Modelica-Based Control of A Delta Robot

Bortoff, Scott A.; Okasha, Ahmed

TR2020-154 December 09, 2020

Abstract

In this paper we derive a dynamic model of the delta robot and two formulations of the manipulator Jacobian that comprise a system of singularity-free, index-one differential algebraic equations that is well-suited for model-based control design and computer simulation. One of the Jacobians is intended for time-domain simulation, while the other is for use in discretetime control algorithms. The model is well posed and numerically well-conditioned throughout the workspace, including at kinematic singularities. We use the model to derive an approximate feedback linearizing control algorithm that can be used for both trajectory tracking and impedance control, enabling some assembly tasks involving contact and collisions. The model and control algorithms are realized in the open-source Modelica language, and a Modelica-based software architecture is described that allows for a seamless development process from mathematical derivation of control algorithms, to desktop simulation, and finally to laboratory-scale experimental testing without the need to recode any aspect of the control algorithm. Simulation and experimental results are provided.

ASME Dynamic Systems and Control Conference

Proceedings of the ASME 2020 Dynamic Systems and Control Conference
DSCC 2020
October 4-7, 2020, Pittsburgh, USA

DSCC2020

MODELICA-BASED CONTROL OF A DELTA ROBOT

Ahmed Okasha[*]
Department of Mechanical Engineering
Michigan State University
East Lansing, MI 48824, USA
Email: aokasha@msu.edu

Scott A. Bortoff [†]
Mitsubishi Electric Research Laboratories
Cambridge, MA 02139, USA
Email: bortoff@merl.com

ABSTRACT

In this paper we derive a dynamic model of the delta robot and two formulations of the manipulator Jacobian that comprise a system of singularity-free, index-one differential algebraic equations that is well-suited for model-based control design and computer simulation. One of the Jacobians is intended for time-domain simulation, while the other is for use in discrete-time control algorithms. The model is well-posed and numerically well-conditioned throughout the workspace, including at kinematic singularities. We use the model to derive an approximate feedback linearizing control algorithm that can be used for both trajectory tracking and impedance control, enabling some assembly tasks involving contact and collisions. The model and control algorithms are realized in the open-source Modelica language, and a Modelica-based software architecture is described that allows for a seamless development process from mathematical derivation of control algorithms, to desktop simulation, and finally to laboratory-scale experimental testing without the need to recode any aspect of the control algorithm. Simulation and experimental results are provided.

INTRODUCTION

In industrial applications, delta robots [1] are used primarily for light-duty, pick-and-place operations because of their relatively low cost and high speed. Control for these applications generally requires no more than the inverse kinematics and high-gain PID control. But their low mass and high mechanical stiffness make them an attractive platform for precise robotic assembly applications. These applications demand more sophisticated control algorithms, such as programmable impedance control for soft collisions and for contact force control.

To derive new control algorithms for robust delta robot assembly, it is important to construct and use system-level dynamic models of the complete robotic manipulator, the assembly task (including appropriate representations of object contact and collision) and also the control algorithms themselves. Our long-term objectives are (1) to invent new control algorithms 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.

Modelica[1] is an open-source computer language used to model complex, heterogeneous, multi-physical systems and includes synchronous language features to precisely and correctly define and synchronize sampled-data systems, making it an excellent platform for such research and development. Modelica libraries serve as a repository that archive new algorithms and models in a high-level language, and Modelica models can be used for desktop simulation, model-based control design, and even real-time laboratory scale experimental testing without having to recode any aspect of the control algorithm, thus supporting all three of our objectives.

[*]This work was done during an internship at MERL
[†]Address all correspondence to this author

[1]https://www.modelica.org/

In this paper, we extend our earlier work on object-oriented modeling and control of delta robots [2,3], deriving two formulations of the manipulator Jacobian that are useful for time-domain simulation and in discrete-time control algorithms. We derive an approximate feedback linearizing control algorithm, which is novel and non-trivial due to the closed-chain kinematics, and also describe our developing Modelica libraries that are used in a system architecture that allows for both simulation of new control algorithms and also real-time experimental testing of those algorithms using our laboratory-scale delta robot. Aside from proprietary API interface for our motion controller, the software is entirely open-source, making use of public domain Modelica libraries that enable real-time simulation and also interface with external hardware via device drivers.

This paper is organized as follows. After reviewing the dynamic model previously published, we derive the Jacobian formulations and explain their use. We use the model to derive the approximate feedback linearizing control algorithm, and also a simple gravity compensating feedback. We then describe our software architecture and delta robot hardware, and provide an example of a nonlinear feedback control in real-time simulation and also experiment.

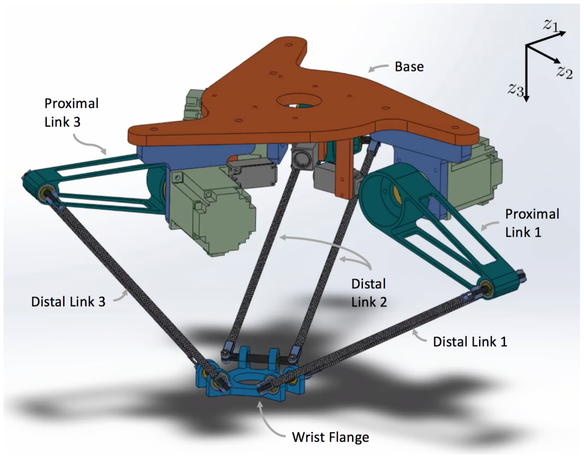

FIGURE 1. DELTA ROBOT.

