Modeling and simulation of dynamically constrained objects for limited structurally variable systems in Modelica

Robert Reiser[1]   Matthias J. Reiner[1]

[1]Institute of System Dynamics and Control, German Aerospace Center (DLR), Germany,
`{firstname.lastname}@dlr.de`

Abstract

This work introduces a new solution for the modeling and simulation of dynamically constrained objects for limited structurally variable systems purely in Modelica. A combination of a collision detection algorithm, the limitation of collisions, and a method to constrain objects based on forces leads to a constraint network in Modelica. It allows a stable and accurate simulation of applications such as robot tool changers in a flexible way without the need for predefined connections in the model.

Keywords: collision detection, structural variability, constraint force, network, tool change, robotics, Modelica

1 Introduction

Structurally variable systems often occur in different fields of technical problems. One prominent example is robot tool changers. The production industry has an ever-increasing demand for flexibility because manufacturing is shifting from standardized products with high quantities to individual goods. Tool changers are a common method to increase the flexibility of an assembly cell.

Simulation is important for the development and testing of robot cells for example in virtual commissioning (Wünsch 2008). There have been many works about the simulation of robots (Paryanto et al. 2014; Bellmann, Seefried, and Thiele 2020; Reiser et al. 2022) and manipulation (Reiser 2021) in Modelica. Existing tools such as RoboDK, CoppeliaSim, and ANSYS (Li et al. 2016) can be used to simulate tool changers but they do not offer the high degree of flexibility of the Modelica language.

In the Modelica environment (Modelica Association 2017) it is especially challenging to simulate applications such as tool changers since structural variability is not possible, limited to special cases (Stüber 2017) or requires additional effort (Tinnerholm, Pop, and Sjölund 2022). To our knowledge, there has been no work in the area of simulating a tool changer based on Modelica models.

There are also examples in the field of aeronautics, such as the structural changes during runtime in the area of stage separation (Acquatella and Reiner 2014) or for in orbit construction of orbital platforms (Reiner 2022).

This work builds on the previous work of (Acquatella and Reiner 2014) and (Reiner 2022) to make it usable in a wider range of applications, and shows how robot tool changers can be modeled using the proposed technique. We present a new solution for the modeling and simulation of structurally variable systems in Modelica. It combines:

- A collision detection algorithm in Modelica

- A method to limit the number of possible collisions (see Figure 1 and section 3.2 for details)

- The Constraint Force Equation method combined with the Baumgarte stabilization technique

The next section introduces the state of the art. Section 3 presents the new solution for the modeling of structurally variable systems in Modelica. The following section contains the implementation. In Section 5, a tool change process and the transition of a spring-borne object are simulated. In conclusion, the approach is discussed and future developments are considered.
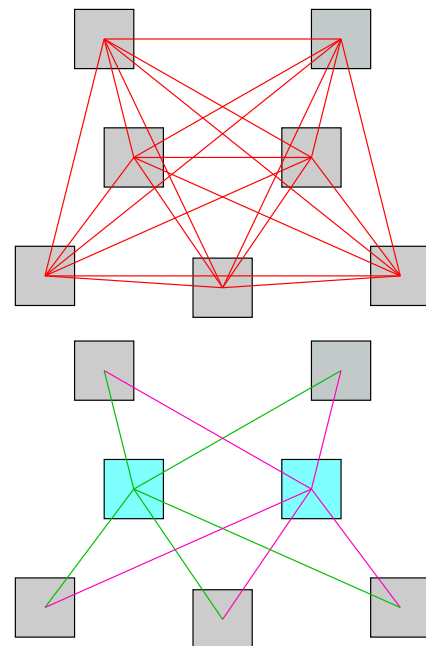

Figure 1. An example showing the limitation of possible collisions. If all collisions are allowed, there are 21 possible collision pairs (top). A restriction divides the objects into two groups and only collisions between blue and grey objects are allowed. Now there are only 10 possible collision pairs (bottom).

2 State of the art

In this section, related existing work is analyzed. This includes an overview of collision detection and the introduction of the Constraint Force Equation (CFE) method and the Baumgarte stabilization technique.

2.1 Collision detection

Collision detection is a complex task. There are special libraries such as the **libccd** (Fiser 2018) to detect collisions between convex shapes. The most common algorithms are the Gilbert-Johnson-Keerthi distance algorithm (GJK) (Gilbert, Johnson, and Keerthi 1988) and the Minkowski Portal Refinement algorithm (MPR) (Snethen 2008).

There are several works about collision detection in Modelica: (Otter, Elmqvist, and López 2005), (Hofmann et al. 2014), and (Elmqvist et al. 2015).

To the knowledge of the authors, the approaches above have in common that they combine Modelica with an external library for the collision detection task. The usage of external function calls can result in delays (i.e. values might be one time step behind in the simulation), additional model complexity, and possible incompatibility when used on different computing platforms.

In the works of (Oestersötebier, Wang, and Trächtler 2014) and (Bortoff 2020), the collision detection is native in Modelica. However, predefined contacts are needed.

