Using Modelica for advanced Multi-Body modelling in 3D graphical robotic simulators

Gianluca Bardaro[1] Luca Bascetta[1] Francesco Casella[1] Matteo Matteucci[1]

[1]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano, Italy,
{luca.bascetta,gianluca.bardaro,francesco.casella,matteo.matteucci}@polimi.it

Abstract

This paper describes a framework to extend the 3D robotic simulation environment Gazebo, and similar ones, with enhanced, tailor-made, multi-body dynamics specified in the Modelica language. The body-to-body interaction models are written in Modelica, but they use the sophisticated collision detection capabilities of the Gazebo engine. This contribution is a first step toward the simulation of complex robotics systems integrating detailed physics modelling and realistic sensors such as lidar and cameras. A proof-of-concept implementation is described in the paper integrating Gazebo collider and the Modelica Multi-Body library, and the results obtained when simulating the interaction of an elastic sphere with a rigid plane are shown.

Keywords: Multi-Body Dynamics, 3D Robotic Simulators, Autonomous Robotics, Autonomous Vehicles.

1 Introduction

The popularity of research on autonomous mobile robots, including autonomous vehicles and mobile manipulators, has been recently increasing due to the huge number of potential applications, ranging from self-driving cars and robots for logistics, to planetary explorations, search and rescue missions, surveillance, humanitarian de-mining, as well as precision agriculture activities such as pruning vines and fruit trees (Paden et al., 2016; Roa et al., 2015; Ko et al., 2015; Chitta et al., 2012).

The design and development of such systems, whose main functionalities are perception, planning, and control, is a multidisciplinary and complex work that has to be supported by virtual prototypes, allowing for a preliminary design and testing of the corresponding algorithms in safe operating conditions. However, due to the huge differences among the three mentioned skills a mobile robot should own, the virtual prototype has to satisfy various requirements. Considering, for example, the development of perception algorithms, the most important characteristics of the virtual prototype are a realistic description, mainly from a geometrical and graphical point of view, of the scene, and the availability of realistic models for the most common commercial sensors, i.e., laser range finders and cameras. On the other hand, testing a control algorithm, e.g., an Advanced Driver Assistance System in a critical situation, requires an accurate physical modelling of the vehicle, including all (and sometimes even only) the phenomena the designer knows to be relevant in the specific application, e.g., cornering stiffness for lateral dynamics control.

Nowadays there are many different, open source and commercial, modelling and simulation environments that are suitable to model vehicles and mobile robots.

A first family is represented by 3D robot simulators, like for example Gazebo[1], V-Rep[2], Webots[3], Morse[4], that are widespread in the robotics community. These simulators allow for an easy development of complex natural/artificial simulation environments, they are already equipped with models of perception devices, and they can be easily integrated with standard robot control middlewares like ROS[5]. For these reasons, they are particularly suitable for the development and testing of planning and perception algorithms, and for the validation of the whole control software before moving to field tests (Bardaro et al., 2014).

The physical simulation implemented in these tools is targeted at real-time execution and ease of virtual prototype set-up; this is obtained by providing the 3D kinematic models for rotational and translational joints to assemble robots and vehicles, and collision detection primitives with simplified translational and rotational friction models. These building blocks are implemented with low level C++ libraries, such as ODE (Drumwright et al., 2010) or MuJoCo (Erez et al., 2015), and the experimenter is expected to use them in a black box fashion with little, if any, way to alter their physical behaviour. Indeed, the differential equations characterizing the physical behaviour of each building block are hidden in the code, often undocumented, and with no direct tool for altering their behaviour. This makes current 3D robotics physical simulation fidelity and accuracy somehow limited, and requires the coding of external plug-ins, e.g., using C++ custom code, every time the phenomenon we are interested in replicating is more complex that the one which can be obtained assembling the available building blocks.

