Object-Oriented Modeling and Control of Delta Robots

Bortoff, S.A.

TR2018-126 August 25, 2018

Abstract

In this paper we derive a dynamic model of the Delta robot that is well-suited to an object-oriented modeling framework. The approach uses an augmented Lagrangian or Hamiltonian formulation together with Baumgarte's method of index reduction, and results in a singularity-free dynamic model that is well suited to dynamic analysis, control system synthesis and time-domain simulation. The object-oriented structure enables broad application to problems such as coordinated control and robotic assembly. We present several common control algorithms and conduct a dynamic analysis of the Delta robot that shows that the open-loop system is unstable for large volumes of the reachable workspace, which has fundamental implications on closed-loop performance.

IEEE Conference on Control Technology and Applications

Object-Oriented Modeling and Control of Delta Robots

Scott A. Bortoff[1]

Abstract— **In this paper we derive a dynamic model of the Delta robot that is well-suited to an object-oriented modeling framework. The approach uses an augmented Lagrangian or Hamiltonian formulation together with Baumgarte's method of index reduction, and results in a singularity-free dynamic model that is well suited to dynamic analysis, control system synthesis and time-domain simulation. The object-oriented structure enables broad application to problems such as coordinated control and robotic assembly. We present several common control algorithms and conduct a dynamic analysis of the Delta robot that shows that the open-loop system is unstable for large volumes of the reachable workspace, which has fundamental implications on closed-loop performance.**

I. INTRODUCTION

With their base-mounted actuators and stiff, low-mass parallel links, Delta robots are designed to meet the need for fast, precise, relatively light-weight pick-and-place industrial applications. Cycle times of three cycles-per-second with payload capacity less than 1kg is typical [1]. But Delta robots, and parallel mechanisms more generally, offer advantages over more common serial-link manipulators, such as lower mass and high stiffness, which will make them attractive for applications besides pick-and-place, such as robotic assembly. And extensions e.g. [2] to the original design [3] are addressing kinematic limitations by increasing the numbers of degrees of freedom.

Delta robots, and similar parallel mechanisms, have drawn the attention of the research community for many years to address the challenges of kinematic and dynamic modeling and control [4], [5], [6], [7]. Unlike the kinematics of serial chain robots [8], the forward kinematics of the Delta robot (the function from actuated joint angles to the location of the end effector) cannot be expressed analytically [6]. This makes formulation of dynamic (and inverse dynamic) equations of motion more difficult.

Equations of motion can be derived using a variety of methodologies such as the principle of virtual work, the Newton-Euler formulation, or Lagrangian and Hamiltonian energy-based approaches [7]. But, there is no single *correct* kinematic, dynamic or inverse dynamic model of a Delta robot (or any robot, for that matter). Different methodologies result in different models, and each has more or less merit depending on the use-case, e.g. some methodologies may be more efficient for the process of model synthesis, while others result more efficient code, and still others are more useful for feedback control system design.

The objective of this work goes beyond a derivation of yet another dynamic model of the Delta robot. In the long term, we are developing a more general modeling framework, and more concretely, model libraries, that are useful for time-domain simulation, dynamic analysis, various forms of optimization, and control system design of not just a single robot, but ultimately an entire assembly line, including the robots' tasks that go beyond pick-and-place, and at varying degrees of detail. Our vision and indeed technical approach is akin to the the ideas and mathematical rigor in [9] including zooming, tearing and accusal modeling.

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

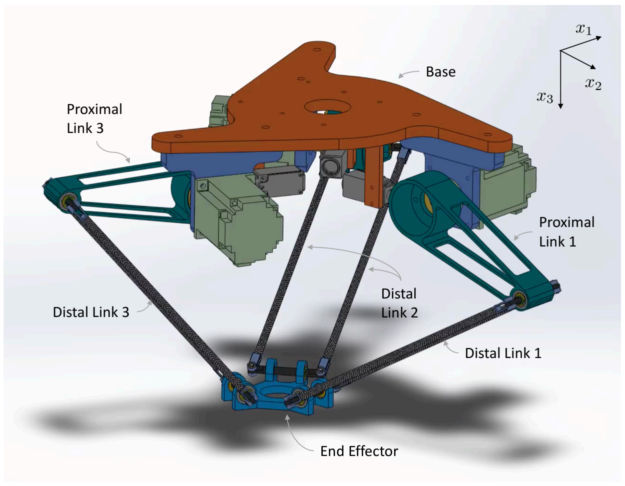


Fig. 1. Delta robot.

It is important to emphasize that modeling is expensive and time-consuming, and we seek a framework in which we can reuse models for a variety of different use-cases, such as control system design at its various hierarchal levels (e.g. low-level feedback loops and also higher-level sequences), possibly as part of the control law itself e.g., using Model Predictive Control, for model-based estimation, mechanism co-design, and also for uses that are not immediately foreseen today.

Object-oriented modeling offers a technical approach and a set of potent tools to address these needs. In particular, the modeling language Modelica has been developed for large-scale, heterogeneous, dynamic systems based on the principles of object-oriented programming, such as abstraction, encapsulation, inheritance and hierarchy as organizational concepts [10], [11], [12]. Its roots date back to the late 1970s [13]. Used properly, models can be built "bottoms up" to be useful for control system design at various levels of the hierarchy in systems as complex and vast as a robotic assembly line.

In this paper we derive Lagrangian and Hamiltonian models for the Delta robot, showing how they are structured into an object-oriented framework, and how it they are useful for not just simulation but also dynamic analysis and control system design. The mathematical model in of itself is perhaps not very novel. It is an application of constrained Lagrangian or Hamiltonian dynamics and Baumgarte's method [14], [15] of index reduction. Unlike previously published models, which are a set of 6 nonlinear ordinary differential equations corresponding to the 3 degrees of freedom, this method results in 18 implicit nonlinear ordinary differential equations in 18 variables. However, while the dimension is larger, each equation is explicit and computationally simple. A quantified analysis of this claim will be published in the future. For now, the novelty is to recognize that this particular modeling approach is well-suited to an object-oriented modeling construct, and is useful for purposes beyond simulation. Further, it is extendable to applications such as force control and cooperative robotics, and is relatively easy to understand and document.