2.2 Constraint Force Equation method and Baumgarte stabilization

The Constraint Force Equation (CFE) method was developed at NASA (Toniolo et al. 2008). The aim is to constrain two bodies by applying joint forces to each body. (Acquatella and Reiner 2014) used this method for the modeling and simulation of stage separation dynamics in Modelica and (Reiner 2022) for robot based in orbit construction of orbital platforms. Their solution is however limited because the contact pairs are predefined or use special cases and cannot be changed during runtime. Therefore an application such as a tool changer cannot be modeled easily. A tool is usually connected to more than one robot and a tool holder and the connections change during a process. A more general method is needed.

The Baumgarte stabilization (Baumgarte 1972) is used to stabilize the constraint force equation. The basic formula for the constrained force calculation with Baumgarte damping can be seen in the following equation:

$$\ddot{\xi} + 2\eta\dot{\xi} + \eta^2\xi = 0 \tag{1}$$

ξ represents the (generalized) difference in position and orientation between the two objects and $\eta > 0$ the damping factor, resulting in an asymptotically stable ODE. Note that the constraint is defined as a kinematic condition. Modelica can automatically calculate the resulting forces and torques when the equation is correctly used together with mechanical bodies with mass and/or inertia (see implementation details in later sections).

Since the constraint $\ddot{\xi} = 0$ is defined on the relative acceleration between the to be constrained objects, small numerical errors can lead to drift in the relative generalized velocity $\dot{\xi}$ and position ξ between the objects. Using the additional damping terms in equation 1 for the relative velocities and positions between the objects can reduce this drift substantially. See section 4.6 for more details on the implementation used here.

3 Modeling structurally variable systems in Modelica

The solution for modeling structurally variable systems in Modelica is presented in the following. The section starts with the general idea behind this approach. Furthermore, the method for limiting the number of possible collisions is introduced and a Modelica native collision detection algorithm is presented.

3.1 Idea

The idea is to build a constraint network within Modelica. By forgoing external libraries, the approach is stable and accurate. In general, the CFE method implemented in (Acquatella and Reiner 2014) is combined with a collision detection algorithm. Thus, predefined contact pairs are no longer required. In addition, the number of possible collisions is limited to achieve a higher performance.

3.2 Limitation of possible collisions

Building a general collision detection library in Modelica is challenging. However, it is possible to restrict the scope.

In collision detection, the number of possible collision pairs x depends on the number of objects in the scene n and can be determined by:

$$x = \frac{n!}{k! \cdot (n-k)!} = \frac{n!}{2 \cdot (n-2)!} \tag{2}$$

This is based on the equation to calculate the number of combinations of k from n elements. An example is shown in Figure 1 (top). Seven objects in a scene have 21 possible collision pairs. With an increasing number of objects, the number of pairs increases significantly. To avoid this, the number of possible collisions is limited by dividing the objects in a scene into two groups and allowing only collisions between objects of one group and another.

In the example, one group contains two and the other group five objects. Therefore the number of pairs decreases to ten objects (see Figure 1 (bottom)).

Now the number of possible combinations is:

$$x = n_1 \cdot n_2 \tag{3}$$

where n_1 is the number of objects of the first and n_2 the number of objects of the second group.

The number of collision checks performed during simulation runtime is directly related to the number of possible collision pairs in a scene. In other words, limiting the number of possible collision pairs allows a fast collision detection algorithm in native Modelica code.

3.3 Collision detection in Modelica

The reasons for building a native Modelica collision detection algorithm are a high stability and compatibility of the resulting simulations and a high accuracy of the results.

Implementing a collision detection algorithm (e.g. GJK or MPR) in Modelica in general is not feasible since operations such as the handling of complex 3D models are hardly manageable without external code. Furthermore, such an algorithm would have a weak performance because it is not possible to use all the optimization techniques usually applied in collision detection libraries.

Hence, for this work, the collision detection algorithm is highly restricted. Only the following contact combinations are allowed:

- Sphere and sphere

- Sphere and rectangle surface (with length and width)

This reduces the complexity significantly and leads to a fast calculation of collision checks.

The collision detection between **two spheres** (located at position $p_{Sphere1}$ and $p_{Sphere2}$) is straight forward: Each object has a radius and the collision check between two objects is based on the Euclidean distance. The Euclidean distance $d_{Euclidean}$ is calculated by:

$$d_{Euclidean} = \|p_{Sphere1} - p_{Sphere2}\| \tag{4}$$

Using the sum of the radius $r_{Sphere1}$ and $r_{Sphere2}$ of both objects a collision occurs when the following inequality is fulfilled:

$$d_{Euclidean} < r_{Sphere1} + r_{Sphere2} \tag{5}$$

The collision check between **sphere and rectangle surface** can also easily be calculated. The sphere (located at position p_{Sphere}) is defined by its radius r_{Sphere} and the rectangle (located at position $p_{Rectangle}$) by its length $l_{Rectangle}$ and width $w_{Rectangle}$. The distance vector between the sphere origin and the rectangle origin d_{SP} in the orientation of the rectangle $T_{Rectangle}$ can be calculated by:

$$d_{SP} = \begin{pmatrix} d_{SP1} \\ d_{SP2} \\ d_{SP3} \end{pmatrix} = T_{Rectangle} \cdot (p_{Sphere} - p_{Rectangle}) \tag{6}$$