On the other side of the spectrum, a second family of sim-

[1] http://gazebosim.org
[2] http://www.coppeliarobotics.com
[3] http://www.cyberbotics.com
[4] http://www.openrobots.org/wiki/morse
[5] http://www.ros.org

ulators is represented by multi-body and/or multi-physics simulators, like for example Modelica tools [6] such as SimulationX[7], whose aim is to accurately represent the dynamic behaviour of the system, and that are thus particularly suitable for accurate dynamic analysis, control system development, and validation in repeatable and safe operating conditions (D'Amelio et al., 2015). These simulators allow a general mechanism for physical systems modelling, based on an high-level language for the definition of the differential equations describing the relevant aspects of the simulation, but little, if any, support is available for geometrical and graphical simulation of the environment and thus for the simulation of robot sensors such as lidar and cameras.

In this paper we present an approach, inspired by the idea already introduced in (Bardaro et al., 2016), to extend the multi-body modelling in the 3D Gazebo simulator using Modelica and the MultiBody library (Otter et al., 2003). This allows to introduce ad-hoc physical models which are tailored to the specific needs of a particular application in a convenient, declarative, equation-based framework, leveraging on the basic infrastructure already provided by the MultiBody library. On the other hand, we are able to extend the level of simulation provided by the Modelica framework by the 3D simulation capabilities of the Gazebo simulator. In particular, this paper focuses on adding customized body-to-body interaction models to the standard components of the MultiBody library, combining the advanced capabilities of collision detection provided by the Gazebo framework with the flexibility provided by the Modelica environment to define sophisticated, tailor-made, equation-based physical models. It must be emphasised, however, that this topic is not important per se, instead it represents a proof-of-concept of the possibility of integrating the two simulation environments in order to set up a new one that is able to better address graphical and physical aspects as well. As a consequence, the contribution of this paper is not related to an innovative or improved interaction model, but to the framework that allows to extend Modelica modelling capabilities by the 3D Gazebo simulation.

The paper is structured as follows. Section 2 describes the design of the modelling framework. In the following Section 3, a proof-of-concept implementation is described, and the results obtained with a simple sphere-to-plane interaction simulation are presented. Section 4 concludes the paper with an outlook to further developments.

2 Design of the modelling framework

The rationale behind the design is to let Gazebo and the Modelica tool each perform the tasks at which they excel, for which they already have good built-in support, and which are more conveniently programmed by the end-user.

[6] `http://www.modelica.org`
[7] `http://www.simulationx.com`

Modelica will then be used for the accurate and tailor-made dynamic modelling of the multi-body objects for which the standard modelling approach of the physical engine embedded in Gazebo is not adequate. Modelica could also be used to represent low-level sensing, actuation and control, such as electric motors and drives, pneumatic actuation, low-pass signal filtering, etc., which are not covered by Gazebo, when their accurate modelling is essential to assess the success or failure of higher-level control functions.
All other tasks, such as building and managing the scenes, simulating other objects for which ad-hoc dynamic modelling is not required, simulating vision-based sensing, and providing geometrical information about object collisions, will be managed by Gazebo.

The present paper focuses on the integration between Gazebo and Modelica to provide accurate ad-hoc physical modelling where needed. How the resulting physical model can then be integrated in the Gazebo environment, together with all the other objects and functions simulated by Gazebo, goes beyond the scope of this paper and will be addressed in future works.

The basic framework for the modelling of multi-body objects is provided by the Modelica MultiBody library, which allows to build modular models of multi-body systems by the connection of link and joint models. Since the Gazebo engine also uses corresponding primitives, automatically generating the Modelica code of the model corresponding to any Gazebo multi-body object is a straightforward task. The availability of flexible link models compatible with the MultiBody library, e.g., those described in (Ferretti et al., 2014), allows to easily take into account flexibility in all those cases where this is crucial to replicate the system dynamic behaviour. This is a feature that could be very useful in the case of soft or flexible robots and which is still not present in Gazebo.