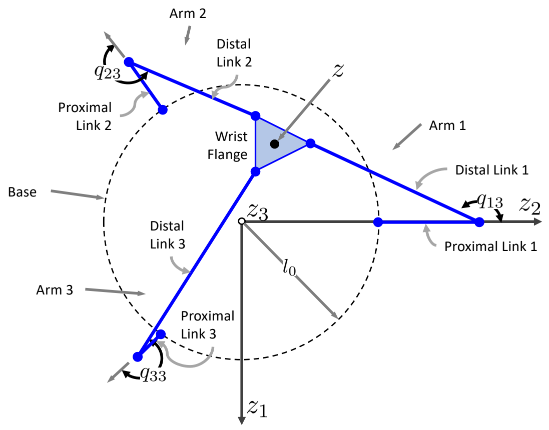

FIGURE 2. DELTA ROBOT COORDINATES, LOOKING UP.

DYNAMICAL MODEL

Referring to Figure 1, the delta robot [1] consists of three (or more) identical, under-actuated arms, arranged symmetrically about the z_3-axis (pointing down). Each arm has a proximal link, attached to the servomotor shaft at the base, and a pair of parallel distal links that are connected to the proximal link by universal joints. The six distal links are connected to the wrist flange by universal joints, so that the two distal links associated with each arm remain parallel. The configuration provides three translation degrees of freedom of the wrist flange, while the orientation of the wrist flange is invariant. This feature decouples the translational and rotational kinematics and dynamics, simplifying control. The servomotor angles are measured, but the universal joint angles are not.

Translational Kinematics and Dynamics

The delta robot is a complex[2] closed kinematic chain [4]. It is not possible to compute an analytic formula for the forward kinematics (the function from measured servomotor angles to the location of the wrist flange), although the inverse kinematics may be computed and expressed in closed-form. This makes the derivation of the dynamics more complex than for serial link manipulators.

One formulation of the delta robot dynamics first defines the dynamics for each unconstrained arm, and then adds the holonomic coupling constraint that represents the connections to the wrist flange [2, 3]. The resulting index-3 differential algebraic equation (DAE) is stabilized using Baumgarte's method [5, 6], giving an index-1 DAE. The DAE has 18 differential equations and variables, and six algebraic equations and variables, but the solution evolves on a six-dimensional, invariant, zero-dynamics manifold, and is mathematically equivalent to the solution of conventional dynamic equations written in six generalized coordinates [2]. This formulation has several advantages: It is a single set of singularity-free equations, despite the robot's kinematic singularities, it is computationally efficient, transparent (easy to verify), and useful for derivation of model-based control algorithms.

Summarizing the derivation here for completeness, let $q_i \in \mathbb{R}^3$ denote the joint angles of arm i, ordered so that q_{i1}, is the ith servomotor angle, and the remaining two angles correspond to the universal joints, for $1 \leq i \leq 3$, as shown in Figure 2. (See [2] for details, especially the definitions of q_{ij}, $1 \leq i, j \leq 3$, that

[2]Meaning one link has a degree of connectivity ≥ 3.

TABLE 1. KINEMATIC PARAMETER DEFINITIONS.

Symbol	Description (Units)	Experimental Values
l_0	Base radius (m)	0.165 m
l_1	Length of proximal link (m)	0.2 m
l_2	Length of distal link (m)	0.4 m
l_3	Width of wrist flange (m)	0.0562 m

place the universal joint's kinematic singularity outside of the robot's work volume.) Then the position of the geometric center of the wrist flange relative to the base frame, denoted $z \in \mathbb{R}^3$, is expressed in terms of arm 1 coordinates as

$$z = \psi(q_1) = \begin{bmatrix} l_2 \sin(q_{12}) \sin(q_{13}) \\ l_0 - l_3 + l_1 \cos(q_{11}) + l_2 \cos(q_{12}) \\ l_1 \sin(q_{11}) + l_2 \sin(q_{12}) \cos(q_{13}), \end{bmatrix}, \tag{1}$$

where the parameters are defined in Table 1. Similarly, we may also write

$$z = R_2 \psi(q_2), \text{ and} \tag{2a}$$

$$z = R_3 \psi(q_3), \tag{2b}$$

to express the kinematic constraint that z is just as well a function of arm 2 and 3 coordinates, respectively, where $R_2 = R_z(-2\pi/3)$, $R_3 = R_z(2\pi/3)$ and $R_z(\theta)$ is the standard rotation matrix about the "z"-axis by an angle θ.

Defining $q = [q_1^T \; q_2^T \; q_3^T]^T \in \mathbb{R}^9$ and $v = \dot{q}$, the translational dynamics can be written as the index 1 DAE

$$\dot{q} = v \tag{3a}$$

$$M(q)\dot{v} + C(q,v) + D(v) + G(q) = H^T(q)\lambda + B(u + \tau_u) + \tau_v \tag{3b}$$

$$\ddot{h}(q,v,\dot{v}) + \alpha_1 \dot{h}(q,v) + \alpha_0 h(q) = 0 \tag{3c}$$

where the constraint $h(q) : \mathbb{R}^9 \to \mathbb{R}^6$ follows from (1)-(2),

$$h(q) = \begin{bmatrix} \psi(q_1) - R_2 \psi(q_2) \\ \psi(q_1) - R_3 \psi(q_3) \end{bmatrix} = 0, \tag{4}$$

$H(q)$ is the Jacobian of $h(q)$, $\lambda \in \mathbb{R}^6$ is the Lagrange multiplier, $\tau_u \in \mathbb{R}^3$ is the vector of *matched* disturbance torques, $\tau_v \in \mathbb{R}^9$ is the vector of *virtual* disturbance torques, $u \in \mathbb{R}^3$ is the control input torque vector, positive constants α_0 and α_1 are such that $s^2 + \alpha_1 s + \alpha_0$ is a Hurtwitz (stable) polynomial, M, C, D, G and B are the inertia matrix, Coriolis and centripetal torque, damping, gravity, and input vectors, respectively (see [2] for details), and

$$y = [q_{11} \; q_{21} \; q_{31}]^T \tag{5}$$

is the vector of measured servomotor angles. The distinction between τ_u and τ_v is explained in the next subsections.