For the implementation, it is assumed that the rectangle normal is the local z-axis of the object. Now a simple distance inequality can be checked to determine the collision. If all of the following inequalities are fulfilled, a collision occurs between the sphere and the rectangle surface:

$$d_{SP1} < l_{Rectangle} \tag{7}$$
$$d_{SP2} < w_{Rectangle} \tag{8}$$
$$d_{SP3} < r_{Sphere} \tag{9}$$

Since these inequalities are easy to solve in Modelica for a limited number of objects, a native implementation is possible with good computational performance, while still maintaining the flexibility and power of the Modelica language.

4 Implementation

This section shows the implementation of the solution presented in section 3. Figure 2 shows an overview of the implementation in the library browser. The structure consists of three objects (see Figure 3 for details):

- **CollisionCollector** (outer object to store information of the CollisionObjects and ConstrainedObjects)

- **CollisionObject** (lightweight object whose position and orientation are stored in the CollisionCollector)

- **ConstrainedObject** (contains the collision detection algorithm and calculates the constraint forces)

4.1 Objects

This section contains descriptions for the objects of the implemented solution (CollisionCollector, CollisionObject, ConstrainedObject). An overview of all objects and their interaction is illustrated in Figure 3.

4.1.1 CollisionCollector

The **CollisionCollector** is an outer object to store all information of the CollisionObjects. This includes the position, velocity, acceleration, orientation, angular velocity, and angular acceleration of each object. Further information are the collision type (see section 4.5), the shape type (sphere/rectangle) with the related radius, length and width, and the closed indicator.

In addition, the CollisionCollector stores information of the ConstrainedObject. This includes the force and torque calculated in the ConstrainedObject and the ID of the CollisionObject in contact with the ConstrainedObject.
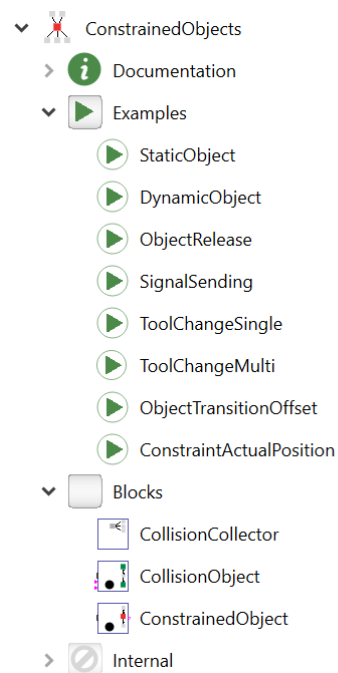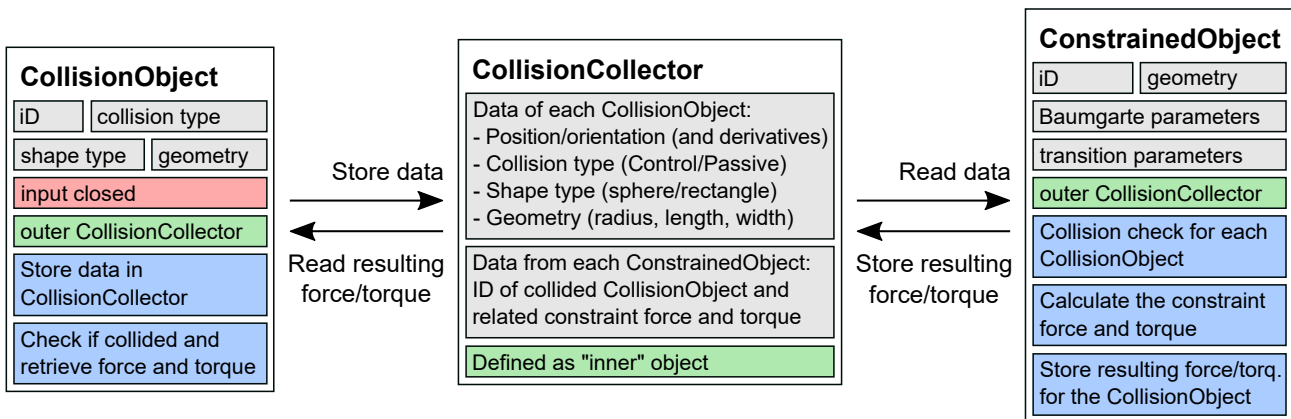

Figure 2. Overview of the library structure.

Figure 3. Overview of the implemented objects with parameters (grey), inputs (red), methods (blue), and the inner/outer dependency (green). The arrows show the interaction based on the data flow. The CollisionObject stores its data in the CollisionCollector. The ConstrainedObject reads this data and performs a collision check for each CollisionObject. If a collision occurs, the ConstrainedObject calculates the constraint force and torque. The resulting force and torque for the related CollisionObject are returned.