A key ingredient of any multi-body model of robots or autonomous vehicles is the modelling of the interaction between different bodies, in particular the tyre-road interaction in vehicles and the interaction between hands or grippers and objects to be manipulated for robots. For this purpose, Gazebo provides so-called collider objects, which take as input the position of the reference frames of any two objects, possibly having a complex shape, and returns information about the presence or absence of contact points, their location, the depth of penetration, and the normal vectors to the object surface at the contact point. Gazebo can also compute the resulting interaction forces and torques, according to some standard embedded model; the idea in the context of this paper is to ignore this information and use Modelica instead to compute them, according to a tailor-made equation-based physical model that is appropriate for the specific simulation scenario.

The Modelica code of the base model for two-body interaction, `PhysicalInteraction`, is listed in the appendix. The model extends the `PartialTwoFrames` model of the MultiBody library. It gets the position and

DOI 10.3384/ecp17132887

orientation of the two potentially interacting objects from the two frame connectors and passes them to the `collisionDetectionModelica` function. This in turn converts the rotation objects into quaternions and calls the external function `collisionDetection`, that passes the two object ID strings and their position and orientation to the Gazebo server. The collider in Gazebo responds returning the number of contact points, the arrays of contact points on both bodies, as well as the penetration depths and the normals to the surface for each contact point.

This data is then passed to the replaceable function `computeInteraction`, which uses the kinematic information to compute the forces and torques exerted on body a by body b. As Modelica functions cannot generate events, a conditional equation is then written in the `PhysicalInteraction` model, which applies the forces and torques computed by the external function to the connector if the penetration depth is positive, zero otherwise. This allows to precisely compute the contact event instant and handle the discontinuity properly, if the Modelica solver provides proper event handling. Finally, the corresponding forces and torques applied on body b by body a are computed by Newton's 3$^{\text{rd}}$ law.

In this context, the Gazebo tool only acts as a server, providing the service of computing the kinematic information regarding the collisions between any two objects of interest. The physical simulation is carried out by the code generated by the Modelica tool, which is the simulation master. This means that the sequence of calls to the Gazebo server does not correspond to a physical sequence of points in time, but rather to the individual function calls required by the Modelica solver, which might go backward and forward in time to compute a solution, e.g., when locating event instants or when a time step is rejected by an adaptive step-size solver. As the Gazebo tool is not the master of the simulation in this context, this is not a problem. In fact, time is not even part of the data which is communicated to the Gazebo server from the Modelica side.

Specific physical interaction models can then be obtained by extending the `PhysicalInteraction` class and by redeclaring the `computeInteraction` function with the specific algorithm that computes the interaction forces and torques, based on the model of interest for the end user. All the infrastructure provided by the Modelica MultiBody library can be used to carry out this task with ease, in particular the functions to resolve vectors in different reference frames and all the functions implementing vector algebra operations.

3 Proof of concept

In this section, a proof-of-concept implementation that demonstrates the proposed approach is presented.

3.1 Implementation details

In order to avoid all the problems related to memory management, in this implementation the external C function `computeInteraction` uses Unix IPC sockets to communicate with the Gazebo server. In the future, this mechanism will be substituted by some more efficient, shared-memory based communication, e.g., by embedding the Modelica model into an FMI and using external objects to set up the communication framework.

A simple exemplary test case has been selected for the demonstration, namely the interaction between an elastic ball and a fixed, rigid plane. When the two bodies collide, the force F_a applied on the sphere at the point of contact is the sum of three components:

$$F_a = F_e + F_d + F_f.$$

The elastic force F_e is directed as the normal vector (which points to the sphere's centre) and its magnitude is computed according to (Nassauer and Kuna, 2013)

$$F_e = k_e \sqrt{V d},$$

where k_e is an elastic constant, d is the penetration depth, and V is the volume of the spherical cap of height d

$$V = \pi d^2 \left(r - \frac{d}{3} \right).$$

The damping force F_d is proportional to the normal component v_n of the relative velocity between the two bodies at the point of contact and opposed to it, thus providing dissipation each time the sphere hits the plane.

