Control primitives for robot systems

D. Curtis Deno **Richard M. Murray** **Kristofer S. J. Pister** **S. Shankar Sastry**

Electronics Research Laboratory
Department of Electrical Engineering and Computer Science
University of California Berkeley, CA 94720

Abstract

This paper develops a methodology for description of hierarchical control of robot systems in a manner which is faithful to the underlying mechanics, structured enough to be used as an interpreted language, and sufficiently flexible to encompass a wide variety of systems. We present a consistent set of primitive operations which form the core of a robot system description and control language. This language, motivated by the hierarchical organization of neuromuscular systems, is capable of describing a large class of robot systems under a variety of single level and distributed control schemes.

1 Introduction

Complex robot actions require coordinated motion of multiple robot arms or fingers to manipulate objects and respect physical constraints. As we seek to achieve more of the capability of biological "robots", additional descriptive structures and control schemes are necessary. A major aim of this work is to propose such a specification and control scheme. The ultimate goal of our project is to build a high level task programming environment which is relatively robot independent.

Motivation for a consistent specification and control scheme may be derived from mammalian motor systems. One reason for a hierarchical design involving the spinal, brainstem, and cortical levels is that high level feedback loops respond too slowly for all of motor control to be localized there. Biological control systems operate with contact and joint kinematic constraints as well as actuator redundancy. Neural controllers at the highest levels deal with a restricted set of goal variables which are expanded at the lowest levels into signals involving individual muscles and sensors.

Two directions of emphasis may be used to distinguish robot programming languages: traditional programming languages (perhaps including multitasking), and dynamical systems based descriptions of systems and control structures. More traditional task specification languages (such as VAL II and AML) are characterized by C or Lisp like data structures and syntax, coordinate frame specification and transformation primitives, sensor feedback conditionally controlling program flow, and motion between specified locations achieved through via points and interpolation. In contrast, Brockett's Motion Description Language [3] (MDL) is more closely aligned with dynamical systems theory. MDL employs sequences of triples (u, k, T) to convey trajectory information, feedback control information, and time interval to an extensible Forth/PostScript like interpreter. The scheme described in this paper was inspired partly by descriptions of MDL.

Robotic applications of hierarchical control are exemplified by HIC [4] which manages multiple low level servo loops with a robot programming language from the "traditional" category above. One emphasis of such control schemes concerns distributed processing and interprocess communication.

2 Review of robot dynamics and control

In this section we selectively review the dynamics and control of robot systems. The basic result is that even for relatively complicated robot systems, the equations of motion for the system can be written in a standard form. This point of view has been used by Khatib in his operational space formulation [7] and in some recent extensions [8]. The results presented in this section are direct extensions of those works, although the approach is different.

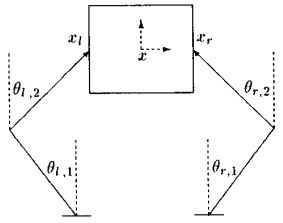

Figure 1: Planar two-fingered hand. Contacts are assumed to be maintained throughout the motion. For this particular system the box position and orientation, x, form a generalized set of coordinates for the system.

The dynamics for a robot manipulator with joint angles $\theta \in \mathbf{R}^n$ and actuator torques $\tau \in \mathbf{R}^n$ can be derived using Lagrange's equations and written in the form

$$M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + N(\theta, \dot{\theta}) = \tau \tag{1}$$

where $M(\theta)$ is a positive definite inertia matrix and $C(\theta, \dot{\theta})\dot{\theta}$ is the Coriolis and centrifugal force vector. The vector $N(\theta, \dot{\theta}) \in \mathbf{R}^n$ contains all friction and gravity terms and the vector $\tau \in \mathbf{R}^n$ represents generalized forces in the θ coordinate frame.

2.1 Constrained manipulators

Constrained robot systems can also be represented by dynamics in the same form as equation (1). As our main example, consider the control of a multi-fingered hand grasping a box (Figure 1) where θ is a vector of all the joint angles and x is a vector describing the position and orientation of the box. The grasping constraint may be written as

$$J(q)\dot{\theta} = G^T(q)\dot{x} \tag{2}$$

where $q = (\theta, x) \in \mathbf{R}^m \times \mathbf{R}^n$, J is the Jacobian of the finger kinematic function and G is the "grasp map" for the system. We will assume that J is bijective in some neighborhood and that G is surjective. This form of constraint can also be used to describe a wide variety of other systems, including grasping with rolling contacts, surface following and coordinated lifting. For the primitives presented in the next section, we also assume that there exists a forward kinematic function between θ and x; that is, the constraint is holonomic.

