Modeling Contact and Collisions for Robotic Assembly Control

Scott A. Bortoff[1]

[1]Mitsubishi Electric Research Laboratories, Cambridge, MA, USA, bortoff@merl.com

Abstract

We propose an implicit, event-driven, penalty-based method for modeling rigid body contact and collision that is useful for design and analysis of control algorithms for precision robotic assembly tasks. The method is based on Baumgarte's method of differential algebraic equation index reduction in which we modify the conventional constraint stabilization to model object collision, define a finite state machine to model transition between contact and non-contact states, and represent the robot and task object dynamics as a single set of differential algebraic inequalities. The method, which is realized natively in Modelica, has some advantages over conventional penalty-based methods: The resulting system is not numerically stiff after the collision transient, it enforces constraints for object penetration, and it allows for dynamic analysis of the Modelica model beyond time-domain simulation. We provide three examples: A bouncing ball, a ball maze, and a delta robot controlled to achieve soft collision and maintain soft contact with an object in its environment.

Keywords: robotics, control

1 Introduction

To derive new control algorithms for robust robot assembly, it is important to construct system-level dynamic models of the robotic manipulator *and* the assembly task, including appropriate representations of object contact and collision *and also* the control algorithms themselves. Of course, Modelica is well-suited for all three of these domains. It is also important that such models should be useful for more than time-domain simulation. For example, we should be able to linearize the full model at various operating conditions, and to compute things such as a multi-variable frequency response for rigorous design and analysis of new types of feedback control algorithms. We also should be able to compute some model objects for purposes of real time model-based estimation and control.

Our long-term objectives are (1) to invent new control algorithms, especially for the delta robot, that make assembly processes robust with respect to uncertainty in the environment (especially uncertainty in the location of an object), and exploit new types of sensors, such as touch sensors, (2) to accelerate the process of experimental validation, and (3) to accelerate and simplify the process of controller parameter tuning that is done at commissioning time. Toward these ends, we not only seek to model and simulate robotic assembly, but we also desire a mathematical abstraction of robotic assembly that is consistent with the Modelica software representation, and is useful for synthesis and analysis of robust, hybrid feedback control algorithms. In addition, we intend to use our developing Modelica library of robot manipulators, assembly tasks and control algorithms together with the Modelica Device Drivers library (Thiele et al., 2017) to control the delta robot in our laboratory in various assembly experiments, without having to recode any aspect of the control algorithm, although this work is beyond our scope here.

In this paper, we propose an event-driven model of rigid body collision and contact that combines differential algebraic inequalities to represent rigid body motion with a Finite State Machine (FSM) to represent the state of contact and collisions. Rigid bodies in contact may be modeled as a set of differential equations coupled via a vector of Lagrange multipliers λ to a set of algebraic equations that represent the contact, resulting in an index-3 Differential Algebraic Equation (DAE). Baumgarte's method of index reduction (Baumgarte, 1972, 1983) replaces the algebraic constraint equations with a linear combination of their derivatives with respect to time, resulting in an index-1 (or 0) set of DAEs (or ODEs). The contribution of this paper is to allow the structure of this DAE to change dynamically from an unconstrained state ($\lambda = 0$) to a constrained state ($\lambda > 0$) in order to model the physical process of collision and subsequent contact (and vis versa), and to observe that the dynamics introduced by Baumgarte's method can in fact model the transient associated with the collision. We introduce the FSM to represent the discrete state of contact, resulting in a hybrid DAE capturing collision, contact, and loss of contact. This mathematics can be represented in the Modelica language natively, and is consistent with other common models of robotic manipulators, allowing for integrated modeling, simulation and analysis of the robotic system and task.

The paper is organized as follows. In Section 2 we propose the contact and collision model in general terms, list its properties, advantages and disadvantages compared to other state-of-the-art methods, and provide a simple bouncing ball example. In Section 3 we present a second example: A ball maze. We design a feedback control algorithm to solve the maze and simulate the closed-loop system, which exhibits a sequence of collisions, using the proposed model. Finally, we show initial results for a delta robot assembly task (Bortoff, 2018, 2019), where control must achieve soft collision between the manipulator and a workspace object. Conclusions are drawn in Section 5.

2 Collision and Contact Model

Modeling collisions and contact among solid objects is a well-studied subject and we refer the reader to the vast literature on the subject, including works associated with the computer graphics community (Erleben et al., 2005). Collision models have been developed for the Modelica MultiBody library, notably (Engelson, 2000; Otter et al., 2005; Hofmann et al., 2011a), resulting in the third-party IdealizedContact library (Hofmann et al., 2011b). These references discuss several methods of contact detection and reaction, including impulsive methods and penalty-based methods. Several works have integrated Modelica with third-party software such as the Bullet Physics Library[1] or Gazebo[2], especially for collision detection but also to exploit their animation capability and potentially tap the large collection of robotic technologies represented in these tools such as advanced sensors (Bardaro et al., 2017). However, the collision detection algorithms used by these third party tools are intended primarily for animation, not dynamic analysis, and have limitations discussed in the literature e.g., requiring a non-zero collision margin or having difficulty with very large and very small shapes (Coumans, 2015).

In this paper we propose an event-triggered, implicit penalty (force) - based collision approach that is native to Modelica, and has some advantages (Pros) over conventional penalty - based methods:

P1 It does not result in a stiff set of ODEs, even for stiff material properties,

P2 It results in steady-state solutions without object penetration, although object penetration occurs during the transient collision phase, and

P3 It is relatively easy to understand and implement, completely in Modelica, which allows representation of the complete physics of a problem in a single tool.

On the other hand, our proposed method has three disadvantages (Cons):

C1 It does not conserve energy before and after elastic collisions,

C2 It is event-driven, meaning it relies on the Modelica tool's ability to properly detect events, and

C3 For large numbers of objects, it will certainly be less memory and time efficient than alternatives.

Despite these disadvantages, the method is useful for our purpose, and even **C1** is not significant except for very limited and well-defined circumstances that are generally outside our interest.