The friction force F_f depends on the tangential component of the relative velocity v_t at the point of contact, has the opposite direction and a magnitude

$$-\mu F_e \frac{v_t}{\sqrt{v_t^2 + v_\varepsilon^2}};$$

where μ is the dry friction coefficient, v_ε is a small velocity threshold, and the fraction is approximately equal to one for $v_t \gg v_\varepsilon$ and approaches zero as $v_t \to 0$. This model is not accurate at low relative velocities, since it leads to a slow sliding at velocities around v_ε instead of proper stiction. On the other hand, it has the nice property of not becoming singular at zero relative velocity and is perfectly adequate for the purposes of this demonstration. Other more sophisticated models that include stiction, such as the one described in (Deur et al., 2004) could be employed if needed.

As to the torques, only the friction force exerts a net torque on the sphere's frame connector, located at the centre of the sphere; the torque vector is simply $\tau = r \times F_t$. For simplicity, the torsional torque due to rolling friction has been neglected in this demonstrator.

3.2 Test cases and simulation results

The results of three sphere-to-plane interaction simulations are here presented. The sphere represents a big inflated balloon, modelled as a hollow sphere of mass $m = 1$

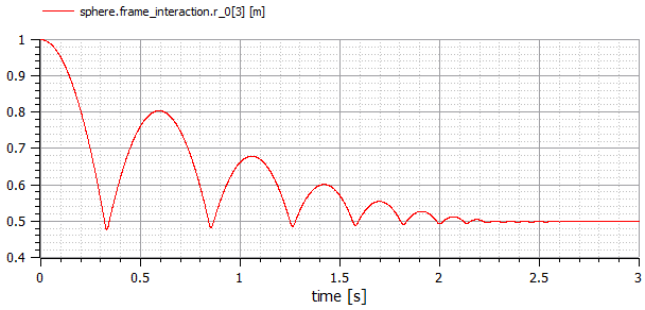


Figure 1. Simulation 1 – Ball height over time.

kg, radius $r = 0.5$ m, moments of inertia $J = \frac{2}{3}mr^2$, elastic constant $k_e = 10^3$ N/m^2 and with a relatively low friction coefficient $\mu = 0.1$. Air friction is neglected. All the simulations start with the center of the sphere at a height of 1 m above the horizontal xy-plane, the z-axis pointing upwards.

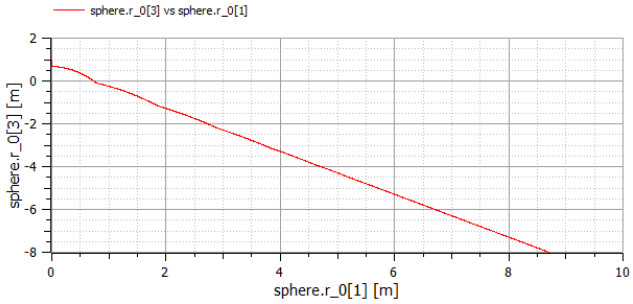
The Modelica code was compiled into executable simulation code with the OpenModelica compiler[8] version 1.12.0-dev, using a Runge-Kutta fixed time step integration algorithm with a time step of 1 ms, which is short enough to correctly handle the elastic impacts, whose typical duration is about 10 ms.

The simulation were first tested by emulating the response of the Gazebo server by a Modelica function. This required to extend the `Sphere2Plane` physical interaction model, which uses the external function calling Gazebo, and to redeclare the `collisionDetection-Modelica` function so that it directly computes the contact point locations, depths of penetration and normal vectors, rather than calling the external function and getting them from Gazebo. This function is implemented easily in Modelica, as the geometry of the sphere-to-plane interaction is extremely simple. Eventually, the same simulation results were obtained when using the Gazebo server, thus validating the entire proof-of-concept implementation. Also, the qualitative behaviour of the system in the three simulations corresponds to what one would expect from physical intuition.