To include velocity constraints we again appeal to Lagrange's equations. Following the approach in Rosenberg [11], the equations of motion for our constrained system can be written as

$$\tilde{M}(q)\ddot{x} + \tilde{C}(q, \dot{q})\dot{x} + \tilde{N}(q, \dot{q}) = F_e \tag{3}$$

where

$$\begin{aligned} \tilde{M} &= M + GJ^{-T}M_\theta J^{-1}G^T \\ \tilde{C} &= C + GJ^{-T}\left(C_\theta J^{-1}G^T + M_\theta \frac{d}{dt}\left(J^{-1}G^T\right)\right) \end{aligned}$$

$$\begin{aligned} \tilde{N} &= GJ^{-T}N \\ F_e &= GJ^{-T}\tau \\ M, M_\theta &= \text{inertia matrix for the box and fingers, respectively} \\ C, C_\theta &= \text{Coriolis and centrifugal terms} \end{aligned}$$

Thus we have an equation with a similar form to our "simple" robot. In the box frame of reference, $\tilde{M}$ is the mass of the effective mass of the box, and $\tilde{C}$ is the effective Coriolis and centrifugal matrix. These matrices include the dynamics of the fingers, which are being used to actually control the motion of the box. However the details of the finger kinematics and dynamics are effectively hidden in the definition of $\tilde{M}$ and $\tilde{C}$.

2.2 Internal forces

Although the grasp map G was assumed surjective, it need not be square. From the equations of motion (3), we note that if finger-tip force $J^{-T}\tau$ is in the null space of G then the net force in the object frame of reference is zero and causes no net motion of the object. These forces act against the constraint and are generally termed *internal* or *constraint* forces. We can use these internal forces to satisfy other conditions, such as keeping the contact forces inside the friction cone (to avoid slipping) or varying the load distribution of a set of manipulators rigidly grasping an object.

To include the internal forces in our formulation, we extend the grasp map by defining an orthonormal matrix $H(\theta)$ whose rows form a basis for the null space of $G(\theta)$. As before we assume that $G(\theta)$ has constant rank and we break all forces up into an external and an internal piece, F_e and F_i. Given these desired forces, the torques that should be applied by the actuators is

$$\tau = J^T \begin{pmatrix} G \\ H \end{pmatrix}^{-1} \begin{pmatrix} F_e \\ F_i \end{pmatrix} = J^T \begin{pmatrix} G^+ & H^T \end{pmatrix} \begin{pmatrix} F_e \\ F_i \end{pmatrix} \tag{4}$$

2.3 Redundant manipulators

Some manipulators contain more degrees of freedom than are necessary to specify the position of the end effector. Mathematically, these robots can be represented by a change of coordinates $f : \mathbf{R}^m \rightarrow \mathbf{R}^n$ where $m > n$. In this case $J := \frac{\partial f}{\partial \theta}$ is not square and hence J^{-1} is not well defined so the derivation of equation (3) does not hold.

It is still possible to write the dynamics of redundant manipulators in a form consistent with equation (3). To do so, we first define a matrix $K(\theta)$ whose rows span the null space of $J(\theta)$. As before we assume that $J(\theta)$ is full row rank and hence $K(\theta)$ has constant rank $m - n$. The rows of $K(\theta)$ are basis elements for the space of velocities which cause no motion of the end effector; we can thus define an *internal motion*, $\dot{x}_i \in \mathbf{R}^{m-n}$ using the equation

$$\begin{pmatrix} \dot{x}_e \\ \dot{x}_i \end{pmatrix} = \begin{pmatrix} J \\ K \end{pmatrix} \dot{\theta} = \bar{J}\dot{\theta} \tag{5}$$

By *construction*, $\bar{J}$ is full rank (and square) so we can use the previous derivation to conclude that

$$\bar{M}(q) \begin{pmatrix} \ddot{x}_e \\ \ddot{x}_i \end{pmatrix} + \bar{C}(q, \dot{q}) \begin{pmatrix} \dot{x}_e \\ \dot{x}_i \end{pmatrix} + \bar{N}(q, \dot{q}) = F \tag{6}$$

where $\bar{M}$ and $\bar{C}$ are obtained from equation (3) replacing J with $\bar{J}$ and having augmented G with a block diagonal identity matrix to preserve the $\dot{x}_i$'s. If we choose K such that its rows are orthonormal then $\bar{J}^{-1} = (J^+ \; K^T)$ where $J^+ = J^T(JJ^T)^{-1}$ is the least-squares inverse of J. This approach is related to the *extended jacobian* technique for resolving kinematic redundancy [1].

2.4 Control

To illustrate the control of these systems we consider the computed torque control law, an exactly linearizing control law that has been used extensively in robotics research. It has been used for joint level control [2], Cartesian control [10], and most recently, control of multi-fingered hands [9, 5]. Given a desired trajectory x_d we use the control

$$F = M(q)(\ddot{x}_d + K_v\dot{e} + K_p e) + C(q, \dot{q})\dot{x} + N(q, \dot{q}) \tag{7}$$

where error $e = x_d - x$ and K_v and K_p are constant gain matrices. The resulting dynamics equations are linear with exponential rate of convergence determined by K_v and K_p. Since the system is linear, we can use linear control theory to choose the gains (K_v and K_p) such that they satisfy some set of design criteria.

3 Primitives

In this section we describe a set of primitives that gives us the mathematical structure necessary to build a system and control specification for dynamical robot systems. We do not require any particular programming environment or language, borrowing instead freely from languages such as C, Lisp and C++. As much as possible, we have tried to define the primitives so that they can be implemented in any of these languages.

As our basic data structure, we will assume the existence of an object with an associated list of attributes. These attributes can be thought of as a list of name-value pairs which can be assigned and retrieved by name. A typical attribute which we will use is the inertia of a robot. The existence of such an attribute implies the existence of a function which is able to evaluate and return the inertia matrix of a robot given its configuration.