In Section II we derive mathematically the dynamic model. In Section III we show how the models are organized as a hierarchy using the concepts of extension, encapsulation and abstraction. In Section IV we use this model to design a variety of feedback controllers, and present the results of some simple dynamic analysis. Interestingly, one result is that the Delta robot is open-loop unstable for a significant volume of its usable workspace, a surprising and relevant result that has not been reported in the literature, to our knowledge. We conclude with some comments on our unfinished work in Section V.

II. Dynamic Model

Consider the Delta robot pictured in Fig. 1, consisting of three symmetric arms constrained kinematically by universal or spherical joints at the end effector. We derive the dynamics first by defining the dynamics for each independent arm, assuming it is unconstrained, and then derive the dynamics of the robot by adding a holonomic coupling constraint representing the end effector. The resulting index-3 Differential Algebraic Equation (DAE) is stabilized using Baumgarte's method, giving an index-1 DAE that is useful directly for simulation, dynamic analysis, control system design, and is extendable to other uses such as force control.

A. Arm Dynamics

Each arm consists of a servomotor attached rigidly a proximal link, which is in turn attached to a distal link by an unactuated universal joint, giving it three degrees of freedom. Referring to Figs. 1-2 in which the fixed "world" frame has axes labeled $[x_1, x_2, x_3]$, let $\phi = [\phi_1, \phi_2, \phi_3]^T$ be the generalized angular position for the arm, defined as follows. The servomotor angle is ϕ_1, which is the rotation of the proximal link about the x_1-axis, measured with respect to the x_2-axis. The universal joint position is represented with ϕ_2 representing the rotation *about* the x_1-axis measured with respect to the x_2-axis, and ϕ_3 representing the rotation *about* the x_2-axis measured with respect to the $x_2 - x_3$ plane. Note that, in these coordinates, the universal joint has a singularity

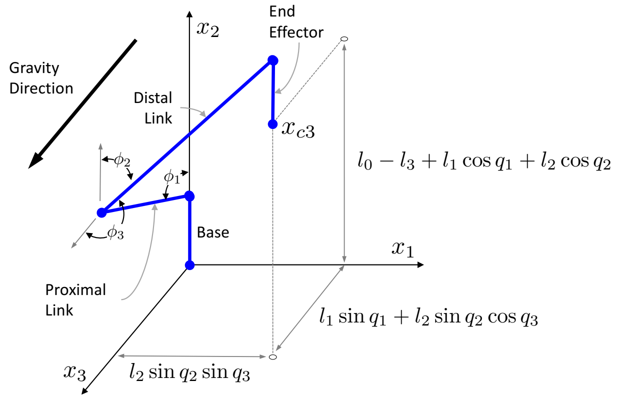


Fig. 2. Delta robot arm coordinates with end effector location x_{c3} indicated. Note that the Cartesian coordinate frame axes are denoted x_1, x_2 and x_3.

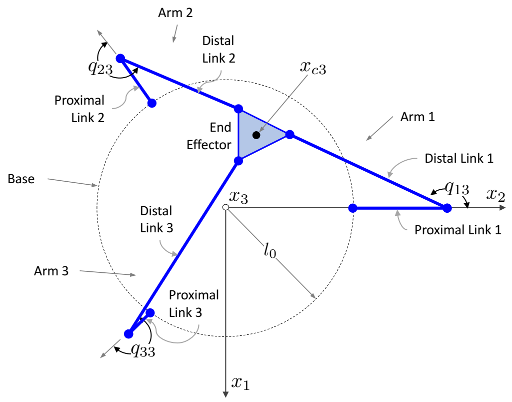


Fig. 3. Delta robot coordinates, bottom view, looking up. Note that the Cartesian coordinate frame axes are denoted x_1, x_2 and x_3.

at $\phi_2 = 0$. However, this is outside the range of motion of the robot once the three arms are kinematically constrained by the end effector.

Assuming that the distal links are thin rods, i.e., neglecting the inertia of the distal link about its longitudinal axis, the kinetic energy of each arm, including 1/3 the mass of the end effector, is

$$\begin{aligned} T(\phi, \dot{\phi}) &= \frac{1}{2} m_1 \dot{x}_{c1}^T \dot{x}_{c1} + \frac{1}{2} m_2 \dot{x}_{c2}^T \dot{x}_{c2} + \frac{1}{6} m_3 \dot{x}_{c3}^T \dot{x}_{c3} \\ &\quad + \frac{1}{2} J_1 \dot{\phi}_1^2 + \frac{1}{2} J_2 \left(\sin(\phi_2)^2 \dot{\phi}_3^2 + \dot{\phi}_2^2 \right), \end{aligned} \tag{1}$$

where the position of the center of mass of the proximal link is

$$x_{c1} = \begin{bmatrix} 0.0 \\ l_{c1} \cos(\phi_1) \\ l_{c1} \sin(\phi_1) \end{bmatrix}, \tag{2}$$

the position of the center of mass of the distal link is

$$x_{c2} = \begin{bmatrix} l_{c2} \sin(\phi_2) \sin(\phi_3) \\ l_1 \cos(\phi_1) + l_{c2} \cos(\phi_2) \\ l_1 \sin(\phi_1) + l_{c2} \sin(\phi_2) \cos(\phi_3) \end{bmatrix}, \tag{3}$$

TABLE I

DELTA ROBOT PARAMETER DEFINITIONS.