To begin, assume that the environment includes a number of other rigid bodies, which we denote as *task objects*, whose collective generalized position and velocity are denoted q_c and v_c respectively. For the task objects, the unconstrained Lagrangian equations of motion are assumed to be

$$\dot{q}_c = v_c \tag{1a}$$

$$M_c(q_c)\dot{v}_c + C_c(q_c, v_c) + D_c(v_c) + G_c(q_c) = f_c, \tag{1b}$$

where M_c is the inertia matrix, C_c represents Coriolis and centripetal forces and torques, G_c represents forces and torques due to gravity, D_c represents frictional forces and torques, f_c represents external forces and torques, and the subscript c denotes "constraint." In addition, we assume that all geometric constraints among the task objects can be expressed as the N_c-dimensional vector inequality

$$h_c(q_c) \geq 0, \tag{2}$$

and that all the geometric constraints between the task objects and the robotic manipulator can be expressed as the N_r-dimensional vector inequality

$$h_{rc}(q, q_c) \geq 0. \tag{3}$$

With this notation, two task objects are in geometric contact if at least one of the corresponding inequalities is zero; otherwise the task objects are not in geometric contact. The functions h_c and h_{rc} are, generally speaking, the distance between key features of the task objects and robotic manipulator, respectively.

For example, if the task objects were three solid cubes above a rigid surface, then (1) is the 36-dimensional rigid body equations for the blocks, with each body having three translational and three rotational degrees of freedom. The matrix M_c is the 18×18 block-diagonal inertia matrix, C_c represents Coriolis and centripetal forces and torques, G_c represents forces and torques to to gravity, and D_c represents frictional forces and torques on the collection of blocks. The vector h_c includes distances from each vertex to the surface, and distances between the vertices and edges of each block in order to capture the constraint that every vertex edge of every cube must lie outside the volume of all other cubes.

In practice, N_c and N_{rc} may be large, and grow unfavorably as the number of objects increase. Physics engines such as Bullet reduce the dimension of inequalities that must be considered in the collision detection problem by eliminating from consideration those pairs of objects that are far away, i.e., $h_{ci}(q_c) >> 0$ for some i, using heuristics like bounding boxes, and by computing only the *minimum* distance between pairs objects e.g. (Gilbert et al., 1988), thereby reducing the number of inequalities in the collision detection problem. These aspects are important in typical computer graphics applications. In this paper we do not concern ourselves with efficiency, and concede that (2) and (3) may be high-dimensional, although for many specific assembly problems, these are reasonably sized.

[1] https://pybullet.org

[2] http://gazebosim.org

We further assume that the unconstrained (free of contact) robot manipulator is modeled in the usual manner:

$$\dot{q} = v \tag{4a}$$
$$M(q)\dot{v} + C(q,v) + G(q) = Bu. \tag{4b}$$

We now propose that Baumgarte's method of index reduction, which stabilizes a holonomic equality constraint $h(q) = 0$ when the constraint is active, can be extended and used to model both solid object contact and also collision, if we cause the Lagrange multipliers that correspond to object-to-object or object-to-robot contact to satisfy an inequality constraint, and optionally if we modify the coefficients and the structure of the constraint stabilization to represent stiff elastic collision. There are two parts to this model: The physics equations of constrained motion, and a finite state machine (FSM) that determines the state of contact between objects.

We first present the physics equations. Define $\lambda_c \in \mathbb{R}^{N_c}$ and $\lambda_{rc} \in \mathbb{R}^{N_{rc}}$ be Lagrange multiplier vectors of the same dimension as h_c and h_{rc}, respectively. For now, denote the state of each FSM as $c_i \in \{0, 1, 2\}$, for $1 \le i \le N_c + N_{rc}$, with $c_i = 1$ corresponding to the contact state. Then we can define the equations of motion for the combined robot-task system as

$$\dot{q} = v \tag{5a}$$
$$M(q)\dot{v} + C(q,v) + G(q) = \lambda^T \frac{\partial h}{\partial q} + \lambda_{rc}^T \frac{\partial h_{rc}}{\partial q} + Bu \tag{5b}$$
$$\ddot{h} + \alpha_1 \dot{h} + \alpha_0 h = 0, \tag{5c}$$

and

$$\dot{q}_c = v_c \tag{6a}$$
$$M_c(q_c)\dot{v}_c + C_c(q_c, v_c) + G_c(q_c) = \lambda_{rc}^T \frac{\partial h_{rc}}{\partial q_c} + \lambda_c^T \frac{\partial h_c}{\partial q_c} + f \tag{6b}$$

and, for $1 \le i \le N_c$ and $1 \le j \le N_{rc}$,

$$\text{If } c_i = 1 \text{ then } \ddot{h}_{ci} + \alpha_1 \dot{h}_{ci} + \alpha_0 h_{ci} + \alpha_2 h_{ci}^3 = 0 \text{ else } \lambda_{ci} = 0 \tag{7a}$$
$$\text{If } c_j = 1 \text{ then } \ddot{h}_{cj} + \alpha_1 \dot{h}_{rcj} + \alpha_0 h_{rcj} + \alpha_2 h_{rcj}^3 = 0 \text{ else } \lambda_{rcj} = 0. \tag{7b}$$

In short, if a constraint is active, i.e., if two objects are in contact, then the corresponding Lagrange multiplier is an algebraic state-variable of the index-1 DAE and the corresponding stabilizing constraint equation is active. Otherwise, the Lagrange multiplier is set to zero, and the corresponding constraint stabilizing equation does not appear. This ensures that the number of equations and variables is the same, regardless of the constraint state.

The FSM for each constraint is required for a subtle reason. It might seem that the logic for constraint activation is that the constraint i (or j) becomes active when $h_{ci} \le 0$ (or $h_{cri} \le 0$), and becomes inactive when $\lambda_{ci} < 0$ (or $\lambda_{rcj} < 0$), for $1 \le i \le N_c$, $1 \le j \le N_{rc}$, giving two well-defined states. (For this point, we drop the subscripts for notational simplicity.) However, in Modelica it is not good practice to have an activation condition (realized using either `when` or `if`), depend on two different variables. Further, it can occur that $\lambda < 0$ while $h < 0$, because of numerical errors when both are near zero, or because forces due to other objects cause $\lambda < 0$ while $h < 0$. In fact, this is common in practice. We therefore define the three-state FSM diagrammed in Figure 1, which ensures correct transition to and from contact. The Ballistic state is included for the case that $h < 0$ and $\lambda < 0$, to ensure that the constraint force can not be negative and pull objects together. Physically it must always be repulsive i.e., positive.

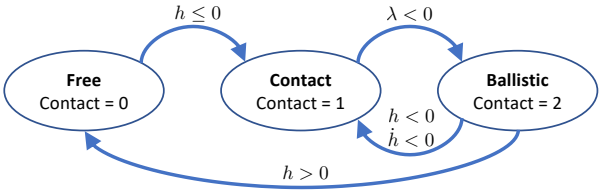

Figure 1. Finite State Machine (FSM) for contact.