4.1.2 CollisionObject

The **CollisionObject** is lightweight and has mainly the aim to store its position and orientation and their derivatives to the CollisionCollector (used as outer object). Parameters are the ID, the collision type (see section 4.5), and the shape type (sphere/rectangle) with the related radius, length, and width. The closed indicator is an input.

4.1.3 ConstrainedObject

The **ConstrainedObject** contains the collision detection algorithm and equations to calculate the constraint forces and torques. It also uses the CollisionCollector as outer object. Parameters are the ID, the radius (the ConstrainedObject is always a sphere), the damping factor for the Baumgarte stabilization (η), the duration for the smooth transition phase, and settings to enable the smooth transition phase and the offset.

4.2 Building the constraint network

The constraint network is built as follows:

- The CollisionObjects store their information in the CollisionCollector. This includes the position and orientation (with velocity and acceleration), radius (or rectangle length and width), collision type, collision shape, and closed indicator.

- Each ConstrainedObject runs a collision check to all CollisionObjects by calculating the corresponding equations from section 3 (it gets the relevant data via the CollisionCollector).

- If a collision occurs, the constraint force and torque are calculated (see section 4.6) to constrain the ConstrainedObject to the related CollisionObject.

- The resulting force and torque for the related CollisionObject is then returned (see section 4.6).

4.3 Boundary conditions

To achieve the procedure in section 4.2, some boundary conditions are necessary:

Only collisions between CollisionObjects and ConstrainedObjects are possible. Two CollisionObjects can't collide. The same applies for two ConstrainedObjects. An example is Figure 1 (bottom), where the blue objects as can be seen as ConstrainedObjects and the grey ones as CollisionObjects.

The ConstrainedObject can only be constrained to one CollisionObject (in other words it can only have a collision with one CollisionObject). This reduces the number of possible combinations significantly and improves the performance, while still allowing to model many relevant scenarios.

4.4 Manual definition of IDs

The *Modelica Language Specification* (Modelica Association 2017) does not provide the capabilities for unique IDs although it has been proposed in the past (Otter, Elmqvist, and López 2005; Hellerer and Buse 2017). To achieve a standard compliant solution, one necessity is the manual definition of unique IDs for each CollisionObject (1 ... n) and ConstrainedObject (1 ... m). In addition, the count must be set for both objects in the CollisionCollector.

4.5 Opening and closing connections

The boundary conditions restrict the ConstrainedObject to only one collision. This leads to the following question: How can tool changers be simulated having contact with both the robot and the holder? A tool changer is only constrained to one other object, the robot or the holder. But there is a transition phase as well.

Therefore additional capabilities are needed. The **CollisionObjects have a mode**, defined by a type. There are two possible types: *Control* and *Passive*. In *Passive* mode,

nothing changes for the CollisionObject. In *Control* mode, the CollisionObject only collides if an additional input *closed* is true.

In addition, the **CollisionObjects are prioritized** based on their mode. If a ConstrainedObject collides with two CollisionObjects, the one in *Control* mode is higher prioritized, i.e. the collision occurs with this object.

Now the simulation of tool changers is possible. The tool is attached to a ConstrainedObject held by a CollisionObject in *Passive* mode. A CollisionObject in *Control* mode is attached to the robot flange. When the robot has approached the tool, the *closed* indicator of its CollisionObject switches from false to true. This enforces the ConstrainedObject of the tool to switch its constraint from the holder to the robot. An application of this procedure is demonstrated in section 5.1.

4.6 Calculation and return of the constraint force and torque

Constraining accelerations in a complex inner/outer scenario only leads to forces and torques in the ConstrainedObject. There is a high relevance for the constraint forces and torques in the CollisionObject, e.g. to determine the load on the robot.

Hence the force and torque in the ConstrainedObject must be transferred back to the CollisionObject. This is achieved by using the CollisionCollector. Each ConstrainedObject adds the resulting force and torque for the related CollisionObject and the ID of the related CollisionObject to the CollisionCollector (if a collision occurs). The CollisionObject is then able to check for its own ID in the CollisionCollector and if it occurs it retrieves the stored force and torque and applies it to its frame.

In the case of a tool changer, it makes sense to constrain the tool exactly at the position of the robot tool center point (TCP). In reality, there would be some kind of mechanical flange to fixate the tool exactly there. For other scenarios, it is necessary to constrain one object to another at the contact point. An example is the robot based in orbit construction of orbital platforms (see (Reiner 2022)). For such applications, the position and orientation of the ConstrainedObject should be kept and reaction forces computed accordingly. To achieve this within the same Modelica framework, position and orientation offsets have to be computed at the time t_0 when the collision occurs.

The calculation of the resulting constrained force is described in the following (the calculation of the torque is omitted for brevity).