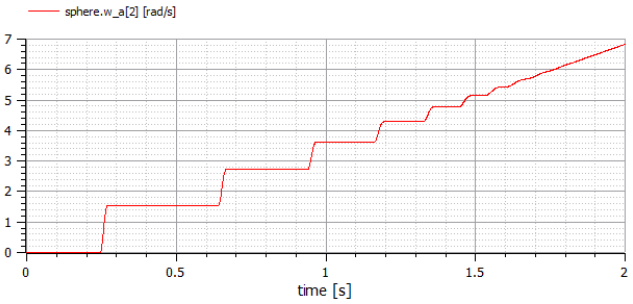
Many different simulations were run, in order to validate each component (elastic, damping, friction) of the interaction forces and torques separately. In this paper, the results of three simulation experiments with realistic choices of the interaction model parameters are reported.

In the first simulation, the plane is horizontal and the sphere has zero initial velocity and angular velocity. As expected, the ball falls onto the plane and bounces a few times before getting to rest, due to the dissipative effect of F_d. Figure 1 shows the vertical position of the sphere centre over time.

The second simulation scenario is similar, save that the plane is tilted by 45° along the y-axis. When the ball hits the plane, it bounces off horizontally. Due to friction, it also gets some angular momentum on the y-axis during the bounce, and thus starts spinning slowly. It then bounces a few more times on the tilted plane until dissipation causes it to remain in contact with the tilted plane and to accelerate while rolling downwards. Figure 2(a) shows the trajectory of the sphere's centre in the vertical plane, while Figure 2(b) shows the angular momentum over time, which increases abruptly at each bounce and finally increases with a constant slope once the sphere stops bouncing and rolls down on the plane surface always remaining in contact.



(a) Trajectory of the sphere centre in the xz-plane



(b) Angular velocity of the sphere in the y-axis direction

Figure 2. Simulation 2 – Ball bouncing on a tilted plane.

The last simulation considers again a horizontal plane; in this case the sphere starts with a non-zero horizontal velocity in the negative x-axis direction, spinning fast backward around the y-axis. Every time the ball bounces on the plane, the friction force slows down the spinning a bit, and accelerates the sphere in the positive x-axis direction, so that eventually the ball changes its horizontal direction and rolls back to a point on the plane below the initial position. Figure 3(a) shows the position of the sphere's center in the vertical xz-plane, while Figure 3(b) shows the angular momentum along the y-axis over time[9].

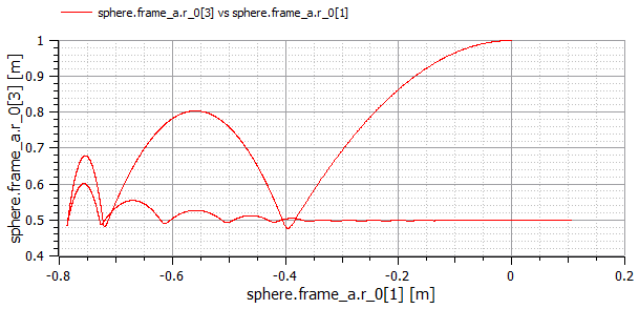
4 Conclusions

In this paper, a proof-of-concept for the integration between the Gazebo 3D robotic simulation tool and Modelica has been presented. The proposed framework al-

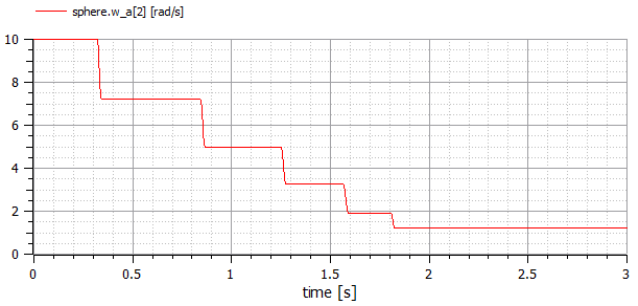
[8] `https://openmodelica.org`

[9] The 3D videos generated by Gazebo of the three simulations are available online at this URL: `https://home.deib.polimi.it/casella/gazebo/videos.html`.

