, pp. 19–49. DOI: 10.1007/978-3-319-40361-8_2.

Tedrake, Russ and the Drake Development Team (2019). *Drake: Model-based design and verification for robotics*. URL: https://drake.mit.edu (visited on 2025-03-26).

Thiele, Bernhard, Thomas Beutlich, Volker Waurich, Martin Sjölund, and Tobias Bellmann (2017). "Towards a Standard-Conform, Platform-Generic and Feature-Rich Modelica Device Drivers Library". In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 713–723. DOI: 10.3384/ecp17132713.

Thulesen, Thomas N. (2016). "Dynamic Simulation of Manipulation & Assembly Actions". PhD thesis. University of Southern Denmark.

Thulesen, Thomas N. and Henrik G. Petersen (2016). "RobWorkPhysicsEngine: A new Dynamic Simulation Engine for Manipulation Actions". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)* (Stockholm, Sweden). IEEE, pp. 2060–2067. DOI: 10.1109/icra.2016.7487354.

Tinnerholm, John, Adrian Pop, and Martin Sjölund (2022). "A Modular, Extensible, and Modelica-Standard-Compliant OpenModelica Compiler Framework in Julia Supporting Structural Variability". In: *Electronics* 11.11. DOI: 10.3390/electronics11111772.

Todorov, Emanuel (2014). "Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo". In: *2014 IEEE International Conference on Robotics and Automation (ICRA 2014)* (Hong Kong, China). IEEE, pp. 6054–6061. DOI: 10.1109/icra.2014.6907751.

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). "MuJoCo: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 5026–5033. DOI: 10.1109/iros.2012.6386109.

Toniolo, Matthew, Paul Tartabini, Bamdi Pamadi, and Nathaniel Hotchko (2008). "Constraint Force Equation Methodology for Modeling Multi-Body Stage Separation Dynamics". In: *46th AIAA Aerospace Sciences Meeting and Exhibit* (Reno, Nevada). DOI: 10.2514/6.2008-219.

van den Bergen, Gino (1997). "Efficient Collision Detection of Complex Deformable Models using AABB Trees". In: *Journal of Graphics Tools* 2.4, pp. 1–13. DOI: 10.1080/10867651.1997.10487480.

van den Bergen, Gino (2004). *Collision Detection in Interactive 3D Environments*. San Francisco: Elsevier and Morgan Kaufmann Publishers. ISBN: 9781558608016.

Wachter, Christian (2023). *MaTiC-M. Beschreibung der Use-Cases. Use Case: Elektromotor (Traktionsmaschine PKW). Internal report.*

Wünsch, Georg (2008). *Methoden für die virtuelle Inbetriebnahme automatisierter Produktionssysteme*. Dissertation, Technische Universität München, 2007. Forschungsberichte IWB. München: Utz. ISBN: 9783831607952.

Zechmair, Michael and Yannick Morel (2022). "Penalty-based Numerical Representation of Rigid Body Interactions with Applications to Simulation of Robotic Grasping". In: *2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*. IEEE, pp. 1–8. DOI: 10.1109/iceccme55909.2022.9988305.

Zimmer, Dirk (2010). "Equation-Based Modeling of Variable-Structure Systems". PhD thesis. ETH Zürich. DOI: 10.3929/ethz-a-006053740.

Zimmer, Dirk, Niels Weber, and Michael Meißner (2021). "The DLR ThermoFluidStream Library". In: *Proceedings of the 14th Modelica Conference 2021, Linköping, Sweden, September 20-24, 2021*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 225–234. DOI: 10.3384/ecp21181225.

List of publications

Koch, Philip, Nico Töpfer, Sebastian Albrecht, Bernhard Thiele, and Robert Reiser (2025). "Modular production using hierarchical planning and a generic OPC UA skills concept". In: *2025 IEEE 21st International Conference on Automation Science and Engineering (CASE)*. IEEE, pp. 2347–2354. DOI: 10.1109/case58245.2025.11164145.

Reiser, Robert (2021). "Object Manipulation and Assembly in Modelica". In: *Proceedings of the 14th Modelica Conference 2021, Linköping, Sweden, September 20-24, 2021*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 433–441. DOI: 10.3384/ecp21181433.

Reiser, Robert (2025). "A Drilling Force Model for Use in Multibody System Simulation Environments". In: *SNE Simulation Notes Europe* 35.4, pp. 179–186. DOI: 10.11128/sne.35.tn.10752.