Attributes will be assigned values using the notation attribute := value. Thus we might define our inertia attribute as

$$M(\theta) := \begin{bmatrix} m_1 l_1^2 + m_2 l_2^2 & m_2 l_1 l_2 \cos(\theta_1 - \theta_2) \\ m_2 l_1 l_2 \cos(\theta_1 - \theta_2) & m_2 l_2^2 \end{bmatrix} \tag{8}$$

In order to evaluate the inertia attribute, we would call M with a vector $\theta \in \mathbf{R}^2$. This returns a 2×2 matrix which as defined above. The Coriolis/centrifugal attribute, C, and friction/gravity/nonlinear attribute, N, are defined similarly.

To encourage intuition, we will first describe the actions of the primitives for the case of non-redundant robots. Additionally, we ignore the internal forces that are present in constrained systems. Extensions to these cases are presented in section 5.

3.1 The robot object

The fundamental object used by all primitives is a *robot*. Associated with a robot are a set of attributes which are used to define its behavior:

M	inertia of the robot
C	Coriolis/centrifugal vector
N	friction and gravity vector
rd	return position and force information about the robot
wr	send position and information to the robot

The rd function returns the current position, velocity, and acceleration of the robot, and the forces measured by the robot. Each of these will be a vector quantity of dimension equal to the number of degrees of freedom of the robot. Typically a robot may only have access to its joint positions and velocities, in which case $\ddot{x}$ and F will be *nil*.

The wr function is used to specify an expected position and force trajectory that the robot is to follow. In the simplest case, a robot would ignore everything but F and try to apply this force/torque at its actuators. As we shall see later, other robots may use this information in a more intelligent fashion. We will often refer to the arguments passed to write by using the subscript e. Thus x_e is the expected or desired position passed to the wr function.

The task of describing a primitive is essentially the same as describing how it generates the attributes of the new robot. The following sections describe how each of the primitives generates these attributes. The new attributes created by a primitive are distinguished by a tilde over the name of the attribute.

3.2 DEFINE primitive

Synopsis:

`DEFINE(`M, C, N, rd, wr`)`

The **define** primitive is used to create a simple robot object. It defines the minimal set of attributes necessary for a robot. These attributes are passed as arguments to the **define** primitive and a new

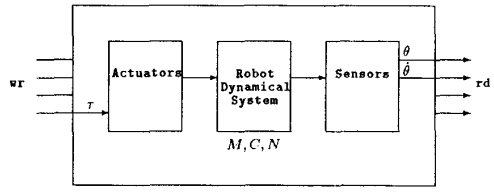

Figure 2: Example of the **define** primitive. The robot shown here corresponds to a robot with torque driven motors and only position and velocity sensing.

robot object possessing those attributes is created:

$$\begin{aligned} \tilde{M}(\theta) &:= M(\theta) \\ \tilde{C}(\theta,\dot{\theta}) &:= C(\theta,\dot{\theta}) \\ \tilde{N}(\theta,\dot{\theta}) &:= N(\theta,\dot{\theta}) \\ \tilde{rd}() &:= rd() \\ \tilde{wr}(\theta_\epsilon,\dot{\theta}_\epsilon,\ddot{\theta}_\epsilon,\tau_\epsilon) &:= wr(\theta_\epsilon,\dot{\theta}_\epsilon,\ddot{\theta}_\epsilon,\tau_\epsilon) \end{aligned}$$

Several different types of robots can be defined using this basic primitive. For example, a DC motor actuated robot would be implemented with a *wr* function which converts the desired torque to a motor current and generates this current by communicating with some piece of hardware (such as a D/A converter). This type of robot system is shown in Figure 2. On the other hand, a stepper motor actuated robot might use a *wr* function which ignores the torque argument and uses the position argument to move the actuator. Both robots would use a *rd* function which returns the current position, velocity, acceleration and actuator torque. If any of these pieces of information is missing, it is up to the user to insure that they are not needed at a higher level. We may also define a *payload* as a degenerate robot by setting the *wr* argument to the *nil* function. Thus commanding a motion and/or force on a payload produces no effect.

3.3 ATTACH primitive

Synopsis:

`ATTACH(J, G, h, payload, robot-list)`

Attach is used to describe constrained motion involving a payload and one or more robots. **Attach** must create a new robot object from the attributes of the payload and of the robots being attached to it. The specification of the new robot requires a velocity relationship between coordinate systems ($J\dot{\theta} = G^T\dot{x}$), an invertible kinematic function relating robot positions to payload position ($x = h(\theta)$), a payload object, and a list of robot objects involved in the contact.

