Control Primitives for Robot Systems

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

Abstract—**A set of primitive operations is presented that forms the core of a robot system description and control language. The actions of the individual primitives are derived from the mathematical structure of the equations of motion for constrained mechanical systems. The recursive nature of the primitives allows composite robots to be constructed from more elementary daughter robots. A few pertinent results of classical mechanics are reviewed, the functionality of our primitive operations is described, and several different hierarchical strategies for the description and control of a two-fingered hand holding a box are presented.**

I. Introduction

The complexity of compound, redundant robotic systems, both in specification and control, continues to present a challenge to engineers and biologists. 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 that is relatively robot independent. In this paper, we describe a language for constructing hierarchical controllers for complex robot systems. Our primary example is a multifingered robot hand. A typical task for such a system might entail moving to an object, grasping and manipulating the object, and using the object to perform a higher level task. Each of these phases of the task requires a separate controller that is compatible with the constraints on the system and the task objective. The primitives described in this paper allow these controllers to be constructed in a simple and organized fashion. Furthermore, as a consequence of the recursive nature of the primitives, it is possible to introduce a degree of device independence in constructing higher level controllers.

This paper is organized as follows: In Section II we review the dynamics and control of coupled, constrained rigid robots in a Lagrangian framework. Section III contains definitions of the primitives of our robot control environment. Section IV illustrates the application of our primitives to the description of a two-fingered robot hand. We show that our environment can be used to specify a variety of control schemes for this hand, including a distributed controller that has a biological analog. Section V extends the basic primitives to include specification and control of constraint forces and redundant motion. In Section VI we discuss future avenues of research. The remainder of this introduction presents motivation and background for our work, and an overview of the primitives we have chosen to use.

Manuscript received September 1, 1990; revised March 7, 1991 and June 17, 1991. This work was supported in part by NEI EY05 913, in part by the Smith-Kettlewell Eye Research Foundation, in part by the Rachael C. Atkinson Fellowship Award, in part by the NSF under Grant DMC 84-51129 and Grant ECS 87-19298, in part by the Air Force Office of Scientific Research (AFSC) under Grant F49620-87-C0041, and in part by an IBM Manufacturing Fellowship.

R. M. Murray was with the University of California, Berkeley, CA 94720. He is now with the Department of Mechanical Engineering, California Institute of Technology, Pasadena, CA 91125.

D. C. Deno, K. S. J. Pister, and S. S. Sastry are with the Electronics Research Laboratory, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720.

IEEE Log Number 9104113.

A. The Musculoskeletal System: Metaphor for a Robotic System

Motivation for a consistent specification and control scheme may be sought in our current knowledge of the hierarchical organization of mammalian motor systems. To some degree of accuracy, we may consider segments of limbs as rigid bodies connected by rotary joints. Muscles and tendons are actuators with sensory feedback that enter into low-level feedback control at the spinal level [12]. Further up the nervous system, the brainstem, cerebellum, thalamus, and basal ganglia integrate ascending sensory information and produce coordinated motor commands. At the highest levels, sensory and motor cortex supply conscious goal-related information, trajectory specification, and monitoring.

The hierarchical structure of neuromuscular control is also evident from differences in time scale. The low-level spinal reflex control runs faster (loop delays of about 30 ms) than the high-level feedback loops (100–200 ms delays). This distinction may be exploited by control schemes that hide information details from high-level controllers by virtue of low-level control enforcing individual details. These concepts are shown in Fig. 1 where a drawing of neuromuscular control structures for a finger is juxtaposed with a block diagram to emphasize the hierarchical nature of the thumb-forefinger system for picking up objects.

Biological control systems commonly operate with constraints and redundancy. Kinematic constraints arise not only from joints that restrict the relative motion of adjacent limb segments, but also from contact with objects that leads to similar restrictions. Many musculoskeletal subsystems possess kinematic and actuator redundancy that may be imagined to be resolved by effort and stability considerations. In any event, the neural controller directs a specific strategy and so expands a reduced set of control variables into the larger complete set.