Several remarks are in order. Note that the Lagrange multipliers have the physical interpretation as the force between two objects required to drive the constraint to zero, according to the second-order stabilized constraint equation (7). Also, the penetration between objects due to collision is governed only by corresponding second-order stabilized constraint equation (7), which is independent of the other dynamical equations by design. The parameters α_0, α_1 and α_2 can be tuned for the specific material stiffness and damping properties. Importantly, we have added a nonlinear cubic term in order to capture nonlinear force behavior due to penetration (Hofmann et al., 2011a). This can be any odd-order polynomial or similar function, as long as (7) is stable.

This formulation has three advantages over more conventional penalty-type methods that explicitly compute a force between objects in contact as a function of penetration. First, the resulting dynamics (5)-(7) are not stiff after the collision transient due to (7) has transpired. If we choose values of α_0 and α_1 that are of the same time-scale as the rigid dynamics, while α_3 may be large to account for stiff material properties, then we see that the Lagrange multiplier is precisely the force that will drive the constraint to zero, at which point the eigenvalues correspond to roots of $s^2 + \alpha_1 s + \alpha_0$, which by design is not stiff. In other words, we can model collisions between very stiff objects by making $\alpha_3 >> 0$, and after the transient from the collision transpires, assuming that the objects remain in contact, the dynamics will have eigenvalues at the roots of $s^2 + \alpha_1 s + \alpha_0$, so the system need not be stiff. Of course, the system will be stiff during the collision transient, necessitating small simulation time steps during this phase, but a variable step solver will be able to increase step size after contact is established between objects, and the transient due to (7) has transpired.

The second advantage is that the stabilized constraint equations (7) drive the constraint to zero when the constraint remains active. This means that, after the transient due to that specific collision transpires, the constraint $h = 0$ is exactly enforced and there is no penetration, aside from numerical error which is of the solver tolerance. The constraint $h = 0$ will be enforced even if the dynamic system continues to evolve: It does not need to be in steady-state. On the other hand, if we were to compute the force between objects using a "spring-damper" model, for example, then there would be nonzero penetration after the collision, it would remain nonzero due to the dynamic behavior of the rest of the system, even in steady-state. The penetration can be made small, but at the expense of using a stiff virtual spring, making the ODE stiff.

A bouncing ball is a good example, with Modelica code as follows.

```
model myBouncingBall

Real q(start=1.0), v(start=0.0), f;
Real h, hDot, hDotDot, lambda(start=0);
discrete Integer contact(start=0);
Boolean b1, b2, b3, b4;
parameter Real g=9.81, m=1.0;
parameter Real a0=100, a1=20, a2=1e6;
parameter Boolean linFlag = false;

algorithm
b1 := h <= 0;
b2 := lambda < 0.0;
b3 := h > 0.0;
b4 := h <= 0 and hDot < 0;
if edge(b1) and contact == 0 then
  contact := 1;
end if;
if contact == 1 and edge(b2) then
  contact := 2;
end if;
if contact == 2 and edge(b3) then
  contact := 0;
end if;
if contact == 2 and edge(b4) then
  contact := 1;
end if;

equation
if contact == 1 or linFlag then
  0 = hDotDot + a1*hDot + a0*h + a2*h^3;
else
  lambda = 0.0;
end if;

f = if linFlag then lambda
     else contact*lambda;

der(q) = v;
m * der(v) = -m * g + f;

h = q;
hDot = der(h);
hDotDot = der(hDot);

end myBouncingBall;
```

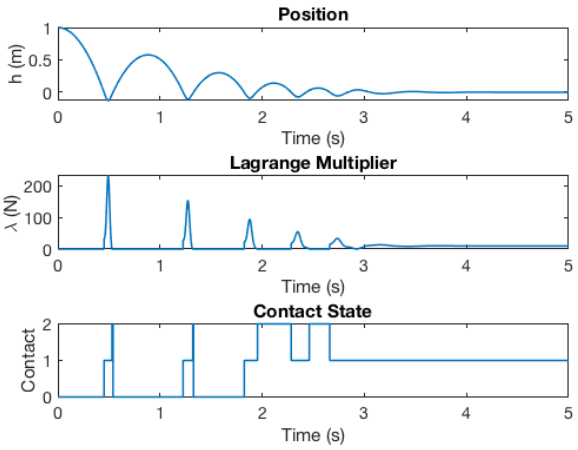

Figure 2. Bouncing ball simulation for stiff materials and low damping ($\alpha_1 = 100$, $\alpha_1 = 1$, $\alpha_2 = 1e6$,). Note that the contact state passes through the "Ballistic" state, `contact = 2`, because λ changes sign, and remains in that state if either $h > 0$ when λ changed sign, or $h < 0$ and $\dot{h} > 0$ when λ changed sign. After four bounces, the ball comes to rest, and the constraint $h \to 0$, and the Lagrange multiplier $\lambda > 0$.

The `algorithm` section computes the FSM. If its state is `contact == 1` then the Lagrange multiplier is active, and we include the stabilizing constraint equation with a stiff spring model of contact. If inactive, then we explicitly set `lambda = 0`, in order to balance the number of equations and states. We have included the flag `linFlag` in order to compute linearization when the constraint is active. It is necessary because Dymola will otherwise compute a mathematically incorrect linearization at the final time of simulation when the constraint is active, due to its numerical algorithm.

Figure 2 shows a Dymola simulation with a small positive value of damping. This is a typical situation in which this method is useful, although our typical uses have more damping than this. However, Figure 3 shows the situation when the damping is zero, ($\alpha_1 = 0$). Here we see the ball height increases over time, which means the model is adding energy to the system, which is not physical. This is for two reasons. First, the Lagrange multiplier, which is the reaction force due to collision, includes the steady-state gravity force mg when the ball is in contact,

$$\lambda(t) = (\alpha_0 q(t) + \alpha_2 q^3(t) + g)m.$$

If we were to implement a mass-spring type of force during collision, this term would be absent and the simulation would conserve energy within the solver tolerance. So, during each contact our method adds a little extra energy that is not physical. This is a clear disadvantage of the method. The second reason that the energy increases is that the constraint is active beyond the point in time that h crosses zero, by design of the FSM, which switches when λ crosses zero, not when h crosses zero. This is apparent in Figure 4. The reason the FSM is designed in this man-