The only difference between the operation of the **attach** primitive and the equations derived for constrained motion of a robot manipulator is that we now have a *list* of robots each of which is constrained to contact a payload. However, if we define θ_R to be the combined joint angles of the robots in **robot-list** and similarly define M_R and C_R as block diagonal matrices composed of the individual inertia and Coriolis matrices of the robots, we have a system which is identical to that presented previously. Namely, we have a "robot" with joint angles θ_R and inertia matrix M_R connected to an object with a constraint of the form

$$J\dot{\theta}_R = G^T\dot{x} \tag{9}$$

where once again J is a block diagonal matrix composed of the Jacobians of the individual robots. To simplify notation, we will define $\mathcal{A} := J^{-1}G^T$ so that

$$\dot{\theta}_R = \mathcal{A}\dot{x} \tag{10}$$

The attributes of the new robot can thus be defined as:

$$\tilde{M} := M_p + \mathcal{A}^T M_R \mathcal{A} \tag{11}$$

$$\tilde{C} := C_p + \mathcal{A}^T C_R \mathcal{A} + \mathcal{A}^T M_R \dot{\mathcal{A}} \tag{12}$$

$$\tilde{N} := N_p + \mathcal{A}^T N_R \tag{13}$$

$$\tilde{rd}() := (h(\theta_R),\ \mathcal{A}^+\dot{\theta}_R,\ \mathcal{A}^+\ddot{\theta}_R + \dot{\mathcal{A}}^+\dot{\theta}_R,\ \mathcal{A}^{+T}\tau_R) \tag{14}$$

$$\tilde{wr}(x_\epsilon,\dot{x}_\epsilon,\ddot{x}_\epsilon,F_\epsilon) := wr_R(h^{-1}(x_\epsilon),\ \mathcal{A}\dot{x}_\epsilon,\ \mathcal{A}\ddot{x}_\epsilon + \dot{\mathcal{A}}\dot{x},\ \mathcal{A}^{+T}F_\epsilon) \tag{15}$$

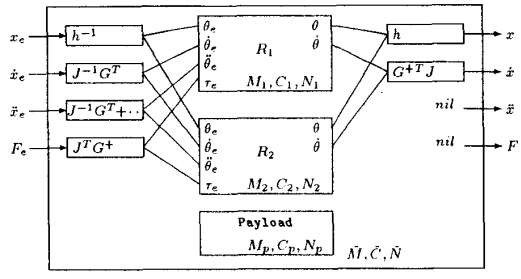

Figure 3: Data flow in a two robot **attach**. In this example we illustrate the structure generated by a call to **attach** with 2 robots and a payload (e.g. a system like Figure 1). The two large interior boxes represent the two robots, with their input and output functions and their inertia properties. The outer box (which has the same structure as the inner boxes) represents the new robot generated by the call to **attach**. In this example the robots do not have acceleration or force sensors, so these outputs are set to *nil*.

where M_p, C_p, N_p are attributes of the payload, M_R and C_R are as described above and N_R is a stacked vector of friction and gravity forces. This construction is illustrated in Figure 3.

The $\tilde{rd}$ attribute for an attached robot is a function which queries the state of all the robots in **robot-list**. Thus θ_R in equation (14) is constructed by calling the individual *rd* functions for all of the robots in the list. The θ values for each of these robots are then combined to form θ_R and this is passed to the forward kinematic function. A similar computation occurs for $\dot{\theta}_R$, $\ddot{\theta}_R$ and τ_R. Together, these four pieces of data form the return value for the $\tilde{rd}$ attribute.

In a dual manner, the $\tilde{wr}$ attribute is defined as a function which takes a desired trajectory (position and force), converts it to the proper coordinate frame and sends each robot the correct portion of the resultant trajectory. A special case of the **attach** primitive is its use with a *nil* payload object and $G = I$. In this case, M_p, C_p, and N_p are all zero and the equations above reduce to a simple change of coordinates.

3.4 CONTROL primitive

Synopsis:

`CONTROL(robot, controller)`

The **control** primitive is responsible for assigning a controller to a robot. It is also responsible for creating a new robot with attributes that properly represents the controlled robot. The attributes of the created robot are completely determined by the individual controller. However, the *rd* and *wr* attributes will often be the same for different controllers. Typically the *rd* attribute for a controlled robot will be the same as the *rd* attribute for the underlying robot. That is, the current state of the controlled robot is equivalent to the current state of the uncontrolled robot. A common *wr* attribute for a controlled robot would be a function which saved the desired position, velocity, acceleration and force in a local buffer accessible to our controller. This configuration is shown in Figure 4.

The dynamic attributes $\tilde{M}$, $\tilde{C}$ and $\tilde{N}$ are determined by the controller. At one extreme, a controller which compensates for the inertia of the robot would set the dynamic attributes of the controlled robot to zero. This does not imply that the robot is no longer a dynamic object, but rather that controllers at higher levels can ignore the dynamic properties of the robot, since they are being compensated for at a lower level. At the other end of the spectrum, a controller may make no attempt to compensate for the inertia of a robot, in which case it should pass the dynamic attributes on to the next higher level. Controllers which lie in the middle of this range may partially decouple the dynamics of the manipulator without actually completely compensating for them. To illustrate these concepts we next consider one possible controller class, computed torque. However, many control laws originally formulated in joint space may also be employed since the structure of equation (3) has been preserved.

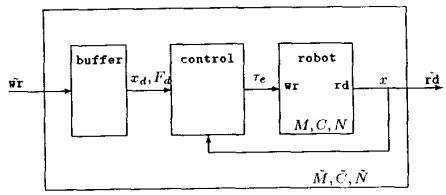

Figure 4: Data flow in a typical controlled robot. Information written to the robot is stored in an internal buffer where it can be accessed by the controller. The controller uses this information and the current state of the robot to generate forces which cause it to follow the desired trajectory.