When a collision is detected as described in section 3.3, the position offset p_{offset} is computed as the difference between the position p_c of the counterpart and the ConstrainedObject itself p_s. The start time t_0 is also saved.

$$p_{offset} = \begin{cases} 0 & \text{if no collision} \\ p_c - p_s & \text{if collsion deteced} \end{cases} \quad (10)$$

Since collisions can occur at high speed between objects this can lead to numerical problems when using fixed-step solvers for quick simulations, especially when elastic systems with weak damping are involved. To elevate this problem a slack or transition function $s_{tr}(t)$ can be enabled (optional) to scale up the constraint forces and torques. If not enabled, $s_{tr}(t)$ is simply set to 1 at all times.

It can be parameterized by its duration t_d (should be chosen as small as possible to achieve a stable simulation). The transition function is a smooth function between zero and one and can be differentiated by Modelica automatically (see equation 11).

$$s_{tr}(t) = \begin{cases} 1 & \text{if } t - t_0 >= t_d \\ \left(\sin\left(\frac{(t-t_0)\cdot\pi}{2\cdot t_d}\right)\right)^2 & \text{if } t - t_0 < t_d \end{cases} \quad (11)$$

Equation 12 shows the constrained equation, which leads to the calculation of the constraint force $f_{con,s}$ acting on the ConstrainedObject itself when connected to a mechanical body.

$$s_{tr}(t) \cdot ((\ddot{p}_c - \ddot{p}_s) + 2 \cdot \eta \cdot (\dot{p}_c - \dot{p}_s) + \eta^2 \cdot (p_c - p_s - p_{offset})) = 0 \quad (12)$$

Equation 13 can then be used to calculate the corresponding reaction force $f_{con,c}$ acting on the CollisionObject using the rotation matrices of both objects (T_c and T_s).

$$f_{con,c} = -(T_c \cdot T_s^T) \cdot f_{con,s} \quad (13)$$

When no constraint is active $f_{con,s}$ is set to zero.

5 Applications

In this section, two examples of dynamically constrained objects are shown, namely the simulation of a tool change process and the simulation of the transition of a spring-borne object with offset. We used Dymola 2023x (64-bit) on Windows 10 with a Rkfix4 solver (0.001 s step size) on an Intel® Core™ i7-11700K workstation.

5.1 Simulation of a tool change process

This example demonstrates the capabilities of the developed solution by simulating a tool change process. It consists of two robots, two tools, and four tool holders. The Modelica model is shown in Figure 5. Both tools are attached to ConstrainedObjects. The robot flanges are connected to CollisionObjects in *Control* mode and the tool holders to CollisionObjects in *Passive* mode.

The flange of Robot1 is moved to the holder of Tool1 (green sphere). Then for the CollisionObject attached to the robot the *closed* input switches from false to true. This causes the ConstrainedObject to switch its constraint from the holder to the robot (see section 4.5 for details). Now equipped with Tool1, Robot1 moves to a different holder (red sphere) and releases Tool1 there. Robot2 does the same simultaneously for Tool2 (equipping at the yellow sphere and releasing at the pink sphere). Subsequently,
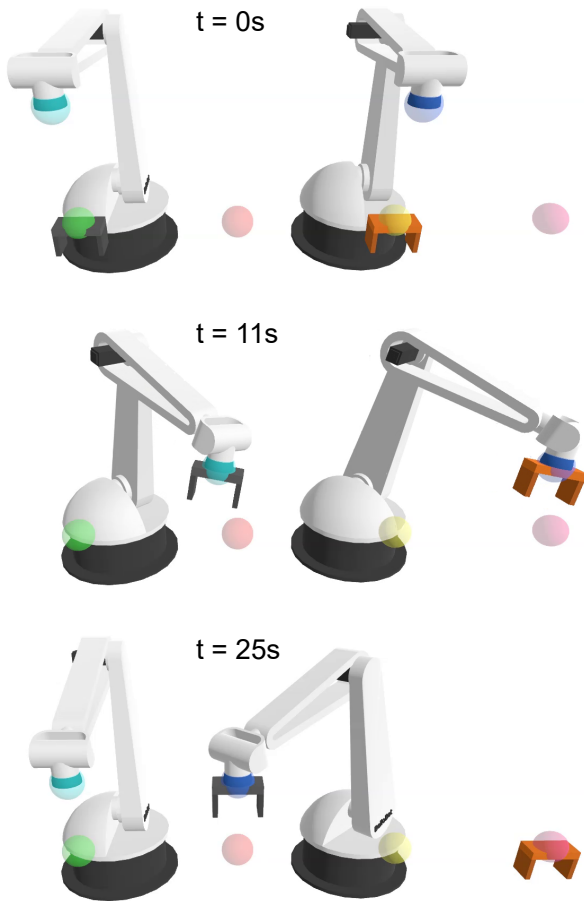
Figure 4. Visualization of the simulation of a tool change process in the DLR Visualization 2 Library. The gray tool is equipped by the left robot and moved to the next holder. The same applies to the right robot and the orange tool ($t = 11s$). In addition, the right robot is equipped with the gray tool ($t = 25s$).

Robot2 is equipped with Tool1 and moved upwards to demonstrate the flexibility of our solution.