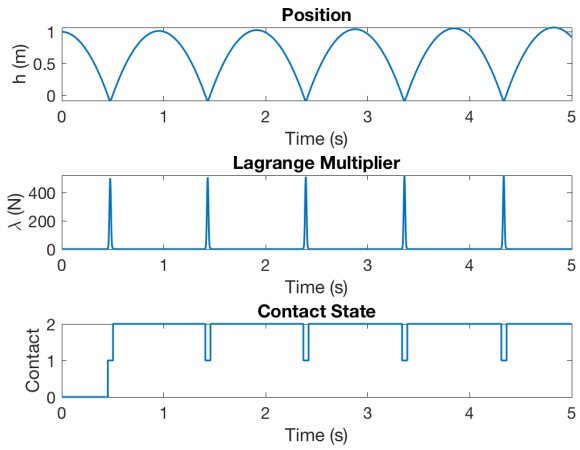

Figure 3. Bouncing ball simulation for stiff materials and zero damping ($\alpha_1 = 100$, $\alpha_1 = 0$, $\alpha_2 = 1e6$), showing the ball height growing over time.

ner is that the constraint equations cause $h \to 0$, and so small numerical errors in h when it is near zero but slightly positive would cause a switch from contact to non-contact, even if $\lambda > 0$. This would cause a kind of undesirable and actually unphysical chatter in the system simulation. We prefer to switch when the constraint force changes sign, so that objects in contact will remain in contact as long as their contact force is positive.

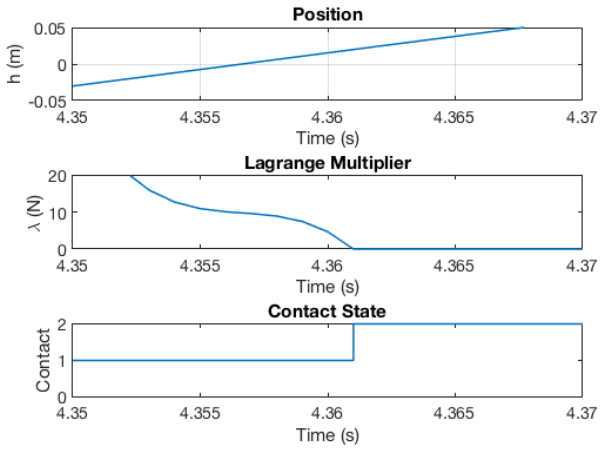

Figure 4. Close-up of the transition between contact and non-contact in Figure 3, which occurs when λ changes sign near $t = 4.262$s. Note that when h crosses zero, near $t = 4.356$s, $\lambda = 9.81$, so additional positive force is applied to the ball for $4.262 < t < 4.356$, adding energy to the ball.

We emphasize that the situation when $\alpha_1 = 0$ is not representative of a typical use of this method, and discussed here only to be clear about the method's limitations. Indeed, in this case, the constraint is not stabilized. Most real problems involving assembly have significant damping, and in these cases the energy increase due to the method is negligible.

Figure 5. Rolling ball maze toy (van Baar et al., 2019).

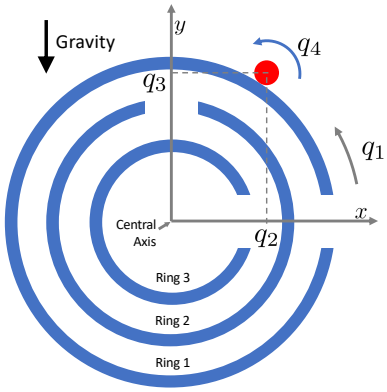

Figure 6. Ball maze diagram.

3 Ball Maze Example

A ball maze exhibits contact and non-contact physics that is effectively modeled using the proposed approach. Referring to Figure 5, the objective of the game is to manipulate the maze orientation in a way to cause the ball to roll into the center ring. The maze has been used to demonstrate learning-type control (van Baar et al., 2019), and typically the manipulation is "tip-tilt" in nature, causing the ball to roll due to gravity. The objective is challenging because after the ball passes through a gate, it collides with a maze ring and once in contact, the configuration is unstable, so the ball will roll one way or another, frustrating the player.

Here we will solve the maze using an alternative approach. Instead of tipping and tilting the maze, we stand it on end, so that the central axis (of symmetry) is orthogonal to the gravity vector, and we rotate the maze about the central axis, so that the maze itself has only one degree of freedom, as shown in Figure 6. The strategy is to use feedback to stabilize the ball when it is in contact with a ring, and then rotate the closed-loop maze system so that the ball falls through a sequence of gates, eventually reaching the center. Therefore we need a model that is appropriate for control system design, and also is appropriate for the simulation of the ball motion as it comes into and out of contact with the sequence of rings.

The ball maze is modeled as

$$M\ddot{q} + G = H^T(q,i)\lambda + Bu \tag{8a}$$
$$\ddot{h}(q,\dot{q},\lambda,i) + \alpha_1 \dot{h}(q,\dot{q},i) + \alpha_0 h(q,i) = 0 \tag{8b}$$

where $q \in \mathbb{R}^4$, q_1 is the angular displacement of the maze, q_2 and q_3 are the Cartesian coordinates of the ball center-of-mass, q_4 is the angular displacement of the ball relative to the base frame,

$$M = \text{diag}(J_m, m_b, m_b, J_b)\,, \tag{9a}$$
$$G = [0\ 0\ g m_b\ 0]^T, \tag{9b}$$
$$B = [1\ 0\ 0\ 0]^T, \tag{9c}$$
$$H(q,i) = \frac{\partial h(q,i)}{\partial q}\,, \tag{9d}$$

J_m and J_b are the rotational inertia of the maze and ball, respectively, m_b is the ball mass, g is the acceleration due to gravity, u is an input torque, and $\lambda \in \mathbb{R}^2$ is the Lagrange multiplier vector. When the ball is in contact with maze ring i of radius r_{mi}, $1 \le i \le 4$, the two holonomic constraints

$$h_1(q,i) = q_1^2 + q_2^2 + (r_{mi} + r_b)^2 \tag{10a}$$
$$h_2(q,i) = r_b(q_4 - q_{40}) + r_{mi}(q_1 - q_{10}) + r_{mi}\left(\text{atan}(q_2/q_3) - \text{atan}(q_{20}/q_{30})\right) \tag{10b}$$

are active so that λ_1 and λ_2 are time-varying algebraic states. In (10), h_1 is the distance constraint, h_2 is the rolling constraint, r_b is the ball radius, and $q_0 = [q_{10}\ q_{20}\ q_{30}\ q_{40}]^T$ is $q(t)$ at the time of collision $t = t_0$ with ring i. When the ball is not in contact with any rings, i.e., it is passing through a gate, then $\lambda_1 = \lambda_2 = 0$.