Reiser, Robert and Matthias J. Reiner (2023). "Modeling and simulation of dynamically constrained objects for limited structurally variable systems in Modelica". In: *Proceedings of the 15th International Modelica Conference, Aachen, Germany, October 09-11, 2023*. Linköping: Modelica Association and Linköping University Electronic Press, pp. 151–158. DOI: 10.3384/ecp204151.

Reiser, Robert, Bernhard Thiele, Tobias Bellmann, Philip Koch, and Christoph Walter (2022). "Real-time simulation and virtual commissioning of a modular robot system with OPC UA". In: *ISR Europe 2022. 54th International Symposium on Robotics*. VDE ITG. Munich: VDE Verlag. ISBN: 9783800758913.

A Appendix

A.1 Fundamentals

In this section, the fundamentals for this work are introduced. This includes known notations from the literature and custom definitions used for this work.

A.1.1 Graph theory

A graph is defined as the following pair (E. A. Bender and Williamson 2010, p. 148), (Cormen et al. 2022, p. 549):

$$G = (V,\ E) \tag{A.1}$$

with V being a finite set of vertices and E being a set of edges. In the scope of this work, graphs are always simple graphs (multiple edges between two vertices are not allowed).
A path in a graph G is defined as a sequence of edges in E whereby the related set of vertices in V is distinct (E. A. Bender and Williamson 2010, p. 162):

$$path = (e_1,\ e_2,\ ...,\ e_{n-1}) \tag{A.2}$$

The corresponding sequence of vertices is referred to as Path Vertices Sequence (PVS):

$$PVS = (v_1,\ v_2,\ ...,\ v_n) \tag{A.3}$$

A.1.2 Forces

The weight force of a component A with the mass m_A is given by:

$$f_{g,\,A} = m_A \cdot g \tag{A.4}$$

The translational spring force of a component A is calculated by:

$$f_{s,\,A} = k \cdot \Delta s_A \tag{A.5}$$

with Δs_A being the difference in length in relation to the length of the not stretched spring and k being the spring stiffness. In general, the stiffness of a spring between the components A and B is referred to as k_{AB}.
The translational damping force of a component A is given by:

$$f_{d,\,A} = d \cdot \dot{s}_A \tag{A.6}$$

with the velocity $\dot{s}_A$ and the damping factor d.

A.1.3 Damped free oscillations

A damped spring-mass oscillator moving in the x-direction has the following equation of motion (Gross et al. 2015, p. 242):

$$m \cdot \ddot{x} + d \cdot \dot{x} + k \cdot x = 0 \tag{A.7}$$

It can be transformed into the following form (Gross et al. 2015, p. 242):

$$\ddot{x} + 2 \cdot \delta \cdot \dot{x} + \omega_0^2 \cdot x = 0 \tag{A.8}$$

$$\Rightarrow \ddot{x} + 2 \cdot D \cdot \omega_0 \cdot \dot{x} + \omega_0^2 \cdot x = 0 \tag{A.9}$$

where ω_0 is the eigenfrequency of the undamped oscillation, δ the decay coefficient, and D the damping ratio (Lehr's damping ratio) (Gross et al. 2015, p. 242-243):

$$\omega_0 = \sqrt{\frac{k}{m}} \tag{A.10}$$

$$2 \cdot \delta = \frac{d}{m} \tag{A.11}$$

$$D = \frac{\delta}{\omega_0} = \frac{d}{2 \cdot \sqrt{m \cdot k}} \tag{A.12}$$

The oscillator is strongly damped for $D > 1$ (no oscillation) and weakly damped for $D < 1$ (with oscillation). Critical damping is defined by $D = 1$. In this case, the resting position is reached fastest when no oscillation is allowed (Gross et al. 2015, p. 243-244).

A.1.4 Notation

Vectors are represented with lowercase bold letters:

$$\boldsymbol{v} = \begin{pmatrix} a \\ b \\ c \end{pmatrix} \tag{A.13}$$

And matrices with uppercase bold letters:

$$\boldsymbol{M} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & j \end{bmatrix} \tag{A.14}$$

A possible Path Vertices Sequence from component A to T is represented with:

$$PVS_{A \to T} = (A,\ ...,\ T) \tag{A.15}$$

The sequence starts with component A and ends with component T. There can be n components in between (with $n \in \mathbb{N}_0$).

A.2 Modelica

In this section, the *Modelica* language is introduced. First, an overview of the language is given and then common *Modelica* libraries, including those used in this work, are presented.