The visualization of the final state based on the DLR Visualization 2 Library (Kümper, Hellerer, and Bellmann 2021) is illustrated in Figure 4. Figure 6 shows the results for the constraint forces applied to the ConstrainedObject and to the CollisionObject. The latter represents the resulting forces for the robot when the tool is attached. Simulating the model with 25 s simulation time took 4.5 s.

5.2 Simulation of the transition of a spring-borne object with offset

The second example of dynamically constrained objects is the simulation of a spring-borne object. In the simple scenario, two rigid bodies are connected with a revolute joint. The joint is connected to a spring damper pair. One of the rigid bodies is connected to a ConstrainedObject with enabled transition function (transition duration $0.5s$) and offset calculation. The model also contains three different CollisionObjects (all in *Control* mode, i.e. they can be enabled or disabled by the input *closed*).
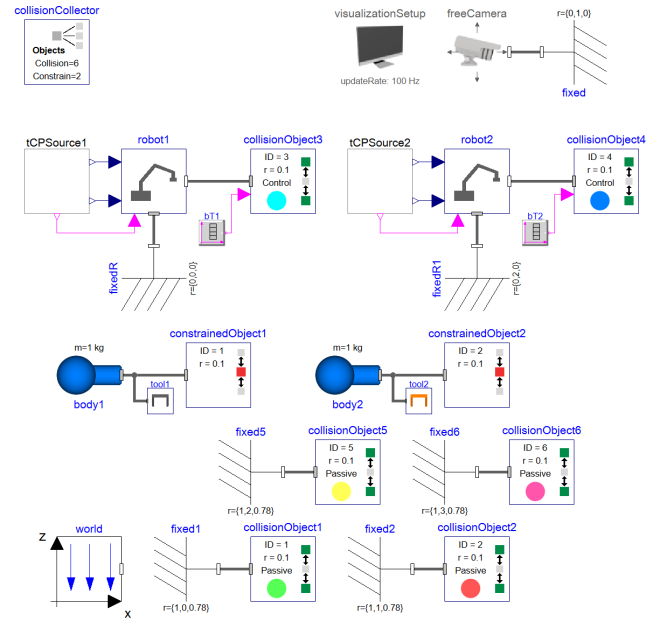
Figure 7 shows an overview of the scenario. At the

Figure 5. Model for the tool change example. There are two robots equipped with CollisionObjects, two tools attached to ConstrainedObjects, and four tool holders with CollisionObjects. No pre-defined connections are necessary, all components can be added to the model by drag-and-drop.
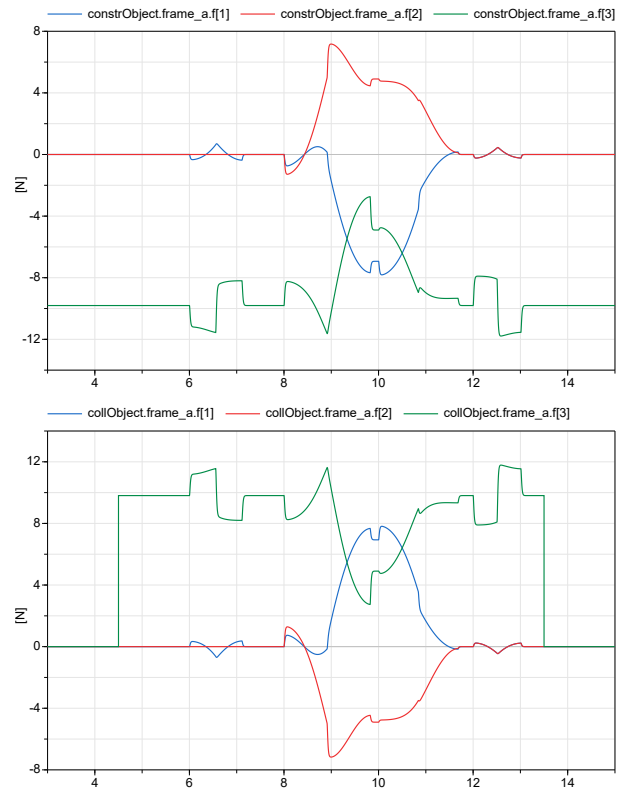