In the sequel we shall see these concepts expressed in a notation that is faithful to the laws of mechanics and flexible enough to permit concise descriptions of robot motion control at various hierarchical levels.

B. Background

The robotics and control literature contains a number of topics that are related to the specification and control scheme of this paper.

Robot Programming Languages: 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 include VAL II, AML, and Robot-BASIC [7], [25], [11], [26]. These languages are characterized by C-, BASIC-, 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 a two stage hierarchy, low-level controllers usually control joint angle trajectories that are specified by high-level language statements and kinematics computations.

Brockett's motion description language (MDL) [3], [10] 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

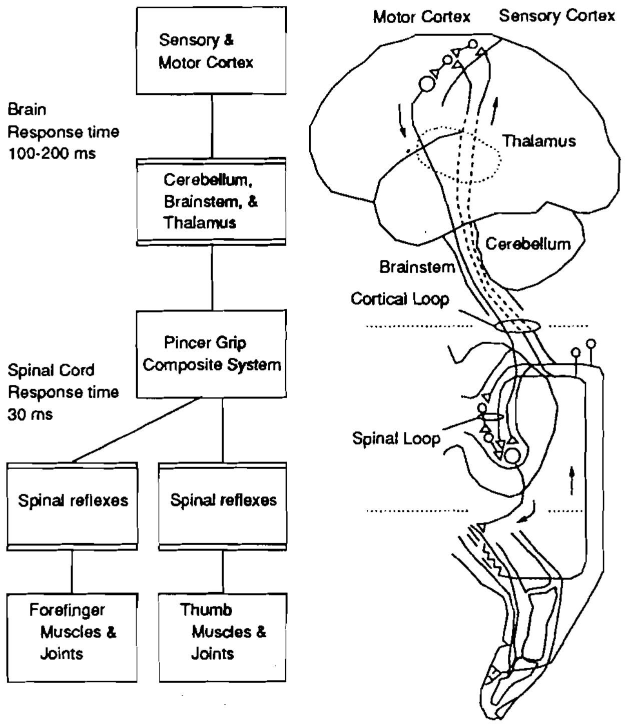

Fig. 1. Hierarchical control scheme of a human finger. At the highest level, the brain is represented as sensory and motor cortex (where sensory information is perceived and conscious motor commands originate) and brainstem and cerebellar structures (where motor commands are coordinated and sent down the spinal cord). A pair of fingers forms a composite system for grasping that is shown integrated at the level of the spinal cord. The muscles and sensory organs of each finger form low-level spinal reflex loops. These low-level loops respond more quickly to disturbances than sensory-motor pathways that travel to the brain and back. Brain and spinal feedback controllers are represented by double lined boxes.

Forth/PostScript like interpreter. The scheme described in this paper was inspired partly by descriptions of MDL. Our work explicitly utilizes geometric and inertial parameters together with the equations of motion to describe the organization and control of complex robots. MDL is less explicit on this matter but is more completely developed in the matter of sequences of motions.

An object oriented approach similar to that presented here has been described by Cutkosky, Howe, and Witkin [8]. They present a method for describing the dynamics of a robot hand grasping an object. The advantage of the method they propose is that it is very general and does not rely on rigid contact models, allowing compliant fingertips to be considered. Their method is closely related to graph theoretic methods in mechanics that keep track of generalized forces and displacements along branches of a graph representing the system interconnection. The main emphasis of their approach is on system description rather than controller design. It is precisely because we are interested in designing controllers that we have initially limited the class of interconnections we are allowed.

Distributed Control, Hierarchical Control: The nervous system controls biomechanical robots using both distributed controllers and hierarchical organization [12]. For example, spinal reflex centers can direct portions of gait in cats and the wiping motions of frog limbs without the brain. One reason for a hierarchical design is that high-level feedback loops may respond too slowly for all of motor control to be localized there. Indeed the complexity and time delays inherent in biological motor control led the Russian psychologist Bernstein to conclude the brain could not achieve motor control by an internal model of body dynamics [13].