A.2.1 Overview of the language

Modelica is an object-oriented modeling language. It offers the ability to create multi-domain models of complex systems such as cyber-physical systems based on component models. Therefore, *Modelica* can be classified as a system simulation environment. The modeling is done in an acausal manner (declarative language). In *Modelica*, the physical behavior of a component is primarily described by equations. Full access to these equations provides a high degree of flexibility. The free language is developed by the non-profit organization *Modelica Association* (Modelica Association 2017).
The *Modelica* environment offers a large number of open and commercial libraries that can be used to create models. Common libraries are presented in Section A.2.2. In contrast to tools such as *MATLAB/Simulink*, in *Modelica* the tools are independent of the language. There are commercial tools such as *Dymola* (Dassault Systèmes 2025) and *SimulationX* (ESI Group 2024). In addition, there is the open source software *OpenModelica* (Fritzson et al. 2020).
Models in *Modelica* can be created by combining components from component libraries using the graphical editor. Additionally, equations and algorithms can be defined in the text layer of the tool. To simulate a *Modelica* model, the tool creates an equation system containing all the equations from the components and connections. This equation system is converted into the state form. Then, C code is compiled and executed. (Otter and Brembeck 2022)
Depending on the tool, multiple solvers can be used to simulate a *Modelica* model. There are variable-step solvers such as *DASSL* (Petzold 1982) and *Cvode* (Hindmarsh et al. 2005). Both solvers are implicit, multi-step, and variable order. Another variable-step solver is *Esdirk* (Kennedy and Carpenter 2019). This solver is implicit, single step, and fixed order. Furthermore, there are fixed-step solvers such as *Euler* and *Runge-Kutta* (Otter and Brembeck 2022). Both of them are explicit solvers. For real-time simulations there are additional features such as inline integration (Elmqvist et al. 1995).
Modelica models can be exported using the Functional Mockup Interface (FMI)[1] standard (Blochwitz et al. 2011). In order to do this, the models must contain inputs and outputs. It allows the exchange of simulation models across multiple tools and is supported by most *Modelica* tools and other tools such as *MATLAB/Simulink*. This allows, for example, to create detailed models with the dynamical behavior of a system in *Modelica* and develop the control in *MATLAB/Simulink* (Software in the Loop (SiL) simulation). The FMI standard is also developed by the non-profit organization *Modelica Association*. FMI 2.0 (Blochwitz et al. 2012) and FMI 3.0 (Junghanns et al. 2021) are the latest developments.

[1] `https://fmi-standard.org/`

A.2.2 Modelica libraries

The following *Modelica* libraries are used in this work:

- *Modelica Standard Library (MSL)*[2] (Modelica Association 2020)
  A free and open source library containing components from multiple domains, including thermal, electrical, and mechanical multibody elements.
- *DLR Robots library* (Bellmann et al. 2020)
  A library for the modeling and simulation of robot systems based on replaceable packages. The library is used internally at the DLR.
- *DLR Visualization 2 Library* (Kümper et al. 2021)
  A commercial library for the real-time visualization of simulation models. The library provides components that can be integrated into *Modelica* models.
- *Modelica Lua Library* (Buse and Bellmann 2021)
  A library that integrates a *Lua* interpreter into *Modelica* models. This interpreter is used in the robot controller. The library is used internally at the DLR.
- *DLR Optimization Library* (A. Pfeiffer 2012)
  A commercial library for multi-criteria optimization tasks. This library is used for the development of controllers for the robot dynamics models in this work.

Other common *Modelica* libraries are:

- *Modelica_DeviceDrivers library*[3] (Thiele et al. 2017)
  An open source library that provides hardware drivers for *Modelica* models. It can be used for tasks such as Hardware in the Loop (HiL) simulations.
- *DLR Thermofluid Stream Library*[4] (Zimmer et al. 2021)
  An open source library for the robust modeling and simulation of thermofluid architectures. It can be used for use cases such as the thermal management of cars or plants.
- *DLR FlexibleBodies library* (Heckmann et al. 2006)
  A commercial library for the modeling of flexible bodies and large beam motions. It is based on object-oriented multibody components.

An overview of most libraries can be found on the *Modelica* website[5].

[2] https://github.com/modelica/ModelicaStandardLibrary
[3] https://github.com/modelica-3rdparty/Modelica_DeviceDrivers
[4] https://github.com/DLR-SR/ThermofluidStream
[5] https://modelica.org/libraries/