Figure 6. Simulation results for the tool change process. The top shows the forces applied to the ConstrainedObject (connected to the tool) and the bottom shows the forces applied to the CollisionObject (connected to the robot).
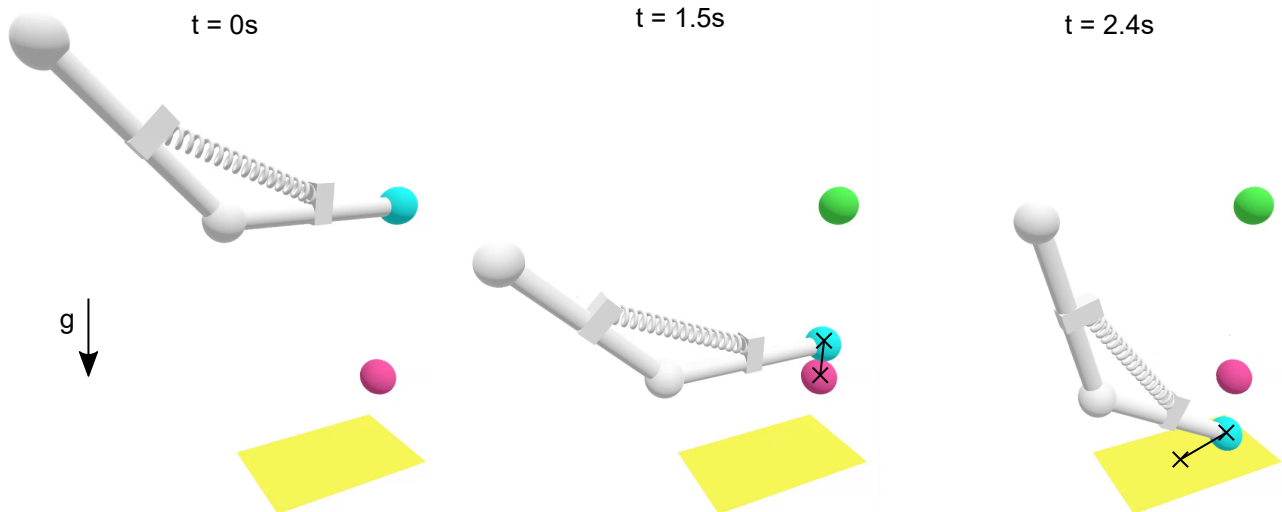
Figure 7. Visualization of the transition of a spring-borne object in the DLR Visualization 2 Library. At first ($t = 0s$), the object is connected to the green sphere. Then it falls and is attached to the pink sphere ($t = 1.5s$). It continues falling and is constrained to the yellow rectangle surface ($t = 2.4s$). The connecting line (black) shows the offset between the origin of both objects.

beginning of the scenario, the ConstrainedObject (blue) is connected to CollisionObject1 (green color). At the time $t = 1s$ the *closed* input of CollisionObject1 is set to false, and the assembly falls down (due to the world gravity in the model) and collides with CollisionObject2 (pink color). At $t = 2.0s$ the input *closed* for this CollisionObject is also set to false, so that the object falls further down

Figure 8. Simulation results for the transition of a spring-borne object with offset. The blue curve shows the constraint force in the z-direction for $\eta = 50$ (damping factor for the Baumgarte stabilization). The red curve shows the result for $\eta = 25$.

until it hits CollisionObject3 (yellow rectangle).

The offset from ConstrainedObject (blue sphere) to CollisionObject2 (pink sphere) and to CollisionObject3 (yellow rectangle) is illustrated in Figure 7. Since the use of offsets is enabled in the ConstrainedObject, the sphere is constrained exactly at the contact point. Otherwise, the sphere would be forced to the center of the rectangle (also with the same orientation as the rectangle). At $t = 2.5s$ the input *closed* for CollisionObject3 is also set to false.

The resulting constraint force in the local z-direction can be seen in Figure 8 for two different values of η (see equation 1). The selection of η is unfortunately not straightforward. In principle, it should be set as low as possible and as high as necessary. A high value of η can lead to a numerically stiff system. This can cause problems with numeric integration, especially when fixed-step solvers are used. However, a too-small value for η can result in large deviations between the objects. As shown in Figure 8, the resulting constrained forces can change significantly for different values of η, especially when flexible elements are involved. This can also lead to different behavior in the overall model. The difference for the beginning of the second force spike (for $t > 2.25s$) in the plot results from the higher constrained force (and torque) which affects the flexible element in the system. As such the parameter η is an engineering (control) parameter and has to be chosen problem specific and very carefully. Simulating the model with 3 s simulation time took 0.08 s. It takes 0.44 s to simulate a model with five spring-borne objects and 1.2 s for one with ten objects.
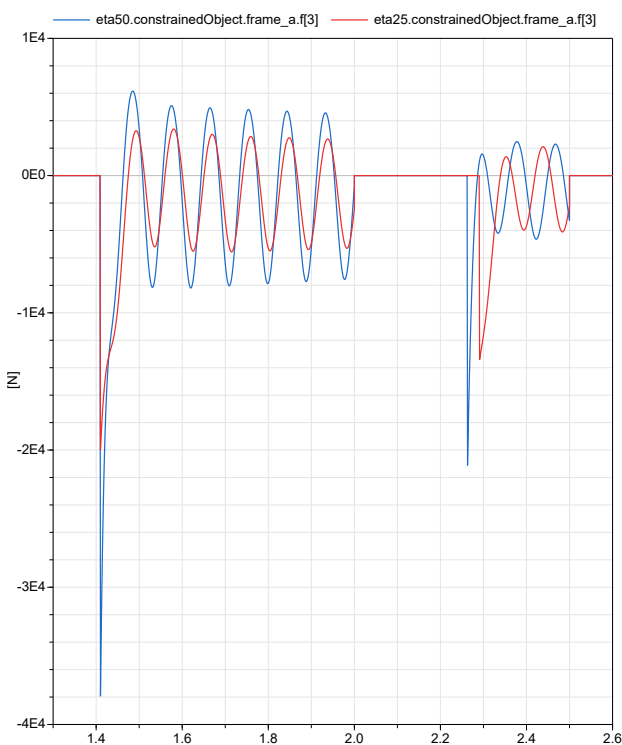
6 Conclusion