Computed torque controller

As we mentioned in section 2, the computed torque controller is an exactly linearizing controller which inverts the nonlinearities of a robot to construct a linear system. This linear system has a transfer function equal to the identity map and as a result has no uncompensated dynamics. The proper representation for such a system sets the dynamical attributes $\hat{M}$, $\hat{C}$, and $\hat{N}$ to zero and uses the *rd* and *wr* attributes as described above. We introduce x_d to refer to the buffered desired trajectory.

The control process portion of the controller is responsible for generating input robot forces which cause the robot to follow the desired trajectory (available in x_d). Additionally, the controller must determine the "expected" trajectory to be sent to lower level robots. For the computed torque controller we use the resolved acceleration [10] to generate this path. This allows computed torque controllers running at lower levels to properly compensate for nonlinearities and results in a linear error response. The methodology is similar to that used in determining that the dynamic attributes of the output robot should be zero. The control algorithm is implemented by the following equations:

$$
\begin{aligned}
(x, \dot{x}, \cdot, \cdot) &= rd() \\
\ddot{x}_e &= \ddot{x}_d + K_v(\dot{x}_d - \dot{x}) + K_p(x_d - x) \\
\dot{x}_e &= \int_0^t \ddot{x}_e \\
x_e &= \int_0^t \dot{x}_e \\
F_e &= \hat{M}(q)\ddot{x}_e + \hat{C}(q,\dot{q})\dot{x} + \hat{N}(q,\dot{q}) + F_d \\
& wr(x_e, \dot{x}_e, \ddot{x}_e, F_e)
\end{aligned}
$$

where *rd* and *wr* are attributes of the robot which is being controlled.

Note the existence of the F_d term in the calculation of F_e. This is placed here to allow higher level controllers to specify not only a trajectory but also an force term to compensate for higher level payloads. In essence, a robot which is being controlled in this manner can be viewed as an ideal force generator which is capable of following an arbitrary path.

The computed torque controller defines two new attributes, K_p and K_v, which determine the gains (and hence the convergence properties) of the controller. A variation of the computed torque controller is the feedforward controller, which is obtained by setting $K_p = K_v = 0$. This controller can be used to distribute nonlinear calculations in a hierarchical controller, as we shall see in section 4.

4 Examples

To make the use of the primitives more concrete we present an example of a planar hand grasping a box (Figure 1) using a complicated control

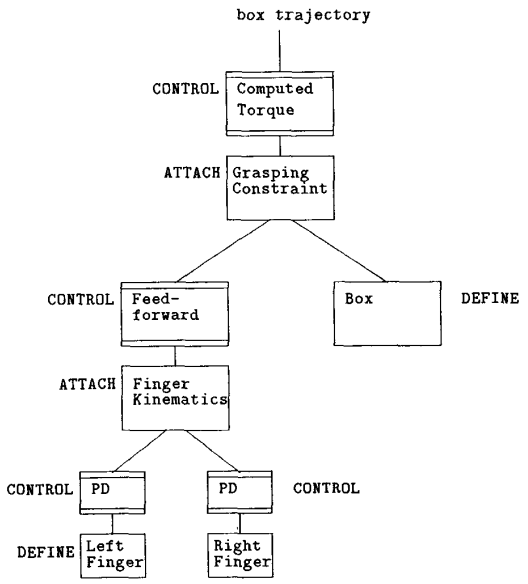

Figure 5: Multi-level computed torque and stiffness (PD). Controllers are used at each level to provide a distributed control system with biological motivation, desirable control properties, and computational efficiency.

structure. We shall assume the existence of the following functions

M_b	box inertia matrix in Cartesian coordinates
M_l, M_r	inertia matrix for the left and right fingers
C_b, C_l, C_r	Coriolis/centrifugal vector for box and fingers
f	finger kinematics function, $f : (\theta_l, \theta_r) \mapsto (x_l, x_r)$
g	grasp kinematics function, $g : (x_l, x_r) \mapsto x_b$
J	finger Jacobian, $J = \frac{\partial f}{\partial \theta}$
G	grasp map, consistent with g
`rd_left, rd_right`	read the current joint position and velocity
`wr_left, wr_right`	generate a desired torque on the joints

where $\theta_l, \theta_r, x_l, x_r$, and x_b are defined as in Figure 1.

Consider the control structure illustrated in Figure 5. This control structure is obtained by analogy with biological systems in which controllers run at several different levels simultaneously. At the lowest level we use simple PD control laws attached directly to the individual fingers. These PD controllers mimic the stiffness provided by muscle coactivation in a biological system [6]. Additionally, controllers at this level might be used to represent spinal reflex actions. At a somewhat higher level, the fingers are attached and considered as a single unit with relatively complicated dynamic attributes and Cartesian configuration. At this point we employ a feedforward controller (computed torque with no error correction) to simplify these dynamic properties, as viewed by higher levels of the brain. With respect to these higher levels, the two fingers appear to be two Cartesian force generators represented as a single composite robot.

Up to this point, the representation and control strategies do not explicitly involve the box, a payload object. These force generators are next attached to the box, yielding a robot with the dynamic properties of the box but capable of motion due to the actuation in the fingers. Finally, we use a computed torque controller at the very highest level to allow us to command motions of the box without worrying about the details of muscle actuation. By this controller we simulate the actions of the cerebellum and brainstem to coordinate motion and correct for errors.