Centralized control has been defined as a case in which every sensor's output influences every actuator. Decentralized control was a popular topic in control theory in the late 1970's and led to a number of results concerning weakly coupled systems and multirate controllers [29]. Graph decomposition techniques, applied to the graph structures employed in a hierarchical scheme, permitted the isolation of sets of states, inputs, and outputs that were weakly coupled. This decomposition facilitated stability analyses and controller design. Robotic applications of hierarchical control are exemplified by HIC [5], 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.

C. Overview of Robot Control Primitives

The fundamental objects in our robot specification environment are objects called robots. In a graph theoretic formalism they are nodes of a tree structure. At the lowest level of the tree are leaves that are instantiated by the `define` primitive. Robots are dynamical systems that are recursively defined in terms of the properties of their daughter robot nodes. Inputs to robots consist of desired positions and conjugate forces. The outputs of a robot consist of actual positions and forces. Robots also possess attributes such as inertial parameters and kinematics.

There are two other primitives that act on sets of robots to yield new robots, so that the set of robots is closed under these operations. These primitives (`attach` and `control`) may be considered as links between nodes and result in composite robot objects. Thus nodes closer to the root may possess fewer degrees of freedom, indicating a compression of information upon ascending the tree.

The `attach` primitive reflects geometrical constraints among variables and in the process of yielding another robot object, accomplishes coordinate transformations. `Attach` is also responsible for a bidirectional flow of information: expanding desired positions and forces to the robots below, and combining actual position and force information into an appropriate set for the higher level robot. In this sense the state of the root robot object is recursively defined in terms of the states of the daughter robots.

The `control` primitive seeks to direct a robot object to follow a specified "desired" position/force trajectory according to some control algorithm. The controller applies its control law (several different means of control are available such as PD and computed torque) to the desired and actual states to compute expected states for the daughter robot to follow. In turn, the daughter robot passes its actual states through the controller to robot objects further up the tree.

The block diagram portion of Fig. 1 may be seen to be an example of a robot system comprised of these primitives. Starting from the bottom: two fingers are `define`d; each finger is `control`led by muscle tension/stiffness and spinal reflexes; the fingers are `attach`ed to form a composite hand; the brainstem and cerebellum help `control` and coordinate motor commands and sensory information; and finally at the level of the cortex, the fingers are thought of as a pincer that engages in high-level tasks such as picking.

The language primitives presented in this paper are intended to codify the description and control of hierarchically organized robots in contact with the environment. This is useful both as an analytic procedure to explore complex control laws, as well as a practical tool to implement control laws at different levels in hierarchical systems—where many different control laws may be needed to accomplish a task. In particular, we are very interested in systems for which the elementary robots remain unchanged, but the system constraints are variable. Such an environment is present, for example, in the control of a multi-fingered robot hand picking up an object.

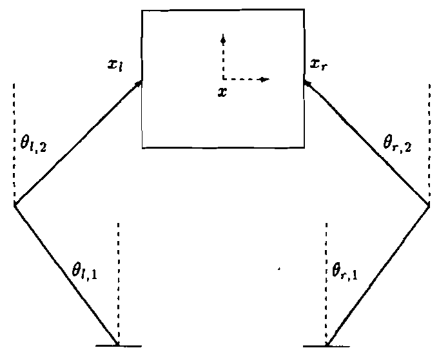

Fig. 2. Planar two-fingered hand. Contacts are assumed to be maintained throughout the motion. For this particular system the box position and orientation, x, form a generalized set of coordinates for the system.

In that situation, separate controllers are needed for the different phases of motion required of the hand: free space motion of the hand, grasping of the object, and manipulation of the grasped object.

II. 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 [16] and in some recent extensions [17]. The results presented in this section are direct extensions of those works, although the approach is different.