Forward Kinematics

The forward kinematics can be computed algorithmically from (4). Defining

$$x = [q_{12} \; q_{13} \; q_{22} \; q_{23} \; q_{32} \; q_{33}]^T, \tag{6}$$

whose elements are the unmeasured joint angles, rewrite (4) by reordering the arguments as

$$h(x,y) = 0. \tag{7}$$

For a given y, the solution x to these six nonlinear equations can be computed iteratively using Newton's method,

$$\frac{\partial h}{\partial x}(x_k, y) \cdot (x_{k+1} - x_k) = -h(x_k, y), \tag{8}$$

which converges locally and quadratically assuming the robot is not near a kinematic singularity (which implies $\frac{\partial h}{\partial x}$ is nonsingular). Then (1) is used to compute z. In practice, (8) may be used in a real-time control algorithm that samples y at a sufficiently high frequency so that it converges to seven digits of precision in 2-3 iterations.

Inverse Kinematics

The inverse kinematics, i.e., the function from z to y (and q) can be computed in closed form by solving (1)-(2) for the three elements of q_1, and efficient formulations have been reported [7, 8]. First solve the z_1 equation for q_{12}, giving

$$q_{12} = \pi - \arcsin\left(z_1 / l_2 \sin q_{13}\right), \tag{9}$$

where the π is included because q_{12} is defined relative to the horizontal plane, so $\pi/2 < q_{12} < \pi$. Next, substitute (9) into the z_3 equation and solve for q_{11}, which is also defined relative to the horizontal plane and is assumed to satisfy $-\pi/2 < q_{11} < \pi/2$ to avoid kinematic singularities, giving, after simplification

$$q_{11} = \arcsin\left(\frac{z_3 - z_1 \cot(q_{13})}{l_1}\right). \tag{10}$$

Finally, substituting (9) and (10) into the z_3 equation gives

$$\sqrt{l_1^2 - (z_3 - z_1 \cot(q_{13}))^2} - \sqrt{l_2^2 - z_1^2 \csc(q_{13})^2} + l_0 - l_3 - z_2 = 0, \tag{11}$$

which can be solved for q_{13} with the aid of symbolic computing software such as Mathematica. (The solution is omitted for space reasons.) Solutions to (2) are similar.

Manipulator Jacobians

The manipulator Jacobian is the map between end effector velocity / force and joint angular velocity / torque, respectively, and is important for (a) modeling and simulating the effect of force disturbances on the end effector, and also for (b) control purposes, such as computing reference joint velocities from reference velocities of the end effector. For serial link robots, it is the mathematical Jacobian of the forward kinematics. For the delta robot, this is nontrivial because the forward kinematics are not expressible analytically. In fact, the two uses for the Jacobian described above require two different expressions for the Jacobian.

First, in order to model and simulate the effect of force disturbances, we compute the Jacobian that maps an external disturbance force $f \in \mathbb{R}^3$ that is applied to the wrist flange to the vector of virtual torques τ_v that is applied to *all* of the joints. Summing equations (1)-(2), and normalizing by $1/3$, we can write

$$z = (\psi(q_1) + R_2\psi(q_2) + R_3\psi(q_3))/3. \tag{12}$$

Differentiating gives the 3×9 *virtual* Jacobian

$$J_v(q) = \frac{1}{3}\left[\frac{\partial\psi}{\partial q}(q_1) \quad R_2\frac{\partial\psi}{\partial q}(q_2) \quad R_3\frac{\partial\psi}{\partial q}(q_3)\right], \tag{13}$$

which we use to compute the *virtual* torque vector τ_v in (3b),

$$\tau_v = J_v^T(q)\cdot f. \tag{14}$$

Equations (1) - (5), (13) and (14) comprise a smooth, singularity-free, computationally efficient index-1 DAE model that is useful for simulation and model-based control design. Importantly, J_v is non-singular, even at manipulator kinematic singularities, as we show below.

A second formulation of the Jacobian is needed for control purposes: The map between end effector forces / velocities and *servomotor* torques / velocities. We denote it the *control* Jacobian, J_c. Because it depends on the forward kinematics, it is computed algorithmically, as follows. By the Implicit Function Theorem, there exists $g: \mathbb{R}^3 \to \mathbb{R}^6$ such that

$$h(g(y), y) = 0, \tag{15}$$

so that we may write $x = g(y)$, if $\frac{\partial h}{\partial x}$ is nonsingular in a neighborhood of (x, y). Define

$$z = \Psi(x, y) = \psi(y_1, x_1, x_2). \tag{16}$$

Then the 3×3 control Jacobian is

$$J_c = \frac{\partial z}{\partial y} = \frac{\partial \Psi}{\partial x}\cdot\frac{\partial g}{\partial y} + \frac{\partial \Psi}{\partial y}. \tag{17}$$

Each of the terms on the right-hand side of (17) is computed as follows. First, differentiating (15) and rearranging,

$$\frac{\partial g}{\partial y} = -\left(\frac{\partial h}{\partial x}\right)^{-1}\cdot\frac{\partial h}{\partial y}, \tag{18}$$

where $(\frac{\partial h}{\partial x})^{-1}$ is computed in the forward kinematics (8) and $\frac{\partial h}{\partial y}$ is computed by differentiating h, rewritten in (x, y) coordinates,

$$h(x,y) = \begin{bmatrix} \psi(y_1, x_1, x_2) - R_2\psi(y_2, x_3, x_4) \\ \psi(y_1, x_1, x_2) - R_3\psi(y_3, x_5, x_6) \end{bmatrix}. \tag{19}$$