In terms of the primitives that we have defined, we build this structure from the bottom up

```
left  = DEFINE(M_l, C_l, 0, 0, rd_left, wr_left)
pd_left =  CONTROL(left, pd)
right = DEFINE(M_r, C_r, 0, 0, rd_right, wr_right)
pd_right = CONTROL(right, pd)
    fingers = ATTACH(J, I, f, nil, pd_left, pd_right)
    ff_fingers = CONTROL(fingers, feed-forward)
    box  = DEFINE(M_b, C_b, 0, 0, nil, nil)
        hand = ATTACH(I, G, g, box, ff_fingers)
        ct_hand = CONTROL(hand, computed-torque)
```

It is helpful to illustrate the flow of information to the highest level control law. In the evaluation of x_b and $\dot{x}_b$, the following sequence occurs through calls to the *rd* attribute:

hand: asks for current state, x_b and $\dot{x}_b$
  finger: ask for current state, x_f and $\dot{x}_f$
    left: read current state, θ_l and $\dot{\theta}_l$
    right: read current state, θ_r and $\dot{\theta}_r$
  finger: $x_f, \dot{x}_f \leftarrow f(\theta_l, \theta_r), J(\dot{\theta}_l, \dot{\theta}_r)$
hand: $x_b, \dot{x}_b \leftarrow g(x_f), G^T \dot{x}_f$

When we write a set of hand forces using the *wr* attribute, it causes a similar chain of events to occur.

The structure in Figure 5 also has interesting properties from a more traditional control viewpoint. The low level PD controllers can be run at high servo rates (due to their simplicity) and allow us to tune the response of the system to reject high frequency disturbances. The Cartesian feedforward controller permits a distribution of the calculation of nonlinear compensation terms at various levels, lending itself to multiprocessor implementation. Finally, using a computed torque controller at the highest level gives the flexibility of performing the controller design in the task space and results in a system with linear error dynamics.

5 Extensions to the basic primitives

Having presented the primitives for non-redundant robot systems in which we ignore internal forces, we now describe the modifications necessary to include both internal motion and internal forces in the primitives. As before, these extensions are based on the dynamic equations given in Section 2 and rely on the fact that the equations of motion of this class of systems can be expressed in a unified way.

Internal motion and force can be thought of manifestations of redundancies in the manipulator, and both can be used to improve performance. A classical use of redundant motion in robotics is to specify a cost function and use the redundancy of the manipulator to attempt to minimize this cost function. If we extend our definition of the *wr* function so that it takes not only an "external" trajectory, but also an internal trajectory (which might be represented as a cost function or directly as a desired velocity in the internal motion directions) then this internal motion can be propagated down the graph structure. A similar situation occurs with internal or constraint forces.

The matrices $J(q)$ and $G(q)$ in equation (2) embody the fundamental properties of the constrained system. We begin by assuming that $J(q)$ and $G(q)$ are both full row rank. The null space of $J(q)$ corresponds to motions which do not affect the configuration of the object, i.e., internal motions. Likewise, the null space of $G(q)$ describes internal forces—the set of forces which cause no motion of the object. A complete trajectory for a robot must specify not only external motion and force for a robot but also the internal motion and force which lie in these subspaces.

5.1 Internal forces

To allow internal forces to be specified and controlled, we must first add them to the *rd* and *wr* attributes. This is done by simply adding an extra value to the list of values returned by *rd* and adding an extra argument to *wr*. Thus the *wr* attribute is called as

$$wr(x_e, \dot{x}_e, \ddot{x}_e, F_e, F_i) \tag{16}$$

where F_i is the desired internal force.

Internal forces are "created" by the **attach** primitive. The internal force directions for a constraint are represented by an orthonormal matrix $H(\theta)$ whose rows form a basis for the null space of $G(\theta)$. Since any of the daughter robots may itself have an internal force component, the internal force vector for a robot created by attach consists of two pieces: the internal forces created by this constraint and the combined internal forces for the daughter robots. We shall refer to these two components as $F_{i,1}$ and $F_{i,2}$, respectively. The force transformations which describe this relationship are

$$\tau_R = \begin{pmatrix} \tau_{R,e} \\ \hline \tau_{R,i} \end{pmatrix} = \left(\begin{array}{cc|c} J^T G^+ & J^T H^T & 0 \\ \hline 0 & 0 & I \end{array}\right) \begin{pmatrix} F_e \\ \hline F_{i,1} \\ F_{i,2} \end{pmatrix} \tag{17}$$

where $\tau_{R,e}$ is the vector of external forces for the daughter robots and $\tau_{R,i}$ is the vector of internal forces. This equation is analogous to equation (4) in Section 2.2. Note that $\tau_{R,i}$ is identical to $F_{i,2}$, thus internal force specifications required by the daughter robots are appended to the internal force specification required due to the constraint. Expanding equation (17) we see the appropriate definition for the new *wr* attribute generated by **attach** is

$$\tilde{wr}(x_e, \dot{x}_e, \ddot{x}_e, F_e, F_i) := wr_R(\cdots, J^T G^+ F_e + J^T H^T F_{i,1}, F_{i,2}) \tag{18}$$