The dynamics for a robot manipulator with joint angles $\theta \in \mathbb{R}^n$ and actuator torques $\tau \in \mathbb{R}^n$ 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 $N(\theta,\dot{\theta}) \in \mathbb{R}^n$ contains all friction and gravity terms and the vector $\tau \in \mathbb{R}^n$ represents generalized forces in the θ coordinate frame. For systems of this type, it can be shown that $\dot{M} - 2C$ is a skew symmetric matrix with proper choice of C (such as that in [31]).

A. Constrained Manipulators

Constrained robot systems can also be represented by dynamics in the same form as (1). As our main example, consider the control of a multifingered hand grasping a box (Fig. 2) where $\theta \in \mathbb{R}^n$ is a vector of all the joint angles and $x \in \mathbb{R}^p$ is a vector describing the position and orientation of the box. In most circumstances $p = 3$ or 6 depending on whether we consider planar or spatial motion. The contacts constrain the relative velocities between fingertips and object, dependent on the type of contact model. The grasping constraint may be written as

$$J(q)\dot{\theta} = G^T(q)\dot{x} \qquad \begin{array}{l} J(q) \in \mathbb{R}^{m\times n} \\ G(q) \in \mathbb{R}^{p\times m} \end{array} \tag{2}$$

where $q = (\theta, x) \in \mathbb{R}^n \times \mathbb{R}^p$, J is the Jacobian of the finger kinematic function and G is the "grasp map" for the system. Here m is the number of velocity constraints imposed by the grasp. 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. A more complete derivation of grasping kinematics can be found in [24] or [20].

We make a number of assumptions to simplify the initial analysis. We will assume that J is bijective in some neighborhood and that G is surjective. For the primitives presented in the next section, we also assume that there exists a forward kinematic function $h\colon \theta \mapsto x$; that is, the constraint is holonomic. We discuss the consequences of removing these assumptions in later sections.

To include velocity constraints we again appeal to Lagrange's equations. Following the approach in Rosenberg [27], 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

$$\begin{aligned} \tilde{M} &= M + GJ^{-T}M_\theta J^{-1}G^T \\ \tilde{C} &= C + GJ^{-T}\left(C_\theta J^{-1}G^T + M_\theta (d/dt)\left(J^{-1}G^T\right)\right) \\ \tilde{N} &= GJ^{-T}N_\theta + N \\ F_e &= GJ^{-T}\tau \end{aligned} \tag{4}$$

M, M_θ inertia matrix for the box and fingers, respectively

C, C_θ Coriolis and centrifugal terms

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 $\tilde{C}$. The skew symmetry of $\dot{\tilde{M}} - 2\tilde{C}$ is preserved by this transformation.

B. 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 fingertip 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 $G(\theta)$ is full row 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 are

$$\tau = J^T \begin{pmatrix} G \\ H \end{pmatrix}^{-1} \begin{pmatrix} F_e \\ F_i \end{pmatrix} = J^T \begin{pmatrix} G^+ & H^T \end{pmatrix} \begin{pmatrix} F_e \\ F_i \end{pmatrix} \qquad \begin{array}{l} F_e \in \mathbb{R}^p \\ F_i \in \mathbb{R}^{m-p} \end{array} \tag{5}$$

C. 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 $f : \mathbb{R}^m \to \mathbb{R}^n$ where $m > n$. In this case, the Jacobian matrix $J := \partial f)/\partial\theta$ is not square and hence J^{-1} is not well defined so the derivation of (4) does not hold.