The other two terms in (17) are computed by differentiating (1), giving

$$J_c = \begin{bmatrix} \left(\frac{\partial\Psi}{\partial y_1} + \frac{\partial\Psi}{\partial x_1}\frac{\partial g_1}{\partial y_1} + \frac{\partial\Psi}{\partial x_2}\frac{\partial g_2}{\partial y_1}\right)^T \\ \left(\frac{\partial\Psi}{\partial x_1}\frac{\partial g_1}{\partial y_2} + \frac{\partial\Psi}{\partial x_2}\frac{\partial g_2}{\partial y_2}\right)^T \\ \left(\frac{\partial\Psi}{\partial x_1}\frac{\partial g_1}{\partial y_3} + \frac{\partial\Psi}{\partial x_2}\frac{\partial g_2}{\partial y_3}\right)^T \end{bmatrix}^T, \tag{20}$$

where $\frac{\partial\Psi}{\partial y_1}$, $\frac{\partial\Psi}{\partial x_1}$ and $\frac{\partial\Psi}{\partial x_2}$ are all 3×1 column vectors, and the transposes are used to express J_c compactly. Note there are several equivalent formulations that use (2) instead or in combination with (16).

It is important to emphasize that $\tau_u = J_c^T \cdot f$ should never be used instead of (14) in (3b) for time-domain simulation to compute the effects of the disturbance f, for two reasons. First, J_c is undefined at kinematic singularities, as we show in the next subsection. It is important that the dynamic model remain well-defined and numerically well-conditioned, even at kinematic singularities, since the physics certainly is. Second, using J_c in this manner would cause the dynamic model to include an algorithm, since it is computed numerically by iteration, making the DAE *non-smooth* because of the termination condition. This can wreck havoc on numerical solvers and is generally considered poor modeling practice [9, 10]. Besides, J_v is more numerically efficient.

Simulation through a Kinematic Singularity

The model (1) - (5), (13) and (14) has been realized in the Modelica language for purposes of simulation, model-based control design [2] and now also experimental validation. We describe the latter in the next section, but first we show an interesting simulation of the robot at one of its kinematic singularities.

If the three servomotor angles assume a constant value of

$$y_i = \operatorname{acos}\left(\frac{l_3 - l_0}{l_1}\right)$$

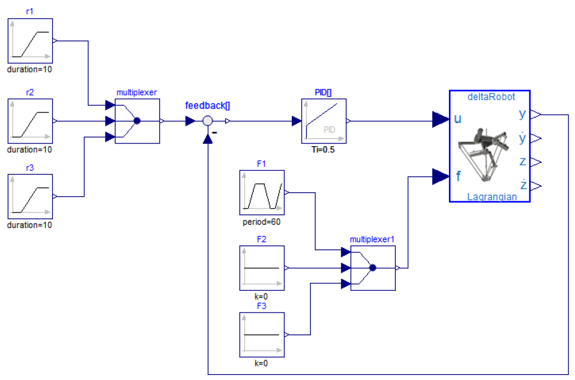

FIGURE 3. DELTA ROBOT WITH PID FEEDBACK.

for $1 \le i \le 3$, then the six distal links are all parallel to one-another, and the wrist flange is free to swing as a spherical pendulum. The robot is at a kinematic singularity, the forward kinematics are not defined, $\frac{\partial h}{\partial x}$ is singular (actually rank four), and therefore J_c is undefined. However, the DAE model (1) - (5), (13) and (14) is perfectly well-defined because $H(q)$ retains full rank and (1) is well-defined for all q_i away from the universal joint singularity at $q_{i3} = \pi/2$, $1 \le i \le 3$, which is not mechanically reachable anyway.

A Modelica realization of the DAE model with a PID feedback controller and references for the servomotor angles that drive the system to this configuration is shown in Figure 3, and simulation results are shown in Figure 4. The PID feedback drives the servomotor angles to the singularity at $y_i \approx 2.15$ rad, $1 \le i \le 3$, by $t = 20$ s (top), an impulsive force applied to the wrist flange in the z_1 direction at $t = 25$ s (second), resulting in free pendulum motion of the wrist flange, plotted in the z-coordinates (third), with no displacement in the z_2-direction. Note the regulation of y_1 by the PID after the impulse, and the Lagrange multipliers (bottom), which are 0 at the singularity in equilibrium. This shows the advantages of modeling the robot as a set of 24 singularity-free index-1 DAEs, namely that the modeling equations are well-posed throughout the possible robot configuration space, even though at this particular configuration, it becomes uncontrollable and practically useless as a manipulator. On the other hand, if the model was expressed as a set of six DAEs or ODEs written using the servomotor angles and velocities as generalized coordinates, which is possible locally, then the simulation would have failed as it approached the singularity.

CONTROL

Our interest is using the delta robot for assembly operations, which manifest frequent collisions and contact. High-gain position feedback control is not appropriate for this application. Here we derive an approximate feedback linearization algorithm that can be used as a basis for programmable impedance control.