We design a stabilizing feedback controller for the case when the ball and ring i are in contact using the root locus method, assuming the ball position along the x-axis, $y_1 = q_2$, and the maze orientation, $y_2 = q_1$, are measured outputs, and splitting the input as $u = u_1 + u_2$. Model (8) and (10) is coded in Modelica, from which we compute a linearization at the open-loop unstable equilibria $q_2 = 0$, and a pole-zero plot of the system from u_1 to y_1. This shows four pole-zero cancellations at $s = -5$, corresponding to the stabilized constraint (8b) with $\alpha_1 = 10$ and $\alpha_0 = 25$, two pole-zero cancellations at the origin, one pole at $s = -1.1$, and one unstable pole at $s = 1.4$. This system can be stabilized using two lead-type compensators, one for each output. The first stabilizes the ball position $y_1 = q_2$, with zero at $s = -0.6$, pole at $s = -10$ and positive feedback gain, as shown in the root locus in Figure 7 (top). This moves the unstable pole into the left half plane, but leaves the two poles at the origin. After closing this loop, we next compute the pole-zero map for the system from u_2 to y_2. This shows the two poles at the origin and a non-minimum phase zero at $s = 0.65$. This can also be stabilized with a lead compensator with zero at $s = -0.05$, pole at $s = -5.0$, and gain $k_2 = 0.035$, as

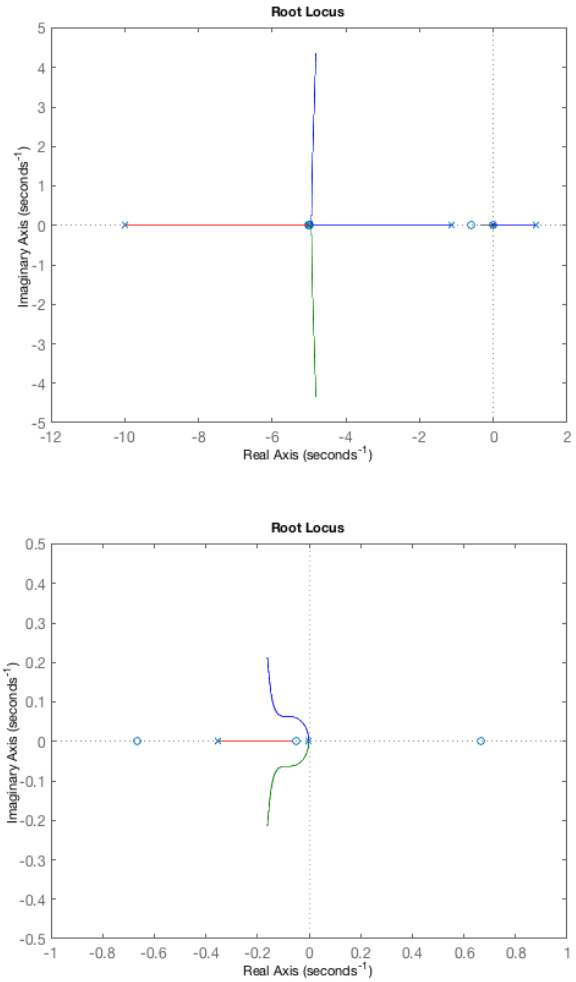

Figure 7. Root locus showing a lead compensator stabilizing the ball position $y_1 = q_2$ (top), and lead compensator stabilizing the maze rotation $y_2 = q_1$ (bottom).

shown by the root locus in Figure 7 (bottom). The second loop has an upper limit on its gain, due to the non-minimum phase zero, and has a slower response, while the first loop has a lower limit on its gain, due to the unstable pole, and has a faster response. The closed-loop system is realized in Modelica as shown in Figure 8. Note that a single set of controller gains is effective for any of the rings $1 \le i \le 4$, but this must be checked by computing a linearization for each ring radius.

To simulate the ball maze, we require a FSM to switch the Lagrange multipliers from active to inactive, similar to Figure 1. However for the ball maze, we have sufficient damping to prevent bouncing, and the system will move one-way through the maze, simplifying the logic. Essentially the constraints become active when the ball contacts a ring, i.e., when h_1 changes sign, for the sequence of rings (note that h_1 depends on ring radius i), and become inactive i.e., the ball falls through a gate, when the rotation of the maze moves the gate under the ball. We can define the

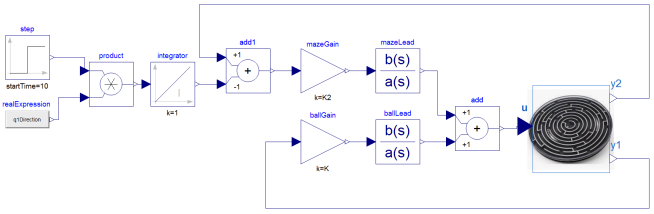

Figure 8. Feedback Controller for Ball Maze.

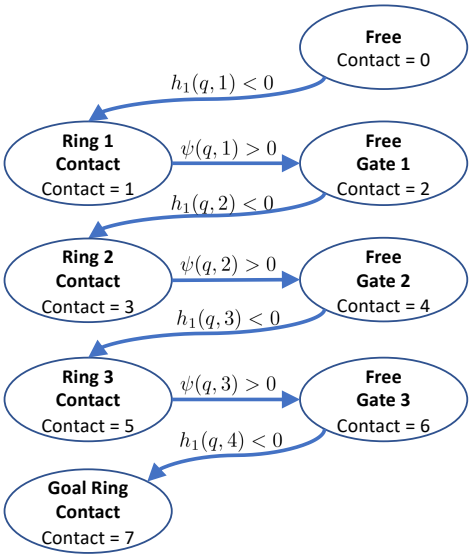

Figure 9. Ball maze FSM.