Symbol	Description (Units)
l_0	Base radius (m)
l_1	Length of proximal link (m)
l_2	Length of distal link (m)
l_3	Width of end effector (m)
l_{c1}	Distance to proximal link center of mass (m)
l_{c2}	Distance to distal link center of mass (m)
m_1	Mass of proximal link (kg)
m_2	Mass of distal mass (kg)
m_3	1/3 mass of end effector (kg)
J_1	Rotational inertia, proximal link $(\mathrm{N}\cdot\mathrm{m/s^2})$
J_2	Rotational inertia, distal link $(\mathrm{N}\cdot\mathrm{m/s^2})$
g	Acceleration due to gravity $(\mathrm{m/s^2})$

the position of the center of mass of the end effector is

$$x_{c3} = \psi(\phi) := \begin{bmatrix} l_2\sin(\phi_2)\sin(\phi_3) \\ l_1\cos(\phi_1) + l_2\cos(\phi_2) \\ l_1\sin(\phi_1) + l_2\sin(\phi_2)\cos(\phi_3), \end{bmatrix}, \quad (4)$$

the velocities $\dot{x}_{c1}$, $\dot{x}_{c2}$ and $\dot{x}_{c3}$ are computed by the chain rule to be functions of ϕ, $\dot{\phi}$ and the parameters are listed in Table I. Equation (1) is simply the expression for kinetic energy of a spherical pendulum [16], representing the distal link, mounted at the end of a simple pendulum representing the proximal link. Note that the forward kinematics of the arm are defined as $\psi(\phi)$ in (4).

The gravitational potential energy of each arm is

$$\begin{aligned} V(\phi) &= -g((l_{c1}m_1 + l_1(m_2 + m_3/3))\sin(\phi_1) \\ &\quad + (l_{c2}m_2 + l_2m_3/3)\sin(\phi_2)\cos(\phi_3)\,, \quad (5) \end{aligned}$$

where gravity points along the positive x_3 axis and 1/3 of the mass of the end effector is included in each arm. The Lagrangian

$$L(\phi,\dot{\phi}) = T(\phi,\dot{\phi}) - V(\phi) \quad (6)$$

is used to define the arm equations of motion in the conventional manner,

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\phi}} - \frac{\partial L}{\partial \phi} = bu, \quad (7)$$

giving

$$m(\phi)\ddot{\phi} + c(\phi,\dot{\phi}) + g(\phi) = bu, \quad (8)$$

where m is the 3×3 inertia matrix, c is the 3×1 vector of Coriolis and centripetal torques, g is the 3×1 vector of torques due to gravity, $b = [1,0,0]^T$ and u is the servomotor torque. Expressions for m, c and g are given in Appendix 1.

B. Robot Lagrangian Dynamics

Each of the three arms is identical except for a $120°$ rotation about the z-axis. To represent the dynamics of the full robot, we sum the unconstrained Lagrangians for each arm (6), and augment the result with the holonomic constraints that equate the x_{c3} positions of the end effectors of each arm (4) in the world coordinates. Euler's equation gives the constrained dynamical equations.

Referring to Fig. 3, define $q_i \in \mathbb{R}^3$ for $1 \le i \le 3$, to be the generalized angular position of each of the three arms, replacing the ϕ-notation used in Section II-A. Define $q = [q_1, q_2, q_3]^T \in \mathbb{R}^9$ and the unconstrained Lagrangian as

$$L_u(q,\dot{q}) = L(q_1,\dot{q}_1) + L(q_2,\dot{q}_2) + L(q_3,\dot{q}_3),$$

and form the augmented robot Lagrangian as

$$L_a(q,\dot{q}) = L_u(q,\dot{q}) + \lambda^T h(q), \quad (9)$$

where the constraint $h(q) : \mathbb{R}^9 \to \mathbb{R}^6$ is

$$h(q) = \begin{bmatrix} \psi(q_1) - R_z(2\pi/3)\cdot\psi(q_2) \\ \psi(q_1) - R_z(-2\pi/3)\cdot\psi(q_3) \end{bmatrix}, \quad (10)$$

the rotation matrix

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad (11)$$

ψ is defined in (4), and $\lambda \in \mathbb{R}^6$ is a vector of Lagrange multipliers. Then the Lagrangian equations of motion for the robot are

$$\begin{aligned} \frac{d}{dt}\frac{\partial L_a}{\partial \dot{q}} - \frac{\partial L_a}{\partial q} &= \lambda^T H(q) + Bu \quad (12) \\ h(q) &= 0, \quad (13) \end{aligned}$$

where

$$H(q) = \frac{\partial h(q)}{\partial q}.$$

Defining $v = \dot{q}$, (12)-(13) can be written as a set of 24 first-order Differential Algebraic Equations (DAEs) of Index 3 [17], [18], in the variables $q \in \mathbb{R}^9$, $v \in \mathbb{R}^2$ and $\lambda \in \mathbb{R}^6$,

$$\begin{aligned} \dot{q} &= v \quad (14) \\ M(q)\dot{v} + C(q,v) + G(q) &= \lambda^T H(q) + Bu \quad (15) \\ h(q) &= 0, \quad (16) \end{aligned}$$

where

$$\begin{aligned} M(q) &= \mathrm{diag}\,(m(q_1), m(q_2), m(q_3)) \in \mathbb{R}^{9\times 9}, \\ C(q,v) &= \mathrm{diag}(c(q_1,v_1), c(q_2,v_2), c(q_3,v_3)) \in \mathbb{R}^9, \\ G(q) &= \mathrm{diag}(g(q_1), g(q_2), g(q_3)) \in \mathbb{R}^9, \\ B &= \mathrm{diag}\,(b,b,b) \in \mathbb{R}^{9\times 3}. \end{aligned}$$

Equations (14) - (16) are a complete dynamic model of the Delta robot. However, the high index make them unsuitable to direct application of most modern solvers such as DASSL [17], and much of modern control theory. Index reduction is discussed in Section II-D.