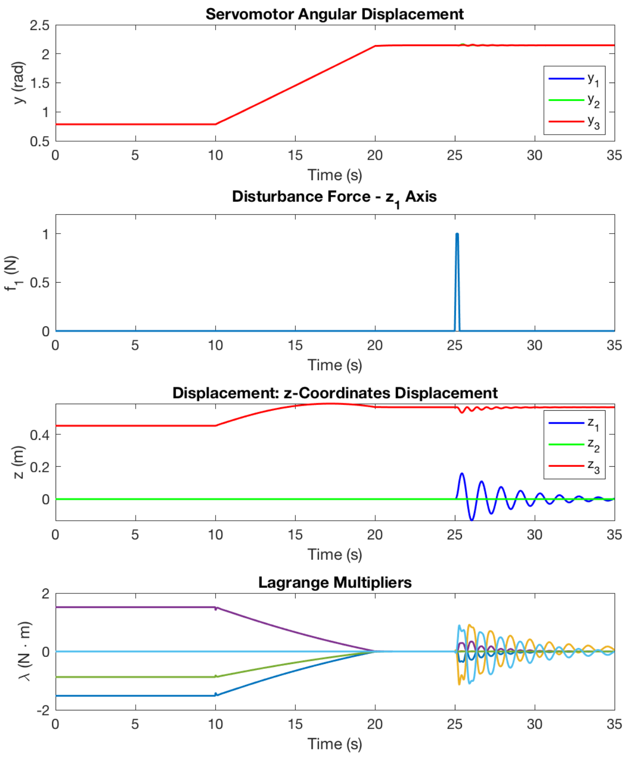

FIGURE 4. SIMULATION AT KINEMATIC SINGULARITY.

Gravity Compensation

The control system measures only the servomotor angles y. From these measurements, the forward kinematic equations are solved, providing q, z and J_c. Referring to (3a), we compute a value for τ_c that cancels the effect of $G(q)$, by solving the six-dimensional set of nonlinear equations

$$[B \quad H^T(q)] \begin{bmatrix} \tau_u \\ \lambda \end{bmatrix} = G(q) \tag{21}$$

for τ_u and λ. (Note that the controller does not measure λ.) This is well-posed throughout the manipulator workspace, and is solved by Newton's method in practice.

Approximate Feedback Linearization

Applying the value τ_c computed in (21) to (3a), results in the gravity-compensated model

$$M(q)\dot{v} + C(q,v) + D(v) = Bu. \tag{22}$$

Note that M is 9×9 and B is rank 3. Using the definition of x in (6), and rearranging the order of equations in (22) so that the measured joint angles y appear as the top block gives

$$\begin{bmatrix} \bar{M}_{11}(q) & \bar{M}_{12}(q) \\ \bar{M}_{21}(q) & \bar{M}_{22}(q) \end{bmatrix} \begin{bmatrix} \ddot{y} \\ \ddot{x} \end{bmatrix} + \begin{bmatrix} \bar{C}_1(q,v) \\ \bar{C}_2(q,v) \end{bmatrix} + \begin{bmatrix} \bar{D}_1(v) \\ \bar{D}_2(v) \end{bmatrix} = \begin{bmatrix} u \\ 0 \end{bmatrix}, \quad (23)$$

where the overbar denotes the reordering. Next we write $\ddot{x}$ in terms of $\ddot{y}$ by differentiating the holonomic constraint (7) twice, giving

$$\frac{\partial h}{\partial y}\ddot{y} + \frac{\partial h}{\partial x}\ddot{x} + \dot{y}^T \frac{\partial^2 h}{\partial y^2}\dot{y} + \dot{x}^T \frac{\partial^2 h}{\partial x^2}\dot{x} = 0. \quad (24)$$

Solving for $\ddot{x}$, substituting into (23) and ignoring higher-order terms in $\dot{x}$ and $\dot{y}$ (v, reordered), gives

$$\bar{M}_y \cdot \ddot{y} = u, \quad (25)$$

where the 3×3 inertia matrix

$$\bar{M}_y = \bar{M}_{11} - \bar{M}_{12} \cdot \frac{\partial h}{\partial x}^{-1} \cdot \frac{\partial h}{\partial y}. \quad (26)$$

The approximate feedback linearizing control is then

$$u = \bar{M}_y \left(k_p(r-y) + k_d(\dot{r} - \dot{y}) + \ddot{r} \right), \quad (27)$$

which, applied to (22) gives

$$\ddot{e} + k_d \dot{e} + k_p e = 0, \quad (28)$$

(ignoring higher-order terms), where $e = y - r$, r, $\dot{r}$ and $\ddot{r}$ are the reference trajectory and its first two derivatives, and k_p and k_d are position and derivative gains, respectively. Note that the Jacobians of h (and the inverse) are computed in the forward kinematic algorithm (18), so few additional operations are required to compute $\bar{M}_y$. Also note that (26) is singular at kinematic singularities, which is to be expected.

MODELICA AS DEVELOPMENT PLATFORM

Modelica is an open-source computer language used to model complex, heterogeneous, multi-physical systems [12–14]. It has a number of features that make it advantageous for modeling the multi-body physics of the delta robot [3], its servomotors, inverters, controller – in both form and function – and even the robot manipulator's task itself [15]. For one, it is a declarative language that represents physical equations as acausal statements. This means that an equals sign in the language means the same thing as it does mathematically, unlike almost all other computer languages, where it means assignment. This fact alone allows for clear transcription of physical equations, without the need for manual, often error-prone causalization that is necessary in signal-flow type languages such as Matlab Simulink. Fundamentally it is based upon a hybrid DAE model of computation [16], so high-index DAEs may be correctly represented, reduced in index, and solved. This is important for physical systems such as the delta robot, as well as control systems. Furthermore, the hybrid support allows for correct representation and solution of discrete-event systems and their interaction with continuous-time physics. Since Modelica 3.3 was introduced in 2012, the language includes synchronous language features to precisely define and synchronize sampled-data systems, including periodic, non-periodic and event-based clocks [17]. It is object-oriented for structure, including inheritance, redeclaration, and encapsulation, enabling organization of component-oriented libraries. Indeed a large number of commercial and open-source libraries have been created to model diverse physical processes, synchronous systems [18], and even to interface models to real-time input and output [19], allowing for experimental testing of control systems.