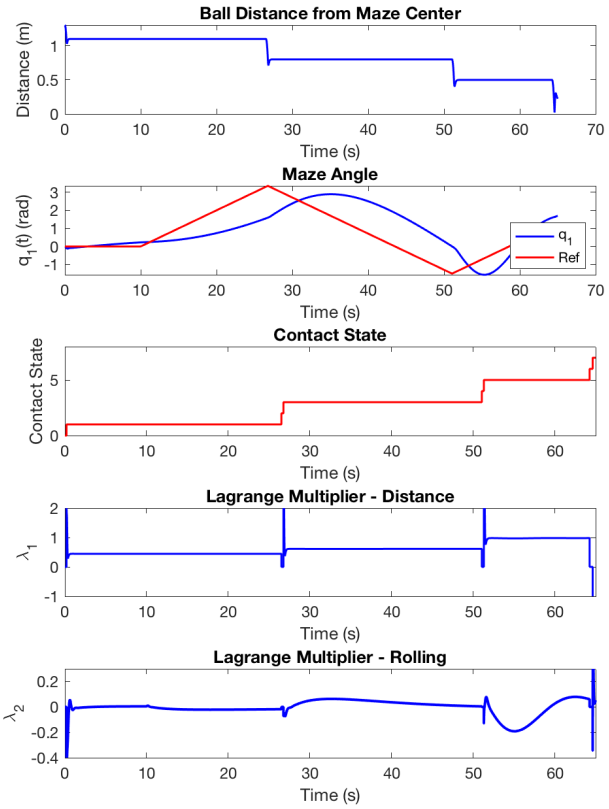

Figure 10. Ball maze simulation. Note the brevity of the free-motion states, contact $= 2,4,6$, when the ball passes through a gate, and when $\lambda = 0$.

location of the gates by defining

$$\psi(q,i) = \begin{cases} q_1 + r_{m1}\text{atan}(q_2/q_3) - \pi/2 & \text{for} \quad i = 1 \\ -q_1 - r_{m2}\text{atan}(q_2/q_3) & \text{for} \quad i = 2 \\ q_1 + r_{m3}\text{atan}(q_2/q_3) - \pi/2 & \text{for} \quad i = 3 \\ -q_1 - r_{m4}\text{atan}(q_2/q_3) & \text{for} \quad i = 4 \end{cases} \tag{11}$$

so that $\psi(q,i)$ changes sign as the ball moves "over" open gate i. Then the FSM is as shown in Figure 9, and is straightforward to implement as a Modelica algorithm.

Figure 10 shows results of a Dymola simulation of the closed-loop system. The ball is initialized in the free state, and falls toward the outer ring, making contact at about $t = 0.1s$. The maze turns counterclockwise beginning at $t = 10s$ and rotates until the ball falls through gate 1. As it falls through the gate, in the contact $= 2$ state, $\lambda = 0$ and the ball moves freely, until it collides with ring 2 at about $t = 27$s. The collision causes the large spike in λ, representing the elastic collision, after which the contact $= 3$ state is maintained as the maze is rotated clockwise until gate 2 is below the ball. Thus, as the maze rotates, the system switches between the constrained contact state and an unconstrained free state, although the number of dynamic states remains constant (8) throughout. The ball maze continues to rotate until the ball gets to the inner ring, at which point the controller is turned off.

Figure 11 shows a sequences of screen captures from a Dymola animation of the closed-loop system. Note that we show only three rings here for simplicity. Space constraints prohibit a full listing of the Modelica code, which is available from the conference website or by contacting the author directly. Note that it would be simpler to rotate the maze in a single direction. However, changing the direction of $\dot{q}_1$ allows for the maze to retain previously collected balls in the center. It also shows the slower and non-minimum phase response of the maze angle q_1 to the reference.

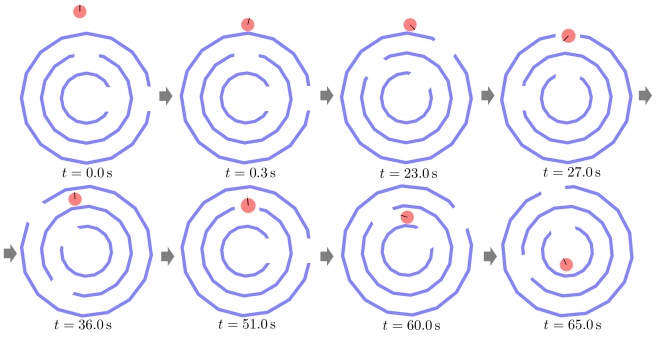

Figure 11. Sequence of ball maze configurations as the closed-loop system drives the ball toward the goal.

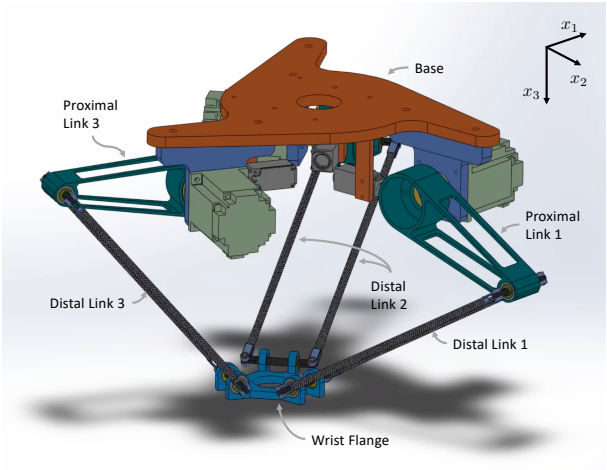

Figure 12. Delta robot.

4 Delta Robot Soft-Touch Control

The delta robot (Clavel, 1990) shown in Figure 12 consists of three (or more) identical under-actuated arms, arranged symmetrically about the x_3-axis (pointing down). Each arm consists of a proximal link, rigidly attached to the servomotor shaft at the base, and a pair of parallel distal links that are attached to the proximal link by universal joints. The six distal links are in turn attached to the wrist flange by universal joints, so that the two distal links associated with each arm remain parallel during operation. The configuration provides three translation DOFs of the wrist flange within the robot's reachable workspace, while the orientation of the wrist flange remains fixed. This beneficial feature of the delta robot decouples the translational kinematics and dynamics of the robot from the rotational kinematics and dynamics associated with a wrist that is mounted below the wrist flange (not shown in Figure 12). The servomotor angles are directly measured, while the universal joint angles are not measured, but can be computed from the servomotor angles via the forward kinematics.

A formulation and Modelica realization of the delta robot translational dynamics was derived previously (Bortoff, 2018, 2019) by defining the dynamics for each unconstrained arm, and then adding the holonomic coupling constraint representing the connections to the wrist flange. The resulting index-3 differential-algebraic equation (DAE) is stabilized using Baumgarte's method (Baumgarte, 1972, 1983), giving an index-1 DAE.