DOI 10.3384/ecp17132887



(a) Trajectory of the sphere centre in the *xz*-plane



(b) Angular velocity of the sphere in the *y*-axis direction

Figure 3. Simulation 3 – Ball starting with a non-zero horizontal velocity in the positive *x*-axis direction and spinning backward around the *y*-axis.

lows to extend the basic 3D multi-body engine embedded in Gazebo, by providing equation-based customized 3D multi-body dynamics. The extension is very convenient and easy to implement, as it leverages on the existing sophisticated collision detection functionality of Gazebo, on the Modelica MultiBody library, and on the possibility of describing an ad-hoc physical behaviour in a high level, equation-based modelling environment. It also makes it possible to perform equation-based multi-domain physical modelling, e.g., by adding Modelica models of physical sensors, actuators and low-level controllers to the mechanical model, and in general by modelling any kind of physical behaviour beyond that of multi-body systems.

The framework has been demonstrated with a proof-of-concept implementation, using IPC sockets to enable the communication between the Gazebo tool and Modelica automatically generated simulation code. In particular, the results of the simulations of a simple system with an elastic ball bouncing on a rigid plane with low friction have been presented. The obtained results are very encouraging and suggest that it might indeed be possible to propose these Modelica extensions, implemented with the open-source OpenModelica compiler, as the preferred way to extend the native Gazebo simulation engine.

To reach our final aim, further developments are under investigation. First of all, we would like to validate the concept with scenarios involving multiple object interactions; currently we already generate Modelica simulation code in the presence of multiple object, what has to be validated is the collision between multiple objects handled by Modelica. To improve on performance and ease of deployment, we are currently encapsulating the Modelica model in an FMU to handle the communication with Gazebo via shared memory and external object interface. Once the FMU will be integrated with the Gazebo plug-in mechanism, it will be possible to integrate the FMU-based simulation into the master simulation loop of the Gazebo tool in a seamless way and transparently to the designer of the simulation.

Finally, we would like to experiment with hybrid simulations with some physical behaviour simulated by the Gazebo physics engine and some physical behaviour with special modelling requirements simulated by the Modelica/FMU code. This set-up could be necessary to handle demanding simulation scenarios with many objects, since we expect the Modelica-based simulation code to be slower than the native and somewhat simplified Gazebo simulation engine, so that using Modelica only where needed could end up in much faster simulations.

References

G. Bardaro, D.A. Cucci, L. Bascetta, and M. Matteucci. A simulation based architecture for the development of an autonomous All Terrain Vehicle. In *SIMPAR*, pages 74–85, 2014.

G. Bardaro, L. Bascetta, F. Casella, and M. Matteucci. Advancement in multi-body physics modeling for 3d graphical robot simulators. In *Workshop on Modelling and Simulation for Autonomous Systems*, pages 189–195, 2016.

S. Chitta, E.G. Jones, M. Ciocarlie, and K. Hsiao. Mobile manipulation in unstructured environments: Perception, planning, and execution. *IEEE Robotics & Automation Magazine*, 19(2):58–71, 2012.

E.L. D'Amelio, L. Bascetta, D.A. Cucci, M. Matteucci, and G. Bardaro. A modelica simulator to support the development of the control system of an autonomous all-terrain mobile robot. In *International Conference on Mathematical Modelling*, pages 274–279, 2015.

Joško Deur, Jahan Asgari, and Davor Hrovat. A 3D brush-type dynamic tire friction model. *Vehicle System Dynamics*, 42(3): 133–173, 2004. doi:10.1080/00423110412331282887.

Evan Drumwright, John Hsu, Nathan Koenig, and Dylan Shell. Extending Open Dynamics Engine for robotics simulation. In *Proceedings of the Second International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, SIMPAR'10, pages 38–50. Springer-Verlag, 2010.

Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

Gianni Ferretti, Alberto Leva, and Bruno Scaglioni. Object-oriented modelling of general flexible multi-body systems. *Mathematical and Computer Modelling of Dynamical Systems*, 20(1):1–22, 2014. doi:10.1080/13873954.2013.807433.

M. Ko, B.-S. Ryuh, K.C. Kim, A. Suprem, and N.P. Mahalik. Autonomous greenhouse mobile robot driving strategies from system integration perspective: Review and application. *IEEE/ASME Transactions on Mechatronics*, 20(4): 1705–1716, 2015.

Benjamin Nassauer and Meinhard Kuna. Contact forces of polyhedral particles in discrete element method. *Granular Matter*, 15(3):349–355, 2013. doi:10.1007/s10035-013-0417-9.

M. Otter, H. Elmqvist, and S. E. Mattsson. The new Modelica MultiBody library. In *Proceedings 3rd International Modelica Conference*, pages 311–330, Linköping, Sweden, Nov. 3–4 2003.

B. Paden, M. Cap, S. Zheng Yong, D. Yershov, and E. Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016.

M.A. Roa, D. Berenson, and W. Huang. Mobile manipulation: Toward smart manufacturing. *IEEE Robotics & Automation Magazine*, 22(4):14–15, 2015.

A Listing of the `PhysicalInteraction` model

```
model PhysicalInteraction "Base class for all physical interaction models"
  extends Modelica.Mechanics.MultiBody.Interfaces.PartialTwoFrames;
  import Modelica.Mechanics.MultiBody.Frames;
  parameter Integer maxContacts = 10 "Number of max contact points";
  parameter String id_a = "" "Id of interacting object a";
  parameter String id_b = "" "Id of interacting object b";
  Real numberOfContactPoints "Number of actual contact points";
  Real cp_a[maxContacts, 3] "Array of contact points on body a, resolved in frame_a";
  Real cp_b[maxContacts, 3] "Array of contact points on body b, resolved in frame_b";
  Real depth_a[maxContacts] "Array of penetration depths in body a";
  Real depth_b[maxContacts] "Array of penetration depths in body a";
  Real normals_a[maxContacts, 3] "Array of normals on body a, resolved in world frame";
  Real normals_b[maxContacts, 3] "Array of normals on body b, resolved in world frame";
  Real r[3] "Vector from frame_a to frame_b resolved in frame_a";
  SI.Force f_a[3] "Interaction force applied on body a, resolved in frame_a";
  SI.Torque t_a[3] "Interaction torque applied on body b, resolved in frame_b";

  replaceable function collisionDetectionModelica
    input Integer maxContacts "Maximum number of contact points";
    input Real r_a[3] "Position vector of interaction frame of object a, resolved in world frame";
    input Frames.Orientation R_a "Orientation of interaction frame of object a";
    input String id_a "unique id for object a";
    input Real r_b[3] "Position vector of interaction fram of object b, resolved in world frame";
    input Frames.Orientation R_b "Orientation of interaction frame of object b";
    input String id_b "unique id for object b";
    output Real numberOfContactPoints "Number of actual contact points";
    output Real cp_a[maxContacts, 3] "Array of contact points on body a, resolved in frame_a";
    output Real cp_b[maxContacts, 3] "Array of contact points on body b, resolved in frame_b";
    output Real depth_a[maxContacts] "Array of penetration depths in body a";
    output Real depth_b[maxContacts] "Array of penetration depths in body a";
    output Real normals_a[maxContacts, 3] "Array of normals on body a, resolved in frame_a";
    output Real normals_b[maxContacts, 3] "Array of normals on body b, resolved in frame_b";
  algorithm
    (numberOfContactPoints, cp_a, cp_b, depth_a, depth_b, normals_a, normals_b) :=
      collisionDetection(maxContacts, r_a, Frames.to_Q(R_a), id_a, r_b, Frames.to_Q(R_b), id_b);