C. Robot Hamiltonian Dynamics

For some applications such as port-Hamiltonian analysis [19] it is useful to have a Hamiltonian model of the robot. This is derived in similar fashion by defining the momentum vector $p \in \mathbb{R}^9$ and the Hamiltonian $H = T + V$ for each arm, augmenting the constraint (10) by the Lagrange multiplier λ and solving the Hamiltonian equations, resulting in

$$\begin{aligned} M(q)\dot{q} &= p && (17)\\ \dot{p} &= \frac{1}{2}v\frac{\partial M(q)}{\partial q}v - G(q) + Bu + H^T(q)\lambda && (18)\\ h(q) &= 0, && (19)\end{aligned}$$

where the partial derivatives of M need to be computed symbolically. This formulation has about the same computational complexity as (14)-(16), results in similar numerical solutions using the same type of solver, but could be used with a symplectic solver to provide speedup if desired.

D. Index Reduction

There are a number of methods to reduce the index of (14)-(16) or (17)-(19) to 1 or 0 (ODE), transforming the system to a new set of equations that is useful for simulation, dynamic analysis and control design. A comprehensive survey is [20]. We highlight in particular the method of "dummy derivatives" [21], [22], [12] which has been particularly successful in our context. It identifies a subset of the states and represents the remainder as dependent algebraic functions. Its advantage is that the number of dynamic states selected is 6, the minimum number required to represent the dynamics, and no additional constraint stabilization is necessary. The algorithm is also very robust and has been automated in tools such as Dymola [22].

But the "dummy derivative" method has two disadvantages. First, there is no single set of selected states that covers the entire robot workspace. At least two sets of states and state equations must be computed and realized in code, and the solver must switch among them to compute a time-domain simulation. This is because the forward kinematics are not one-to-one, which cause the Jacobian of the reduced system (not to be confused with the manipulator Jacobian) for any one set to become singular at certain points in the workspace. Modern tools such as Dymola use proprietary logic to switch among the selected states and equations, together with index 1 DAE solvers such as DASSL [17], [23] to compute an accurate numerical solution. The second disadvantage is that the analytic expressions for the dependent states are complex due to the trigonometric functions in the kinematics. Both of these properties adversely affect simulation time.

Here we use Baumgarte's method of index reduction [20], [14], [15], which results in a single set of index 1 DAEs that covers the entire robot workspace. There are no singularities. Importantly, this single model can be used for dynamic analysis and model-based control design in addition to time domain simulation. And it can be extended to consider robotic interactions with the environment, cooperative control of multiple robots, and consideration of multi-physical problems in which detailed models of servomotors, for example, can be inserted.

In this method, the constraint (16) is replaced with a linear combination of its first two derivatives with respect to time. Define

$$\begin{aligned} z_0 &= h(q) && (20)\\ z_1 &= \dot{z}_0 = \frac{\partial H(q)}{\partial q}\dot{q} && (21)\\ z_2 &= \dot{z}_1 = \dot{H}(q)\dot{q} + H(q)M^{-1}(q)\left(\lambda^T H(q)\right.\\ &\quad \left. + Bu - C(q,\dot{q}) - G(q)\right), && (22)\end{aligned}$$

and replace (16) or (19) with

$$z_2 + \alpha_1 z_1 + \alpha_0 z_0 = 0, \qquad (23)$$

where $s^2 + \alpha_1 s + \alpha_0 = 0$ is a Hurwitz polynomial (all poles in the open left-half plane). The model (14)-(15) and (23) or (17)-(18) and (23), is an index 1 DAE with 18 differential equations, 6 algebraic equations and 24 states q, v and λ, or q, p and λ, respectively.

E. Analysis and Discussion

In this section we limit discussion to the Lagrangian formulation, but the comments and results also hold for the Hamiltonian formulation. It is well documented in the numerical analysis literature that the mathematical solution to (14)-(15) and (23) is identical to (14)-(15) and (16). Research has focused on criteria for selection of values of α_i that might improve the numerical performance of solvers. For our purposes it is useful to understand the structure of the index-1 system (14)-(15) and (23) from a control theoretic point of view.

Following [24], we define $\xi = [z_0, z_1]^T \in \mathbb{R}^{12}$ to be the "linear" part. Then there exist coordinates $\eta \in \mathbb{R}^6$ which are functions of q and v (after eliminating λ through laborious algebraic manipulation) so that (14)-(15) and (23) can be written locally in so-called Zero Dynamics Normal Form [24],

$$\begin{aligned} \dot{\eta} &= f(\eta, \xi, u) && (24)\\ \dot{\xi} &= A\xi, && (25)\end{aligned}$$

where the 12 eigenvalues of A are located at the roots of (23), and the 6-dimensional zero dynamics

$$\dot{\eta} = f(\eta, 0, u) \qquad (26)$$

define the dynamics of the robot. In other words, there is a 6-dimensional manifold defined by $\xi = 0$ on which the robot dynamics exist and evolve according to (26). The ξ-dynamics are exponentially stable, are not controllable from u, and once they converge to 0, do not affect q or v. This means that if we linearize (14)-(15) and (23), we expect to see 12 poles and zeros at the roots to (23), and these dynamics are neither controllable nor observable. They are easily removed using a Hankel-type model truncation, since the corresponding Hankel singular values are all zero, and the resulting reduced order model is 6 dimensional and equivalent to a linearization obtained otherwise. We therefore have a model that is well-suited to at least linear control system design and associated dynamical analysis.

We remark that, in practice, expressions for z_1 and z_2 in (21)-(22) can be computed automatically using a tool such as

Dymola. Also, because the model is an index 1 DAE (instead of an index 0 ODE), it is not necessary to compute the inverse of the inertia matrix for either the Lagrangian or Hamiltonian formulations. Further, it is *not* necessary to compute η or f in (24)-(25). Deriving these expressions is done to understand the structure and properties of the 18-dimensional dynamic equations. Indeed the model can be expressed compactly and correctly as shown in the Appendix.