Taken together, these features make Modelica an excellent platform for research and development of next generation control theory and algorithms in general, but especially for the delta robot. At MERL, we have developed three Modelica libraries (called *packages*) to represent (a) the delta robot dynamics, (b) delta robot control algorithms, and (c) interface to real-time software for experimental testing. A fourth is in development to model contact and collisions of task space objects [15]. Our system architecture is shown in Figure 5. The robot dynamics library is described in [2], and includes both Lagrangian and Hamiltonian formulations in addition to the virtual Jacobian J_v described earlier. This library includes continuous - time dynamics of the physical system exclusively. The control library includes a growing number of algorithms, with some implemented in continuous-time, such as the PID used in Figure 3 (although this particular component is taken from the Modelica Standard Library), and others implemented in discrete-time using the Modelica Synchronous Library [18]. These include the usual PD and PID compensators for motion control, the forward and inverse kinematics, control Jacobian J_c, gravity compensation and approximate feedback linearization algorithms described earlier, and also new force and impedance control algorithms (beyond the scope of this paper).

The Modelica libraries serve several purposes. First, they are an archive of research results, serving as a form of documentation since the language itself is very readable, and also includes annotations that support diagrams, description and even animation. Second, they can of course be used for desktop simulation. For this a Modelica compiler is required, and there are several proprietary and open source alternatives. We use both Dy-

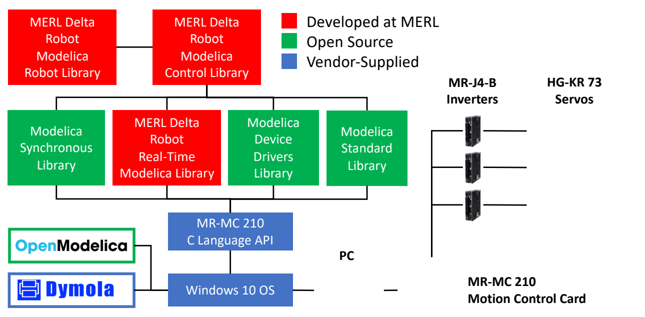
MERL Delta Robot Modelica Robot Library
MERL Delta Robot Modelica Control Library
Developed at MERL
Open Source
Vendor-Supplied
MR-J4-B Inverters
HG-KR 73 Servos
Modelica Synchronous Library
MERL Delta Robot Real-Time Modelica Library
Modelica Device Drivers Library
Modelica Standard Library
MR-MC 210 C Language API
OpenModelica
Dymola
Windows 10 OS
PC
MR-MC 210 Motion Control Card

FIGURE 5. DEVELOPMENT SYSTEM ARCHITECTURE.

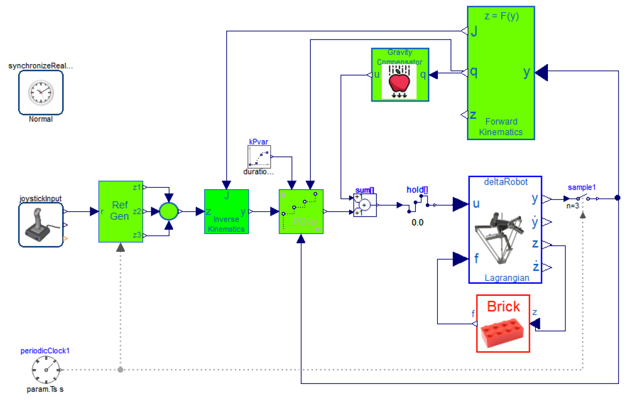

FIGURE 6. REAL-TIME CONTROL SIMULATION.

mola and OpenModelica[3]. Third, our system architecture allows for immediate experimental testing of control algorithms on our delta robot manipulator using the Delta Robot Real-Time Modelica Library. In fact, the software and hardware architecture is designed so that any discrete-time control algorithm in the library is simply disconnected from the continuous-time delta robot dynamic model, and is connected to the manipulator via custom interface code. We need only recompile and experiments may be conducted, with the real-time controller now running on the PC. For this we make use of the Modelica Device Drivers Library, which supports sufficiently fast and accurate sampling and real-time behavior for our purposes. (We typically run at a sample rate of 100-500Hz.) Importantly, there is no need to recode anything when moving from simulation to experiment.

Figures 6 and 7 show an example. Figure 6 shows the Modelica iconic view of a real-time, closed-loop simulation model in Dymola. For this particular controller, references for the three axes in the task frame (z) are sampled from a joystick via the Device Drivers library. The joystick inputs are interpreted as velocity commands, and consistent position and acceleration references are computed with filters in the Ref_Gen block. These are passed through the inverse kinematics block to provide references in the y-coordinates to the outer loop, which is compensated by the approximate feedback linearization (27), with variable gains to allow for adjustable impedance during contact operations. The inner loop is the gravity compensation (21). Both inner and outer loops require q which is computed by the forward kinematics block. The green blocks are all implemented in discrete time. Sample and hold blocks interface to the continuous-time DAE model of the delta robot labeled deltaRobot. The block below deltaRobot models the physics of an object in the task space, in this case a block of Lego, which interacts with the robot via the continuous-time Jacobian J_v, which is realized inside the deltaRobot model.