  end collisionDetectionModelica;

  function collisionDetection
    input Integer maxContacts "Maximum number of contact points";
    input Real r_a[3] "Position vector of interaction frame of object a, resolved in world frame";
    input Frames.Quaternions.Orientation Q_a "Quaternion of the orientation of interaction frame of object
        a";
    input String id_a "unique id for object a";
    input Real r_b[3] "Position vector of interaction fram of object b, resolved in world frame";
    input Frames.Quaternions.Orientation Q_b "Orientation of interaction frame of object b";
    input String id_b "unique id for object b";
    output Real numberOfContactPoints "Number of actual contact points";
    output Real cp_a[maxContacts, 3] "Array of contact points on body a, resolved in frame_a";
    output Real cp_b[maxContacts, 3] "Array of contact points on body b, resolved in frame_b";
    output Real depth_a[maxContacts] "Array of penetration depths in body a";
    output Real depth_b[maxContacts] "Array of penetration depths in body a";
    output Real normals_a[maxContacts, 3] "Array of normals on body a, resolved in frame_a";
    output Real normals_b[maxContacts, 3] "Array of normals on body b, resolved in frame_b";
  external "C"
  end collisionDetection;
```

```
  replaceable partial function computeInteraction "Compute interaction torques and forces on frame_a,
      resolved in frame_a"
    input Real numberOfContactPoints "Number of actual contact points";
    input Integer maxContacts "Maximum number of contact points";
    input Real r_a[3] "Position of frame_a resolved in world frame";
    input Real r_b[3] "Position of frame_b resolved in world frame";
    input Real v_a[3] "Velocity of frame_a resolved in world frame";
    input Real v_b[3] "Velocity of frame_b resolved in world frame";
    input Frames.Orientation R_a "Orientation of frame_a";
    input Frames.Orientation R_b "Orientation of frame_b";
    input Real cp_a[maxContacts, 3] "Array of contact points on body a, resolved in frame_a";
    input Real cp_b[maxContacts, 3] "Array of contact points on body b, resolved in frame_b";
    input Real depth_a[maxContacts] "Array of penetration depths in body a";
    input Real depth_b[maxContacts] "Array of penetration depths in body a";
    input Real normals_a[maxContacts, 3] "Array of normals on body a, resolved in frame_a";
    input Real normals_b[maxContacts, 3] "Array of normals on body a, resolved in frame_a";
    output SI.Force[3] f_a "Equivalent force applied to frame_a, resolved in frame_a";
    output SI.Torque[3] t_a "Equivalent torque applied to frame_a, resolved in frame_a";
  end computeInteraction;

equation
  (numberOfContactPoints, cp_a, cp_b, depth_a, depth_b, normals_a, normals_b) =
    collisionDetectionModelica(maxContacts, frame_a.r_0, frame_a.R, id_a, frame_b.r_0, frame_b.R, id_b);
  assert(numberOfContactPoints <= maxContacts, "Too many contact points");
  (f_a, t_a) = computeInteraction(numberOfContactPoints, maxContacts,
    frame_a.r_0, frame_b.r_0, der(frame_a.r_0), der(frame_b.r_0), frame_a.R, frame_b.R,
    cp_a, cp_b, depth_a, depth_b, normals_a, normals_b);
  if sum(depth_a + depth_b) > 0 then
    frame_a.f = f_a;
    frame_a.t = t_a;
  else
    frame_a.f = {0, 0, 0};
    frame_a.t = {0, 0, 0};
  end if;
  r = Frames.resolve2(frame_a.R, frame_b.r_0 - frame_a.r_0);
  zeros(3) = frame_a.f + Frames.resolveRelative(frame_b.f, frame_b.R, frame_a.R);
  zeros(3) = frame_a.t + Frames.resolveRelative(frame_b.t, frame_b.R, frame_a.R) - cross(r, frame_a.f);
end PhysicalInteraction;
```


DOI 10.3384/ecp17132887