The primary disadvantages of Baumgarte's method are (1) that 24 equations in 24 variables are produced, instead of the minimal six (although λ can be removed by algebraic manipulation, leaving 18 implicit first-order differential equations in 18 differential variables), and (2) that numerical solutions to (14)-(15) and (23) can drift off the constraint manifold $h = 0$ when the system is in motion. For the latter reason, Baumgarte's original method has been criticized in the numerical analysis literature, and modifications have been proposed to improve its numerical performance [20]. Yet for this application we find the drift to be very small - on the order less than microns for typical tolerance settings. Further, the drift is easily computable for monitoring purposes and controllable in the sense that it is reduced by reducing the solver tolerance. Moreover, simulation times for (14)-(16) are an order-of-magnitude faster than the model that results from index reduction by the dummy derivative method, despite the fact that we require three times more equations and dynamic states, due to the simplicity of the equations. Finally, the model is amenable to dynamic analysis and control system design directly and is singularity-free.

III. Modelica Realization

The Modelica library is organized as a hierarchy, with partial models of the kinematics and parameter definitions at the lowest level, full models of the arms at the intermediate level, and models of the full robot at the highest level. The kinematics, arm dynamics and full Lagrangian robot dynamics are listed in the Appendix. For readers unfamiliar with Modelica, the syntax is Matlab-like with a some important differences. First, it is a declarative language, so order is not relevant in the source code. Second, the equals sign is not an assignment, but rather a declaration that the left-hand side and right-hand side are constrained to be equal, as in physics. A key operator is the derivative operator `der(·)` which symbolically computes derivatives with respect to time. This is used, for example, in the full robot model listed in the Appendix to compute z_1 and z_2. The Modelica compiler symbolically computes these derivatives, saving the user tedious labor and making the source code compact, readable and relatively easy to verify.

The language and library include important elements of object-orientation. At the lowest level of the hierarchy, the kinetics are defined in a partial model, which is reused at higher levels. At the next level, the arm dynamics are defined. We have both Lagrangian and Hamiltonian representations defined. The arm dynamics are encapsulated within their own model, because at the robot level, these details are not relevant. At the third level, the full delta robot model is

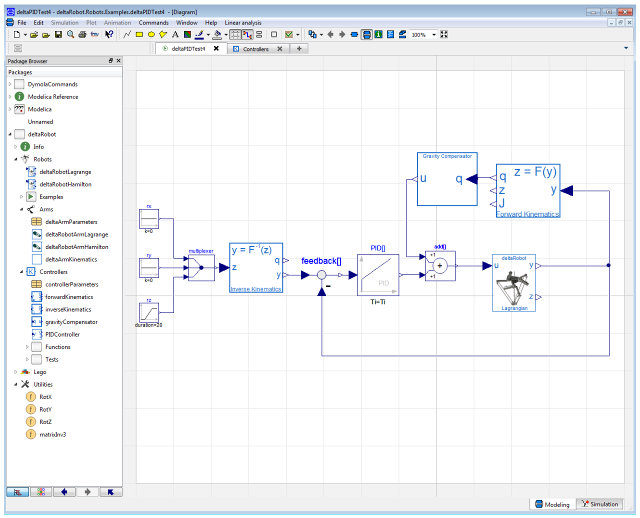


Fig. 4. Dymola screenshot showing the Modelica deltaRobot library (left) and an a gravity-compensating PID feedback controller (right), showing the use of forward and inverse kinematics, gravity compensation and PID. The library contains models of the kinematics at the lowest level, arms, and robots at its highest level. We also have a package of controller components and a growing library of tasks, such as assembling Lego as a "toy" problem.

defined. Arms are instantiated, and can be replaced should the user wish to define a new type of arm. The number of arms can be changed, by simply declaring the desired number, and changing the few lines that define the end effector constraint. In fact, at this level, the only equations introduced are the stabilized constraints.

At a higher level, multiple robots can be declared, and constraints among them defined in a manner analogous to what we have done for the arms. This allows for analysis of cooperative control using the same mathematics and approach. We may also consider force control and contact and collisions although these are beyond the present paper's scope. In addition to the robot models, the library contains a package of controller components, realizing functions like forward and inverse kinematics that are computed using Netwon's method directly in the language. This has a similar hierarchy. In addition, we have developed a growing library of assembly tasks, which are also beyond the scope of this paper. One package includes models of Lego bricks as "toy" assembly problems. The bricks possess 3 body dynamics that include elastic collisions and allow for the accurate modeling of contact, friction etc. that is necessary when modeling the process of assembly.

For the reader interested in learning more about Modelica, an approachable introduction is [25], while [10] is more thorough and comprehensive. The texts [11] and [12] are excellent introductions to the subjects of multi-physical modeling and simulation.

IV. Dynamic Analysis and Control

The model (14)-(15), (23) is useful for model-based control design and analysis, directly. Here we highlight a few applications and examples.

A. Forward Kinematics

The forward kinematics function takes as input the three joint measurements at the servos and computes the other six joint angles and the location of the end effector in world coordinates. The robot Jacobian is also computed. The forward kinematics are one-to-one but not onto, and defined implicitly by (10), which needs to be solved numerically. Specifically, partition q into measured and unmeasured states by defining $y = [q_{11}, q_{21}, q_{31}]^T$ to represent the measured joint angles, and $x = [q_{12}, q_{13}, q_{22}, q_{23}, q_{32}, q_{33}]^T$ to represent the unmeasured joint angles, and rearrange the variables of h so that (10) can be written

$$h(x, y) = 0. \tag{27}$$

This is easily solved using Newton's method

$$\frac{\partial h}{\partial x}(x_k, y) \cdot (x_{k+1} - x_k) = -h(x_k, y), \tag{28}$$

which typically converges to 7 decimal places of accuracy in 2-3 iterations since it can be initialized close to its solution in any real-time application. Of course each iteration requires the solution to a 6-dimensional set of linear equations. With the solution (x, y), the end effector location is computed using ψ in (4), and the robot Jacobian is available as a by product of the Newton iterations.