The objective here is to derive and simulate a soft-contact control algorithm for the delta robot. In this case the task object is a Lego brick on a surface, and the task is to pick up the brick with a gripper that is mounted to the wrist flange without sliding the brick along the surface, despite uncertainty in its location on the surface. Such a control algorithm would be used in practice to grasp very fragile objects. To accomplish this, the gripper frame is moved by the robot servos so that the left finger is close to the block surface, and then it approaches the block with

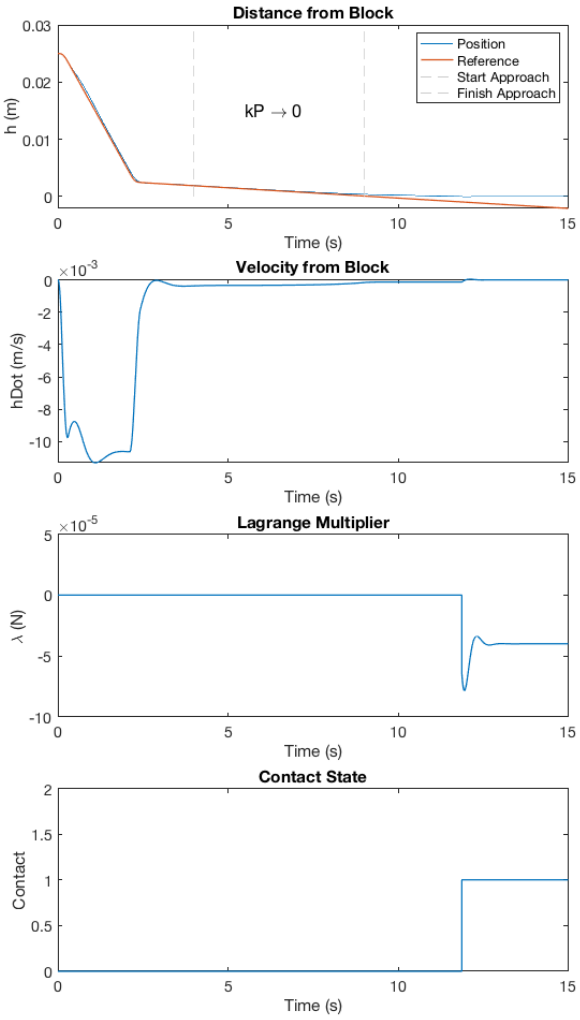

Figure 13. Simulation of soft collision and contact. The distance to the object (top), is commanded by a smooth reference to be within 5mm of the object in the first 2.5s. Then it approaches at low velocity of 0.2mm/s, and the position gain k_p is reduced continuously to zero until $t = 9$s. At this point the robot is in velocity control mode. Contact is made at $t = 12$s, and the force of impact peaks at 90mN, below what would cause the block to move. Contact is maintained with a force of 40mN, and there is no motion of the block or bouncing.

low velocity and low impedance so that it does not transfer energy sufficient to cause motion of the block. For this particular simulation, the gripper servo is not used.

Figure 14 is a block diagram in Dymola showing the feedback controller. The reference generator computes smooth reference trajectories for position, velocity and acceleration in task space within constraints on maximum values for velocity, acceleration and jerk, using cubic splines. In this specific case, the trajectory is such that the approach velocity is 0.2mm/s. The PDFF controller block is a digital PD controller with feedforward

$$u(k) = k_p(k)(r(k) - y(k)) + k_d(k)(\dot{r}(k) - \dot{y}(k)) + \ddot{r}(k) + f(k) \tag{12}$$

where $\dot{y}(k)$ is computed by filtering $y(k)$ to approximate the derivative, the feedback gains k_p and k_d are time-varying, set by a higher-level controller, and f is an ex-

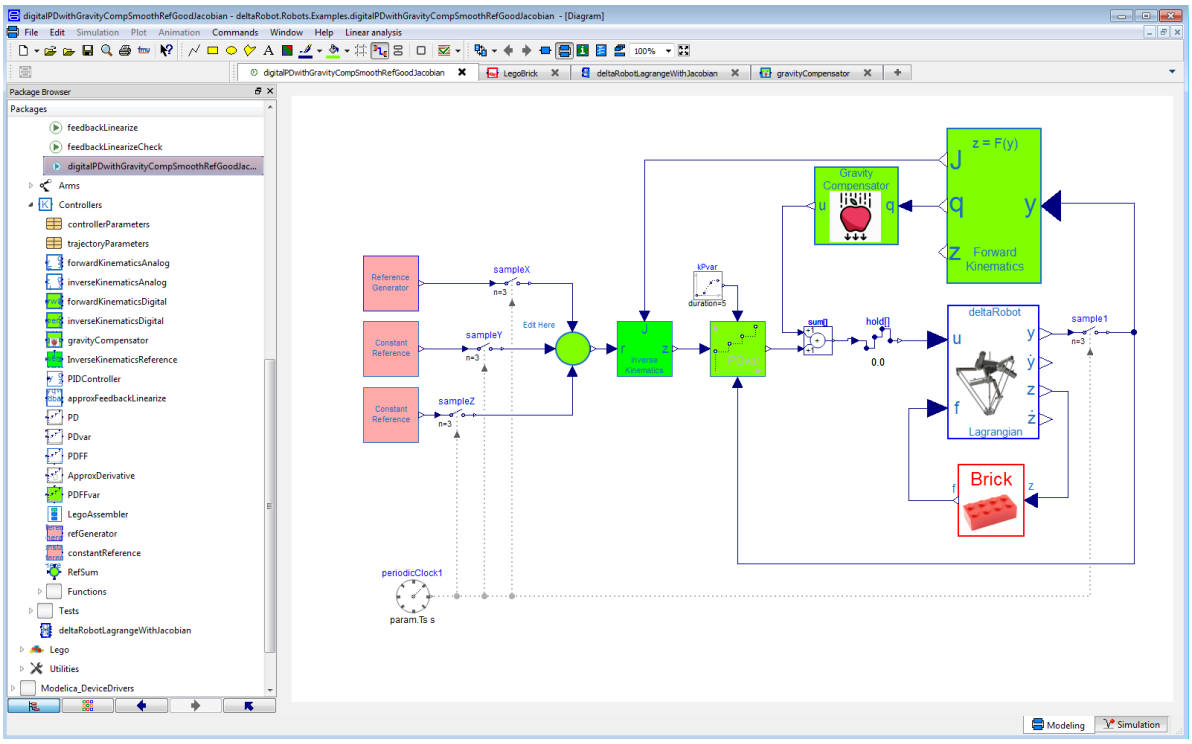

Figure 14. Feedback control model for soft contact. Blocks in green are discrete-time control blocks that use the Modelica Synchronous Library and compute forward and inverse kinematics, PD control with variable gains, and gravity torque compensation. The pink blocks at left compute smooth reference trajectories. The red Brick block computes the dynamics of a block in contact with a surface, which comes in contact with the delta robot end effector during simulation. The input f into the delta robot model is the force applied to the end effector by the brick, which is a Lagrange multiplier internal to the Brick model. Our deltaRobot library is open at left, showing some of the control system components.