Figure 7 shows the Modelica model for exactly the same

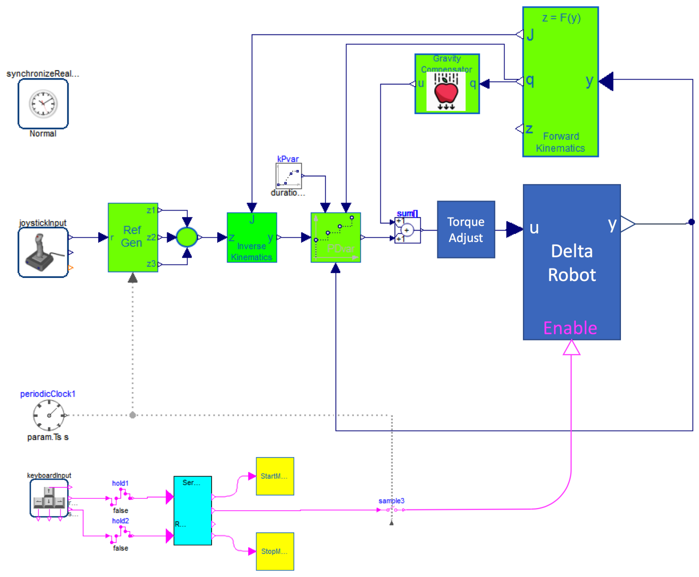

FIGURE 7. CONTROL ALGORITHM EXPERIMENTAL TEST.

controller, but where the dynamic model of the robot has been replaced with interfaces to the actual robot hardware. The Torque Adjustment and DeltaRobot blocks are from our Delta Robot Real-Time Modelica Library. These blocks sample the position and velocity of the servo motors, and write the value of torque to the motion control API using a set of C functions. At the lower left, the yellow objects contain start-up, shut-down and safety sequences represented as finite state machines, which start and stop the servomotors safely and ensure operation remains within hard limits on stroke, velocity and torque, and also prevent the robot from colliding with itself.

[3]https://openmodelica.org/

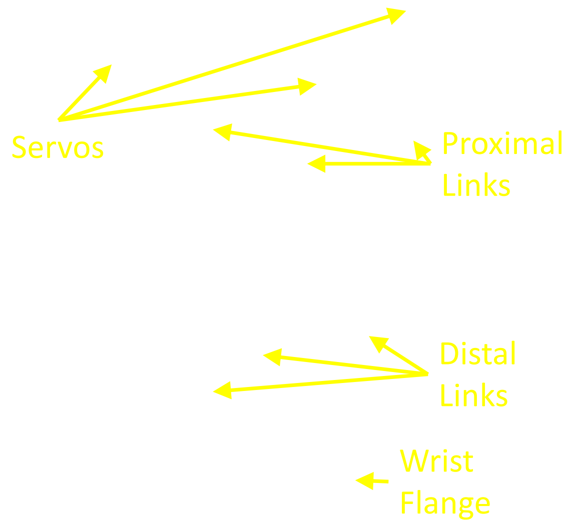

FIGURE 8. MERL DELTA ROBOT.

DELTA ROBOT HARDWARE

The MERL delta robot, known affectionately as *Kamaji* (after the industrious spider in the classic Japanese film *Spired Away*), is shown in Figure 8, with kinematics parameters listed in Table 1. It is a custom-built research robot intended for assembly control experiments. Each proximal link is machined aluminum and is directly actuated by a Mitsubishi Electric HG-KR-73B rotary AC servomotor with rated power 0.75 kW, rated torque 2.4 Nm, and 22-bit encoder resolution, giving a task-space resolution of less than 5μm. The distal links, which are under only compression or tension because of the universal joints at each end, are are hollow carbon fiber tubes to reduce weight. The wrist flange is aluminum, and the wrist and end effector, which would be mounted below the wrist flange, are not shown. The servomotors are each driven by an MR-J4-B servo amplifier, which in turn is controlled by a single MR-MC210 motion controller board that is installed in a PCI slot of the PC. The motion controller includes an API, which is a set of C-language programs that support myriad operating modes that are commonly used in industry. For our purposes, only torque mode is used.

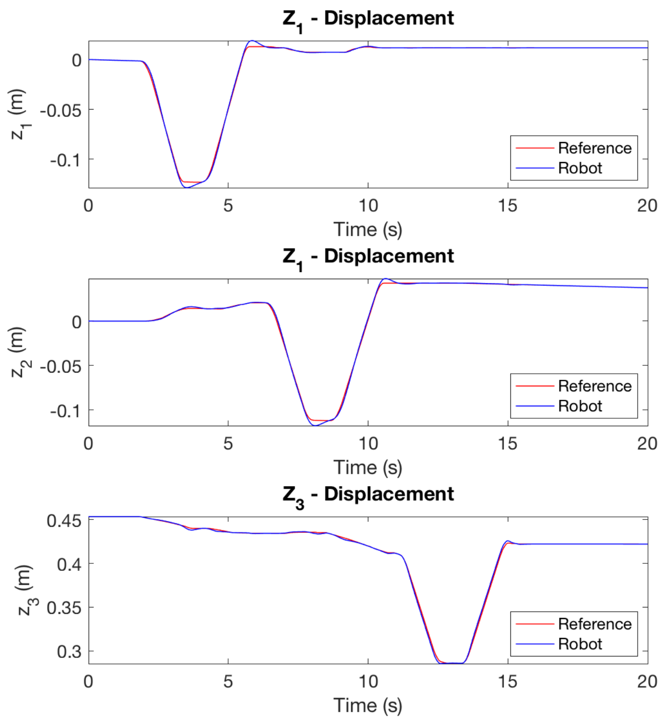

FIGURE 9. REAL-TIME SIMULATION: WRIST FLANGE POSITION.