B. Inverse Kinematics

The inverse kinematics takes as input a location of the end effector $w \in \mathbb{R}^3$ and computes values for the joint angles $q \in \mathbb{R}^9$. This is not one-to-one: there is not a unique solution for all values of w. The inverse kinematics defined implicitly by the nine equations

$$\begin{aligned} \psi(q_1) - w &= 0 && (29)\\ \psi(q_2) - w &= 0 && (30)\\ \psi(q_3) - w &= 0. && (31) \end{aligned}$$

This can also be solved using Newton's method in the conventional manner, with some logic for choosing the desirable solutions. Each Newton iteration involves computing the solution to three 3-dimensional linear systems of equations, making the complexity less than the forward kinematics.

C. Gravity Compensation

One popular control scheme is to cancel the effect of gravity on the manipulator with an inner loop, and then close an outer feedback loop with a linear PD or PID compensator. The gravity compensating feedback is computed as the solution to the 9-dimensional linear set of equations

$$\begin{bmatrix} u \\ \lambda \end{bmatrix} \cdot [B \ H^T(q)] = G(q), \tag{32}$$

where in any real-time application q is computed via the forward kinematics from the joint measurements y. A model of this feedback scheme is shown in Fig. 4.

D. Feedback Linearization

A feedback linearizing control law can be defined as follows. Let

$$w_1 = \psi(q_1) \tag{33}$$

denote the location of the end effector. Symbolically differentiate this twice

$$\begin{aligned} w_2 &= \dot{w}_1 = d\psi(q_1)v_1 && (34)\\ \dot{w}_2 &= \dot{d\psi}(q_1)v_1 + d\psi(q_1)\dot{v}_1. && (35) \end{aligned}$$

Solving (15) for $\dot{v}$ and substituting the result into (35) gives

$$\dot{w}_2 = \alpha(q) + \beta(q) \cdot u$$

from which the control law

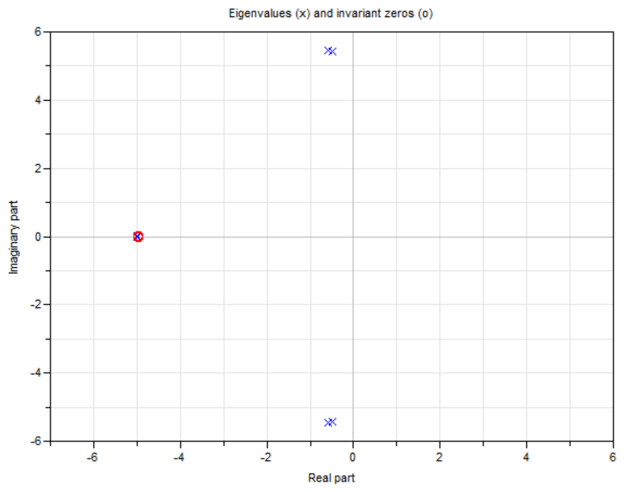
$$u = \frac{1}{\beta(q)}\left(-\alpha(q) - k_1 w_1 - k_2 w_2 + w_r\right)$$

renders the system linear from w_r to w_1. Expressions for α and β can be computed automatically. They will require inversion of the inertia matrix M, which is not difficult because it is block diagonal.

E. Linear Control Design and Analysis

The model (14)-(15), (23) enables dynamic analysis and linear model-based design methodologies. By this we mean that a linearization can be computed symbolically (using tools such as Dymola or OpenModelica) and evaluated numerically, and the resulting linear system can be used for frequency-domain analysis and design, for example, or any similar methodology. By way of example, we compute the linearization at two equilibria, using parameters values measured from a Delta robot that we have constructed in our laboratory. Pole-zero plots are shown in Fig. 5. First, notice that there are 12 pole-zero cancellations at $s = -5$ corresponding to the dynamics of (23), as expected. These do not affect the input-output behavior and can be eliminated from the system by a Hankel norm-based truncation. In the top plot, the robot is linearized at the equilibria $q_{i1} = \pi/4$ for $1 \leq i \leq 3$, i.e, the proximal links have identical angles of $\pi/4\,\mathrm{rad}$. We see poles corresponding to a lightly damped, stable response, as we might expect. In the bottom plot, the robot is linearized when the proximal links are at an angle of $0\,\mathrm{rad}$. Perhaps surprisingly, the second configuration is open-loop unstable. Note that both of these configurations are well within the reachable workspace. (The unstable root crosses into the right-half plane at an angle of approximately $q_{i1} = 22°$, for our robot.) It should be noted that this kind of instability is neither uncommon, nor is it a unique property of parallel-link mechanisms. For example, the Acrobot [26] has a similar property even near its pendant configurations.

Of course, the closed-loop can be stabilized using a PID controller for each axis, which is very common in practice. It is also notable that a gravity-compensating inner loop (32) will render all equilibrium solutions in the robot workspace exponentially stable. But the fact remains that the open-loop system, regardless of the feedback controller used, is unstable for a significant range of its workspace. This has important