In this paper, a new solution for the modeling and simulation of structurally variable systems in Modelica was presented. It combines a collision detection algorithm in

native Modelica code, a method to limit the number of possible collisions, the Constraint Force Equation method, and the Baumgarte stabilization. The result is a constraint network within Modelica. It allows the stable and accurate simulation of structurally variable systems in a flexible way (no pre-defined connections are necessary). The ability of the new solution was demonstrated in two examples: the simulation of a tool change process and the simulation of the transition of a spring-borne object with offset. However, the presented approach has some restrictions and limitations: The user has to manually set unique IDs for the objects since it is not (yet) possible within the Modelica language standard and the scalability of the concept is limited. Possible future developments are the support of more geometries for the collision check and external objects to automatically generate unique IDs.

References

Acquatella, Paul and Matthias J. Reiner (2014). "Modelica Stage Separation Dynamics Modeling for End-to-End Launch Vehicle Trajectory Simulations". In: *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*, pp. 589–598. DOI: 10.3384/ecp14096589.

Baumgarte, J. (1972). "Stabilization of constraints and integrals of motion in dynamical systems". In: *Computer Methods in Applied Mechanics and Engineering* 1.1, pp. 1–16. ISSN: 0045-7825. DOI: 10.1016/0045-7825(72)90018-7.

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*. Ed. by Rui Gao and Yutaka Hirano. Linköping, pp. 153–161. DOI: 10.3384/ecp2020174153.

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*, pp. 54–63. DOI: 10.3384/ecp2016954.

Elmqvist, Hilding et al. (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*, pp. 427–440. DOI: 10.3384/ecp15118427.

Fiser, Daniel (2018). *libccd: Library for collision detection between two convex shapes*. URL: https://github.com/danfis/libccd (visited on 2023-07-05).

Gilbert, E. G., D. W. Johnson, and S. S. 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. ISSN: 08824967. DOI: 10.1109/56.2083.

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*. Ed. by Dirk Zimmer and Bernhard Bachmann. New York, New York, USA, pp. 81–86. DOI: 10.1145/3158191.3158205.

Hofmann, Andreas et al. (2014). "Simulating Collisions within the Modelica MultiBody library". In: *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*, pp. 949–957. DOI: 10.3384/ECP14096949.

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

Li, Na et al. (2016). "The Dynamic Simulation of Robotic Tool Changer Based on ADAMS and ANSYS". In: *2016 International Conference on Cybernetics, Robotics and Control - CRC 2016*. Ed. by Sun Dong, Wei-Hsin Liao, and Sergei Gorlatch, pp. 13–17. DOI: 10.1109/CRC.2016.013.

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

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*, pp. 929–937. DOI: 10.3384/ecp14096929.

Otter, Martin, Hilding Elmqvist, and José Díaz López (2005). "Collision handling for the Modelica multibody library". In: *Proceedings of the 4th International Modelica Conference*.

Paryanto et al. (2014). "Energy Consumption and Dynamic Behavior Analysis of a Six-axis Industrial Robot in an Assembly System". In: *Procedia CIRP* 23, pp. 131–136. ISSN: 22128271. DOI: 10.1016/j.procir.2014.10.091.

Reiner, Matthias J. (2022). "Simulation of the on-orbit construction of structural variable modular spacecraft by robots". In: *Proceedings of the American Modelica Conference 2022*, pp. 38–46. DOI: 10.3384/ECP2118638.

Reiser, Robert (2021). "Object Manipulation and Assembly in Modelica". In: *Proceedings of 14th Modelica Conference 2021, Linköping, Sweden, September 20-24, 2021*. Ed. by Martin Sjölund et al., pp. 433–441. DOI: 10.3384/ecp21181433.

Reiser, Robert et al. (2022). "Real-time simulation and virtual commissioning of a modular robot system with OPC UA". In: *ISR Europe 2022*. Munich: VDE Verlag. ISBN: 978-3-8007-5891-3.

Snethen, Gary (2008). "Xenocollide: Complex collision made simple." In: *Game programming gems 7*. Ed. by Scott Jacobs. Boston, MA: Charles River Media/Course Technology. ISBN: 9781584505273.

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*, pp. 291–298. DOI: 10.3384/ecp17132291.

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, p. 1772. DOI: 10.3390/electronics11111772.

Toniolo, Matthew et al. (2008). "Constraint Force Equation Methodology for Modeling Multi-Body Stage Separation Dynamics". In: *Aerospace Sciences Meetings*. DOI: 10.2514/6.2008-219.

Wünsch, Georg (2008). *Methoden für die virtuelle Inbetriebnahme automatisierter Produktionssysteme*. Vol. 215. Forschungsberichte IWB. München: Utz. ISBN: 978-3-8316-0795-2.