The inclusion of internal forces in the *rd* attribute is similar. The sensed forces from the robots, τ_R, are simply split into external and internal components and converted to the appropriate internal and external forces for the new robot. This is equivalent to inverting equation (17):

$$F = \begin{pmatrix} F_e \\ \hline F_{i,1} \\ F_{i,2} \end{pmatrix} = \left(\begin{array}{cc|c} GJ^{-T} & & 0 \\ HJ^{-T} & & 0 \\ \hline 0 & & I \end{array}\right) \begin{pmatrix} \tau_{R,e} \\ \hline \tau_{R,i} \end{pmatrix} \tag{19}$$

It follows that

$$\tilde{rd}() := (\cdots, GJ^{-T}\tau_{R,e}, \begin{pmatrix} HJ^{-T}\tau_{R,e} \\ \tau_{R,i} \end{pmatrix}) \tag{20}$$

Internal forces are resolved by the **control** primitive. In principle, a controller can specify any number of the internal forces for a robot. Internal forces which are not resolved by a controller are left as internal forces for the newly defined robot. In practice, controllers will often be placed immediately above the attached robots since internal forces are best interpreted at this level. Unlike external motions and forces, internal forces are not subject to coordinate change and so leaving such forces unresolved forces higher level controllers to use low level coordinates.

5.2 Internal motions

Internal motions are also created by the **attach** primitive, this time due to a non-square Jacobian matrix. As before, we must add arguments to the *rd* and *wr* attributes of robots to handle the extra information necessary for motion specification. We only assume that the redundant velocities and accelerations are defined, so we add only those quantities to *rd* and *wr*. Since the notation becomes quite cumbersome, we won't actually define the *rd* and *wr* primitives, but just specify the internal and external motion components.

Given a constraint which contains internal motions, the **attach** primitive must again properly split the motion among the robots attached to the object. Define $K(\theta)$ to be a matrix whose rows span the null space of $J(\theta)$. Then we can rewrite our constraint as

$$\left(\begin{array}{c|c} J & 0 \\ K & 0 \\ \hline 0 & I \end{array}\right) \begin{pmatrix} \dot{\theta}_{R,e} \\ \hline \dot{\theta}_{R,i} \end{pmatrix} = \left(\begin{array}{ccc|c} G^T & 0 & 0 \\ 0 & I & 0 \\ \hline 0 & 0 & I \end{array}\right) \begin{pmatrix} \dot{x}_e \\ \hline \dot{x}_{i,1} \\ \dot{x}_{i,2} \end{pmatrix} \tag{21}$$

Defining $\bar{J}$ and $\bar{G}$ as the extended Jacobian and grasp matrices,

$$\bar{J} = \begin{pmatrix} J \\ K \end{pmatrix} \qquad \bar{G}^T = \begin{pmatrix} G^T & 0 \\ 0 & I \end{pmatrix} \tag{22}$$

we see that $\bar{J}$ is full rank and so we can use it to define $\mathcal{A} = \bar{J}^{-1}\bar{G}^T$ in equations (11–15). This then defines the dynamics attributes created by **attach**. Note that the dimension of the constrained subspace (where internal forces act) is unchanged by this extension.

The input and output attributes are described in a manner similar to those used for internal forces. For wr the external component of the motion is given by

$$\dot{\theta}_{R,e} = \mathcal{A}\begin{pmatrix} \dot{x}_e \\ \dot{x}_i \end{pmatrix} \quad (23)$$

$$= J^+ G^T \dot{x}_e + K^T \dot{x}_i \quad (24)$$

$\ddot{\theta}_{R,e}$ is defined similarly. $\theta_{R,e}$ is only defined if an inverse kinematic function, h^{-1}, is given. Otherwise that information is not passed to the daughter robots. As before, if the robots themselves have internal motions then these should be split off and passed unchanged to the lower level robots.

The rd attribute is defined by projecting robot motions into an object motion component and an internal motion component. That is

$$x_e = h(\theta_{R,e}) \quad (25)$$

$$\dot{x}_e = G^+ J \dot{\theta}_{R,e} \quad (26)$$

$$\dot{x}_i = \begin{pmatrix} K\dot{\theta}_{R,e} \\ \dot{\theta}_{R,i} \end{pmatrix} \quad (27)$$

$\ddot{x}_e$ and $\ddot{x}_i$ are obtained by differentiating the expression for $\dot{x}_e$ and $\dot{x}_i$.

Controllers must also be extended to understand redundant motion. This is fundamentally no different than control of an ordinary manipulator except that position information is not available in redundant directions. Thus the computed torque law would become

$$F = M(q)\begin{pmatrix} \ddot{x}_{e,d} + K_v\dot{e}_e + K_p e_e \\ \ddot{x}_{i,d} + K_v\dot{e}_i \end{pmatrix} + C(q,\dot{q})\begin{pmatrix} \dot{x}_e \\ \dot{x}_i \end{pmatrix} + N(q,\dot{q}) \quad (28)$$

Motion specification for such a control law would be in terms of a position trajectory $x_e(\cdot)$ and a velocity trajectory $\dot{x}_i(\cdot)$. If a controller actually resolves the internal motion (by specifying $\dot{x}_{i,d}(\cdot)$ based on a pseudo inverse calculation for example), then the internal motion will be masked from higher level controllers; otherwise it is passed on.