ternal input from the left touch sensor, which is not used here. The Forward and Inverse Kinematics blocks, and the Gravity Compensator block, compute the forward and inverse kinematics, and also a torque to cancel the effect of gravity on the robot. These functions cannot be computed analytically for the delta robot, and use Newton's method to solve a set of implicit functions, described in (Bortoff, 2018).

To achieve soft contact, the k_p gain in the direction of travel is continuously reduced to zero as it approaches the brick, putting the robot effectively in velocity control mode and reducing its impedance at the moment of contact. Closed-loop stability is maintained if

$$0 \leq k_p(k) \leq k_d^2, \tag{13}$$

for fixed k_d, which follows from the Circle criteria (Vidyasagar, 1993). Importantly, there is no heuristic switch from a "position control mode" to a "velocity control mode." Rather it is a single controller that is continuously adjusted between these two extremes along the reference trajectory as the end effector approaches the task object. Further, there is no switch from "motion control mode" to a "force control mode" when contact is detected. In fact, the robot lacks a force - torque sensor. Rather, there is a single feedback controller that can achieve soft contact without any switch. This is important because we want to eliminate tuning and commissioning effort, and we desire a control architecture amenable to robustness analysis.

Figure 13 shows a simulation result of the end effector approaching and contacting the brick. The contact between the block and surface, and between the block and robot is modeled as in Section 2. In this simulation, a smooth trajectory commands the left finger to approach at low velocity, while the impedance is reduced by continuously until $k_p = 0$ before impact. From this point forward it is effectively in a velocity control mode. Contact is made, and the Lagrange multiplier becomes active at $t = 12$s. There is no bounce and the force imparted is sufficiently small to maintain the friction contact with the surface, so the block does not move. The small force is maintained after the contact.

5 Conclusions

We have presented a useful model of contact and collision intended to support development of model-based control design and analysis for robotic assembly. The advantages and disadvantages of the modeling approach were discussed. In particular, this method will be effective in situations involving low numbers of contacts, where it is

important to enforce penetration constraints after the collision transient occurs, and in cases where dynamic analysis is required, not just time-domain simulation. Three examples illustrate the method, with the ball maze providing a control design and simulation use case.

There are several important issues that are not addressed in this paper. First, friction at the contact is not included, although it would not be difficult to augment the approach to include it. Second, and more importantly, redundant constraints, where the Jacobian H is non full row rank, and therefore the Lagrange multipliers are not unique, is also not considered. Additional logic or a way to regularize the problem, i.e., by adding an additional constraint, might work. But despite these and other disadvantages, we intend to use this method to develop a range of control algorithms for robotic assembly, in addition to modeling other mechatronic problems in which contact and collisions are central to the problem.

References

Gianluca Bardaro, Luca Bascetta, Francesco Casella, and Matteo Matteucci. Using Modelica for advanced multi-body modelling in 3D graphical robotic simulators. In *Proceedings of the 12th International Modelica Conference*, pages 887–894, 2017.

J. W. Baumgarte. Stabilization of constraints and integrals of motion in dynamic systems. *Computer Methods in Applied Mechanics and Engineering*, 1:1–16, 1972.

J. W. Baumgarte. A new method of stabilization for holonomic constraints. *ASME Journal of Applied Mechanics*, 50:869–870, 1983.

Scott A. Bortoff. Object-oriented modeling and control of delta robots. In *IEEE Conference on Control Technology and Applications*, pages 251–258, 2018.

Scott A. Bortoff. Using Baumgarte's method for index reduction in Modelica. In *Proceedings of the 13th International Modelica Conference*, pages 333–342, March 2019.

R. Clavel. Device for the movement and positioning of an element in space. U.S. Patent 4, 976, 582, Dec. 11 1990.

Erwin Coumans. Bullet 2.83 physics SDK manual. https://github.com/bulletphysics/bullet3/tree/master/docs, 2015.

Vadim Engelson. *Integration of Collision Detection with the Multibody System Library in Modelica*. PhD thesis, Link oping University, 2000.

Kenny Erleben, Jon Sporring, Knud Henriksen, and Henrik Dohlmann. *Physics-Based Animaation*. Charles River Media, 2005.

Elmer Gilbert, Daniel W. Johnson, and S. Sathiya Keerthi. A fast procedure for compting the distance bewteen complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, 1988.

Andreas Hofmann, Lars Mikelsons, Ines Gubsch, and Christian Schubert. Simulating collisions within the Modelica multibody library. In *Proceedings of the 10th International Modelica Conference*, pages 949–957, 2011a.

Andreas Hofmann, Lars Mikelsons, Ines Gubsch, and Christian Schubert. Modelica idealized contact library. https://github.com/modelica-3rdparty/IdealizedContact, 2011b.

Martin Otter, Hilding Elmqvist, and José Díaz López. Collision handling for the Modelica MultiBody library. In *Proceedings of the 4th International Modelica Conference*, pages 45–53, March 2005.

Bernhard Thiele, Thomas Beutlich, Volker Waurich, Martin Sjölund, and Tobias Belmann. Towards a standard-comform, platform-generic and feature-rich Modelica device drivers library. In *Proceedings of the 12th International Modelica Conference*, pages 713–723, 2017.

Jeroen van Baar, Alan Sullivan, Radu Cordorel, Davesh Jha, Diego Romeres, and Daniel Nikovski. Sim-to-real transfer learning using robustified controllers in robotic tasks involving complex dynamics. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2019.

M. Vidyasagar. *Nonlinear Systems Analysis: Second Edition*. Prentice-Hall, 1993.

Miomir Vukobratovic, Veljko Potkonjak, and Vladimir Matijevic. *Dynamics of Robots with Contact Tasks*. Kluwer, 2003.