SIMULATION AND EXPERIMENTAL RESULTS

Figure 9 shows the results of the simulation model of Figure 6, running in real-time. The joysticks are moved to command a change in velocity in the three cardinal directions in task space, and the approximate feedback linearizing control does a good job tracking the reference. For these simulations, $k_p = 1.0$ and $k_d = 0.25$, which are very low gains. The tracking quality is due to the inverse model in (27) with acceleration feed-forward and the lack of any model uncertainty.

Figure 10 shows the result of an experiment using the controller shown in Figure 6, where the joystick inputs are also used to command reference velocities. This is exactly the same control law as for the simulation, but recompiled with the experimental interface instead of the delta robot DAE model. (Of course, the reference is different.) This shows more tracking error due to some uncertainty in the model, especially in gravity compensation and joint friction, but it is still quite good given the very low outer-loop feedback gains.

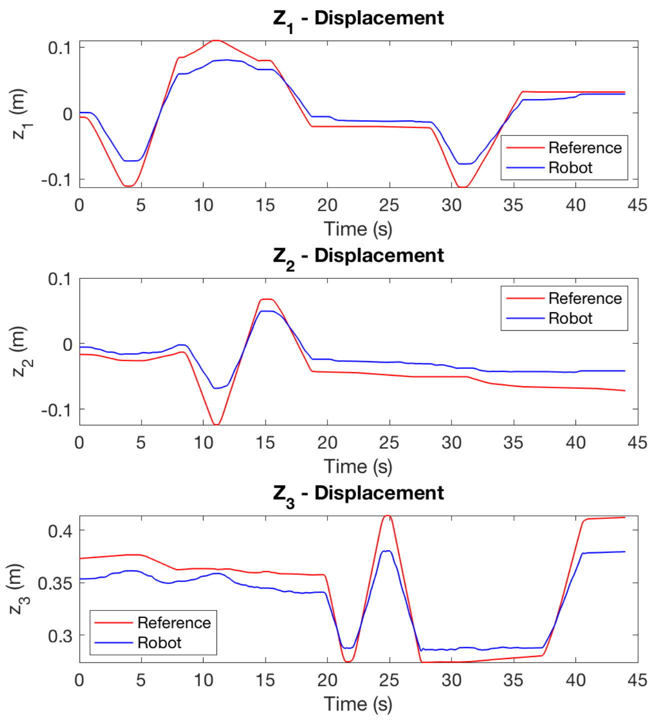

FIGURE 10. REAL-TIME EXPERIMENT: WRIST FLANGE POSITION

CONCLUSION

In this paper, we extended our earlier work on object-oriented modeling and control of delta robots by deriving two formulations of the manipulator Jacobian and also an approximate feedback linearizing control law. The Jacobian formulations are useful for different purposes, specifically for continuous-time simulation of the effect of external disturbance forces, and for feedback control. Both are computationally efficient and may prove useful for applying optimization-based path planning methods. We also described a software and hardware architecture that is realized primarily in the Modelica language, enabling representation, desktop and real-time simulation, and also real-time experimental testing of new robot control algorithms. The Modelica software architecture allows us to test algorithms in simulation and experiment without recoding, and is based almost entirely on open-source software, offering an competitive alternative to commercial software and hardware.

REFERENCES

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

[2] Bortoff, S. A., 2018. "Object-oriented modeling and control of delta robots". In IEEE Conference on Control Technology and Applications, pp. 251–258.

[3] Bortoff, S. A., 2019. "Using Baumgarte's method for index reduction in Modelica". In Proceedings of the 13th International Modelica Conference, pp. 333–342.

[4] Merlet, J.-P., and Gosselin, C., 2008. *Springer Handbook of Robotics*. Springer, ch. Parallel Mechanisms and Robots.

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

[6] Baumgarte, J. W., 1983. "A new method of stabilization for holonomic constraints". *ASME Journal of Applied Mechanics*, ***50***, pp. 869–870.

[7] Brinker, J., Corves, B., and Wahle, M., 2015. "A comparative study of inverse dynamics based on Clavel's delta robot". In Proceedings of the 14th IFToMM World Congress.

[8] Brinker, J., and Corves, B., 2015. "A survey of parallel robots with delta-like architecture". In Proceedings of the 14th IFToMM World Congress.

[9] Cellier, F. E., and Greifeneder, J., 1991. *Continuous System Modeling*. Springer.

[10] Cellier, F. E., 2006. *Continuous System Simulation*. Springer.

[11] Staicu, S., and C., D. C. C.-C. D., 2003. "Dynamic analysis of Clavel's delta parallel robot". In Proceedings of the 2003 International Conference on Robotics and Automation, pp. 4116–4121.

[12] Otter, M., 2011. Modelica overview. August.

[13] Fritzon, P., 2015. *Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. Wiley.

[14] Tiller, M., 2001. *Introduction to Physical Modeling with Modelica*. Springer.

[15] Bortoff, S. A., 2020. "Modeling contact and collisions for robotic assembly control". In Proceedings of the American Modelica Conference.

[16] Modelica Association, 2017. *Modelica Language Specification Version 3.4*. https://www.Modelica.org/.

[17] Elmqvist, H., Otter, M., and Mattsson, S. E., 2011. "Fundamentals of synchronous control in Modelica". In Proceedings of the 11th International Modelica Conference, pp. 15–25.

[18] Otter, M., Thiele, B., and Elmqvist, H., 2011. "A library for synchronous control systems in modelica". In Proceedings of the 11th International Modelica Conference, pp. 27–36.

[19] Thiele, B., Beutlich, T., Waurich, V., Sjölund, M., and Belmann, T., 2017. "Towards a standard-comform, platform-generic and feature-rich Modelica device drivers library". In Proceedings of the 12th International Modelica Conference, pp. 713–723.