Control laws commonly use the position of the object as part of the feedback term. This may not always be available for systems with non-integrable constraints (such as grasping with rolling contacts). If the object position cannot be calculated from θ then we must retrieve it from some other source. One possibility is to use an external sensor which senses x directly, such as a camera or tactile array. The function to "read the sensor" could be assigned to the payload rd function and `attach` could use this information to return the payload position when queried. Another possible approach is to integrate the object velocity (which is well defined) to bookkeep the payload position.

Some care must also be taken with the evaluation of dynamic attributes for robots which do not have well defined inverse kinematic functions. There are some robot control laws which use feedforward terms that depend on the desired output trajectory, e.g., $M(x_d)\ddot{x}_d$. The advantage of writing such control laws is that this expression can be evaluated offline, increasing controller bandwidth. This calculation only makes sense if the desired configuration, q_d, can be written as a function of x_d and more generally if q can be written as a function of x. One solution to this problem is to only evaluate dynamic attributes of a robot at the current configuration. Assuming each robot in the system can determine its own position, these attributes are then well defined. For all the control laws presented in this paper, M, C and N are always evaluated at q, the current configuration.

6 Discussion

Working from a physiological motivation we have developed a set of robot description and control primitives consistent with Lagrangian dynamics. Starting from a description of the inertia, sensor, and actuator properties of individual robots, these primitives allow for the construction of a composite constrained motion system with control distributed at all levels. Robots, as dynamical systems, are recursively defined in terms of daughter robots. The resulting hierarchical system can be represented as a tree structure in a graph theoretic formalism, with sensory data fusion occurring as information flows from the leaves of the tree (individual robots and sensors) toward the root, and data expansion as relatively simple motion commands at the root of the tree flow down through contact constraints and kinematics to the individual robot actuators.

One of the major future goals of this research is to implement the primitives presented here on a real system. This requires that efforts be made toward implementing primitives in as efficient fashion as possible. The first implementation choice is deciding when computation should occur. It is possible that the entire set of primitives could be implemented off-line. In this case, a controller-generator would read the primitives and construct suitable code to control the system. A more realistic approach is to split the computation burden more judiciously between on-line and off-line resources. Symbolically calculating the attributes of the low level robots and storing these as precompiled functions might enable a large number of systems to be constructed using a library of daughter robot systems. Although the expressions employed are continuous time, in practice digital computers will be relied upon for discrete time implementations. This raises the issue of whether lower computation rates may be practical for higher level robots/controllers.

In addition to implementation issues, there are still several theoretical issues which we hope to address. We would like to have stability proofs for classes of control hierarchies, e.g. any hierarchy with a computed torque controller at the highest level and only feedforward controllers below it can be shown to be exponentially stable. There is also no provision in the primitives for dynamics which can not be written in the form of equation (1). Adaptive identification and control techniques may be useful in cases where unmodeled dynamics substantially affect system performance.

Acknowledgements

D.C. Deno was supported by NEI EY05913, the Smith-Kettlewell Eye Research Foundation, and the Rachael C. Atkinson Fellowship Award. K.S.J. Pister, R.M. Murray, and S.S. Sastry were supported in part by the NSF under grant numbers DMC 84-51129, ECS 87-19298 and the Air Force Office of Scientific Research (AFSC) under grant number F49620-87-C0041. R.M. Murray was also supported in part by an IBM Manufacturing Fellowship.

References

[1] J. Baillieul. Kinematic programming alternatives for redundant manipulators. In *IEEE ICRA*, pages 722–728, 1985.

[2] A. K. Bejczy. Robot arm dynamics and control. Technical Report 33-699, Jet Propulsion Laboratory, 1974.

[3] Roger W. Brockett. On the computer control of movement. Technical Report CICS-P-31, Center for Intelligent Control Systems, Harvard Univ., November 1987.

[4] Dayton Clark. IHC: An operating system for hierarchies of servo loops. In *IEEE ICRA*, pages 1004–1009, 1989.

[5] A. Cole, J. Hauser, and S. Sastry. Kinematics and control of multifingered hands with rolling contact. In *IEEE ICRA*, pages 228–233, 1988.

[6] Neville Hogan, Emilio Bizzi, F. A. Mussa-Ivaldi, and Tamar Flash. Controlling multijoint behavior. *Exercise and Sport Sciences Reviews*, 15:153–190, 1987.

[7] O. Khatib. A unified approach for motion and force control of robot manipulators: The opertional space formulation. *IEEE J. Robotics and Automation*, RA-3(1):43-53, February 1987.

[8] O. Khatib. Augmented object and reduced effective inertia in robot systems. *Amer. Control Conf.*, pages 2140–2147, 1988.

[9] Z. Li, P. Hsu, and S. Sastry. On kinematics and control of multifingered hands. In *IEEE ICRA*, pages 384–389, 1988.

[10] J. Y. S. Luh, M. W. Walker, and R. P. Paul. Resolved acceleration control of mechanical manipulators. *IEEE Trans. Autom. Control*, AC-25, 1980.

[11] R. M. Rosenberg. *Analytical Dynamics of Discrete Systems*. Plenum Press, New York, 1977.