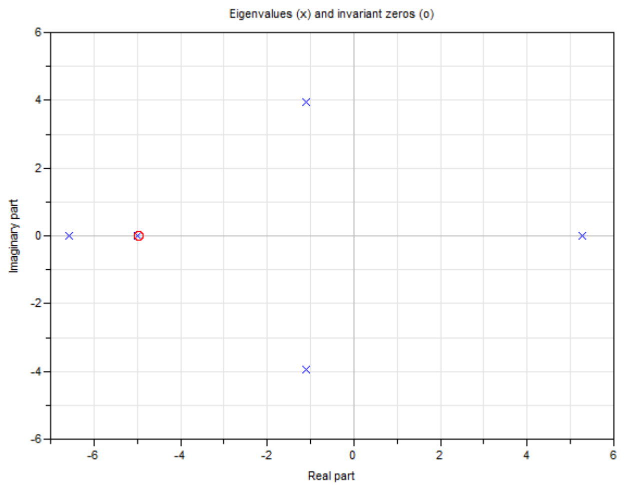


Fig. 5. Pole-zero plot of the system in equilibrium with $q_1 = \pi/4\,\text{rad}$ (top) and $q_1 = 0\,\text{rad}$ (bottom) for all three proximal links. In both plots there are 12 pole-zero pairs at $s = -5$ corresponding to the uncontrollable, unobservable dynamics of (23). In the top plot the 6 poles of the zero dynamics are all near $s = -0.5 \pm j5.5$, giving the system a lightly damped response. The bottom plot shows 4 poles at approximately $s = -2 \pm j$, moving slightly from the top configuration, one at $s = -6.5$, and the unstable pole at $s = 5.2$.

consequences. For example, stabilizing feedback gains have lower limits [27]. In some applications such as fine force control, it is common practice to reduce feedback gains to maintain stability during contact. But this clearly has limits. A careful control designer must be aware of this property, which it is not at all obvious without model-based analysis.

V. Conclusions

In this paper we derived a dynamic model for the delta robot that uses an augmented Lagrangian formulation together with Baumgarte's method of index reduction. The model is well structured for implementation in an object-oriented framework, and can be used directly for control system synthesis, model-based design and dynamic analysis, and also time-domain simulation. Importantly it is extendable to study problems such as cooperative control and robotic assembly.

The reader familiar with Delta dynamic models may be dismayed that this approach requires the calculation of the full set of 9 joint angles for simulation or control. For a real-time control application this must be done numerically and requires the solution to a set of linear equations associated with Newton iterations, or example. But with advances in solver algorithms, we do not feel that this is a significant impediment to either model-based design or real-time implementation, and it facilitates investigation of new methods of control for new purposes, with a clear and easy to understand technical approach.

Future work includes the development of the task library and its use in developing new control algorithms especially for robotic assembly. Contact and collisions are central to this study, and this work lays a firm foundation that can scale up to ultimately very large scale factory automation settings.

Acknowledgment

We gratefully acknowledge Akihiko Imagi, Masahiko Kurishige, Mitsunori Saito and Tamon Yamazaki of the Advanced Technology Center, Mitsubishi Electric Corporation, Japan, for their strong collaboration and unwavering support of this work.

Appendix

Arm kinematics are defined in the following partial Modelica model.

```
partial model deltaArmKinematics

 deltaArmParameters p;   // Record of parameters
 Real q[3], psi[3], dpsi[3,3];

equation
 // End effector location in world coordinates...
 psi[1] = p.L2*sin(q[2])*sin(q[3]);
 psi[2] = p.L1*cos(q[1]) + p.L2*cos(q[2]) + p.L0 - p.L3;
 psi[3] = p.L1*sin(q[1]) + p.L2*sin(q[2])*cos(q[3]);

 // Gradient of end effector location ...
 dpsi[1,1] = 0.0;
 dpsi[1,2] = p.L2*cos(q[2])*sin(q[3]);
 dpsi[1,3] = p.L2*sin(q[2])*cos(q[3]);
 dpsi[2,1] =-p.L1*sin(q[1]);
 dpsi[2,2] =-p.L2*sin(q[2]);
 dpsi[2,3] = 0.0;
 dpsi[3,1] = p.L1*cos(q[1]);
 dpsi[3,2] = p.L2*cos(q[2])*cos(q[3]);
 dpsi[3,3] = -p.L2*sin(q[2])*sin(q[3]);
end deltaArmKinematics;
```

Arm dynamics are defined in the following Modelica model, extending the kinematics model. Extension allows multiple different models to be expressed, such as a Hamiltonian model, reusing the kinematics model.

```
model deltaRobotArmLagrange

extends deltaArmKinematics;
Real v[3], tau[3];

protected
Real M[3,3];
Real C1[3], C2[3], C3[3], C4[3], C5[3], C6[3];
Real G[3];

equation
// Inertia Matrix...
m[1,1]=p.J1+p.LC1^2*M1+p.L1^2*(p.M2+p.M3);
m[1,2]=p.L1*(p.LC2*p.M2+p.L2*p.M3)...
    *(cos(q[1])*cos(q[2])*cos(q[3])+sin(q[1])*sin(q[2]));
```

```
m[1,3]=-p.L1*(p.LC2*p.M2+p.L2*p.M3)...
    *cos(q[1])*sin(q[2])*sin(q[3]);
m[2,1]=m[1,2];
m[2,2]=p.J2+p.M2*p.LC2^2+p.M3*L2^2;
m[2,3]=0.0;
m[3,1]=m[1,3];
m[3,2]=0.0;
m[3,3]=(p.J2+p.M2*p.LC2^2+p.M3*p.L2^2)*sin(q[2])^2;

// Centripetal and Coriolis vectors...
c1[1]=0.0;
c1[2]=p.L1*(p.LC2*p.M2+p.L2*p.M3)*(cos(q[1])*sin(q[2])...
    -cos(q[2])*cos(q[3])*sin(q[1]));
c1[3]=p.L1*(p.LC2*p.M2+p.L2*p.M3)...
    *sin(q[1])*sin(q[2])*sin(q[3]);
c2[1]=p.L1*(p.LC2*p.M2+p.L2*p.M3)*(cos(q[2])*sin(q[1])...
        -cos(q[1])*cos(q[3])*sin(q[2]));
c2[2]=0.0;
c2[3]=0.0;
c3[1]=-(p.L1*(p.LC2*p.M2+p.L2*p.M3)...
    *cos(q[3])*cos(q[1])*sin(q[2]));
c3[2]=-(p.J2+p.LC2^2*p.M2+p.L2^2*p.M3)....
    *cos(q[2])*sin(q[2]);
c3[3]=0.0;
c4[1]=0.0;
c4[2]=0.0;
c4[3]=0.0;
c5[1]=-2.0*p.L1*(p.LC2*p.M2+p.L2*p.M3)...
    *cos(q[1])*cos(q[2])*sin(q[3]);
c5[2]=0.0;
c5[3]=(p.J2+p.LC2^2*p.M2+p.L2^2*p.M3)*sin(2.0*q[2]);
c6[1]=0.0;
c6[2]=0.0;
c6[3]=0.0;

// Gravity vector...
G[1]=-p.g*(p.LC1*p.M1+p.L1*(p.M2+p.M3))*cos(q[1]);
G[2]=-p.g*(p.LC2*p.M2+p.L2*p.M3)*cos(q[2])*cos(q[3]);
G[3]= p.g*(p.LC2*p.M2+p.L2*p.M3)*sin(q[2])*sin(q[3]);

// Arm Dynamics...
der(q) = v;
m*der(v)+c1*v[1]^2+c2*v[2]^2+c3*v[3]^2+c4*v[1]*v[2]...
     +c5*v[2]*v[3]+c6*v[1]*v[3]+G+p.DAMPING.*v = tau;

end deltaRobotArmLagrange;
```

Below is the Lagrangian robot model. Note that the derivatives of h are computed automatically.

```
model deltaRobotLagrange

Arms.deltaRobotArmLagrange arm1, arm2, arm3;
Real lambda[6];
Real h0[6], h1[6], h2[6];
Input Real u[3];
parameter Real POLE = 5.0;

protected
constant Real Rot2[3,3] = Utilities.RotZ(2.0*PI/3.0);
constant Real Rot3[3,3] = Utilities.RotZ(-2.0*PI/3.0);
constant Real B[3] = {1, 0, 0};  // input torque vector

equation
// tau = H^T(q) * lambda...
arm1.tau=transpose(arm1.dpsi)*lambda[1:3]...
    +transpose(arm1.dpsi)*lambda[4:6]+B*u[1];
arm2.tau=-transpose(Rot2*arm2.dpsi)*lambda[1:3]+B*u[2];
arm3.tau=-transpose(Rot3*arm3.dpsi)*lambda[4:6]+B*u[3];

// Baumgarte's method of index reduction...
h0=cat(1,arm1.psi-Rot2*arm2.psi,arm1.ps -Rot3*arm3.psi);
h1=der(h0);
h2=der(h1);
zeros(6) = h2 + 2.0 * POLE * h1 + POLE^2 * h0;

end deltaRobotLagrange;
```

References

[1] J. Brinker and B. Corves, “A survey of parallel robots with delta-like architecture,” in *Proceedings of the 14th IFToMM World Congress*, Oct. 2015.

[2] J. Brinker, N. Funk, P. Ingenlath, Y. Takeda, and B. Corves, “Comparative study of serial-parallel delta robots with full orientation capabilities,” *IEEE Robotics and Automation Letters*, vol. 2, no. 1, pp. 920–926, April 2017.

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

[4] P. Guglielmetti, “Model-based control of fast parallel robots: A global approach in operational space,” Ph.D. dissertation, Ecole Polytechnique Federale de Lausanne, 1994.

[5] S. St. and C.-C. D. C., “Dynamic analysis of clavel’s delta parallel robot,” in *Proceedings of the 2003 International Conference on Robotics and Automation*, 2003, pp. 4116–4121.

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

[7] J. Brinker, B. Corves, and M. Wahle, “A comparative study of inverse dynamics based on clavel’s delta robot,” in *Proceedings of the 14th IFToMM World Congress*, Oct. 2015.

[8] M. M. Spong and M. Vidyasagar, *Robot Dynamics and Control*. Wiley, 2004.

[9] J. C. Willems, “The behavioral approach to open and interconnected systems: Modeling by tearing, zooming and linking,” *IEEE Control Systems Magazine*, vol. 27, pp. 46–99, December 2007.

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

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

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

[13] H. Elmqvist, “A structured model language for large continuous systems,” Ph.D. dissertation, Lund Institute of Technology, 1978.

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

[15] ——, “A new method of stabilization for holonomic constraints,” *ASME Journal of Applied Mechanics*, vol. 50, pp. 869–870, 1983.

[16] L. D. Landau, *Course of Theoretical Physics: Volume 1 Mechanics*. Butterworth-Heinenann, 1976.

[17] K. E. Brenan, S. L. Cambell, and L. R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. SIAM, 1996.

[18] P. Kunkel and V. Mehrmann, *Differential-Algebraic Equations: Analysis and Numerical Solution*. European Mathematical Society, 2006.

[19] A. van der Schaft, *Surveys in Differential-Algebraic Equations I*. Springer, 2013, ch. Port-Hamiltonian Differential-Algebraic Systems, pp. 173–226.

[20] O. A. Bauchau and A. Laulusa, “Review of contemporary approaches for constraint enforcement in multibody systems,” *Journal of Computational and Nonlinear Dynamics*, 2007.

[21] S. E. Mattsson and G. Söderlind, “Index reduction in differential algebraic equations using dummy derivatives,” *SIAM Journal on Scientific Computing*, vol. 14, no. 3, 1993.

[22] B. Bachmann, “Mathematical aspects of object-oriented modeling and simulation,” in *Proceedings of the 5th International Modelica Conference*, 2006.

[23] U. M. Ascher and L. R. Petzold, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, 1998.

[24] A. Isidori, *Nonlinear Control Systems*. Springer-Verlag, 1989.

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

[26] S. A. Bortoff and M. W. Spong, “Pseudolinearization of the acrobot using spline functions,” in *Proceedings of the IEEE Conference on Decision and Control*, 1992, pp. 593–598.

[27] S. Skogestad and I. Postlethwaite, *Multivariable Feedback Control: Analysis and Design*. Wiley, 2005.