It is still possible to write the dynamics of redundant manipulators in a form consistent with (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 that cause no motion of the end effector; we can thus define an *internal motion*, $\dot{x}_i \in \mathbb{R}^{m-n}$ using the equation

$$\begin{pmatrix} \dot{x}_e \\ \dot{x}_i \end{pmatrix} = \begin{pmatrix} J \\ K \end{pmatrix} \dot{\theta} = \bar{J}\dot{\theta} \tag{6}$$

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

$$\tilde{M}(q)\begin{pmatrix} \ddot{x}_e \\ \ddot{x}_i \end{pmatrix} + \tilde{C}(q,\dot{q})\begin{pmatrix} \dot{x}_e \\ \dot{x}_i \end{pmatrix} + \tilde{N}(q,\dot{q}) = F_e \tag{7}$$

where $\tilde{M}$, $\tilde{C}$, $\tilde{N}$, and F_e are obtained from (4) replacing J with $\bar{J}$ and augmenting 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 inverseof J. This approach is related to the *extended Jacobian* technique for resolving kinematic redundancy [1].

It might appear from our notation that we have parameterized the internal motion of the system by a variable x_i. This is not necessarily the case since only $\dot{x}_i$ was defined in (6). Because we chose K only to span the null space of J, there may not exist a function g such that $x_i = g(\theta)$ and $\partial g/\partial\theta = K$. A necessary and sufficient condition for such a g to exist is that each row of K satisfy $\partial K_{ij}/\partial\theta_k = \partial K_{ik}/\partial\theta_j$. This is merely the statement that mixed partials of g must commute.

It may not always be easy to choose $K(\theta)$ such that it is the differential of some function. For this reason, we shall not generally assume that an explicit coordinatization of the internal motion manifold is available. A more detailed discussion of this point can be found in [22].

D. Control

To illustrate the control of robot systems, we look at two controllers that have appeared in the robotics literature. We start by considering systems of the form

$$M(q)\ddot{x} + C(q,\dot{q})\dot{x} + N(q,\dot{q}) = F \tag{8}$$

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

For the case of more complicated manipulators, the dynamics look essentially the same with appropriate definition of x and F. For redundant manipulators we define x as (x_e, x_i), where it is understood that derivatives of x_i may be used in a control law, but x_i should not appear unless a true coordinatization of the internal motion has been chosen. This is point illustrated in more detail below.

For manipulators that contain constraints, we define F as F_e, cf. (3) and we add an internal force F_i to the applied actuator forces. Thus

$$\tau = J^{-T}G^+F_e + J^{-T}F_i$$

(assuming J is either full rank or has been extended as described previously). For simplicity, we omit discussion of the choice of internal forces (see [19], [23] for details and references). Both internal motions and forces are specified in terms of the basis vectors for the appropriate null spaces.

Computed Torque: Computed torque is an exactly linearizing control law that has been used extensively in robotics research. It has been used for joint level control [2], Cartesian control [21], and most recently, control of multifingered hands [19], [6]. Given a desired trajectory x_d we use the control

$$F = M(q)(\ddot{x}_d + K_v\dot{e} + K_pe) + C(q,\dot{q})\dot{x} + N(q,\dot{q}) \tag{9}$$

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.

The disadvantage of this control law is that it is not easy to specify the interaction with the environment. We might think that we could use K_p to model the stiffness of the system and exert forces by commanding trajectories that result in fixed errors. Unfortunately this is not uniformly applicable as can be seen by examining the force due to a quasi-static displacement Δx:

$$\Delta F = M(q)K_p\Delta x \tag{10}$$

Since K_p must be constant in order to prove stability, the resultant stiffness will vary with configuration. Additionally, given a desired stiffness matrix it may not be possible to find a positive definite K_p that achieves that stiffness, even at a fixed value of q (the product of positive definite matrices is not necessarily positive definite).

PD + Feedforward Control: PD controllers differ from computed torque controllers in that the desired stiffness (and potentially damping) of the end effector is specified, rather than its position tracking characteristics. Typically, control laws of this form rely on the skew-symmetric property of robot dynamics, namely $\alpha^T\left(\dot{M} - 2C\right)\alpha = 0$ for all $\alpha \in \mathbb{R}^n$. Consider the control law

$$F = M(q)\ddot{x}_d + C(q,\dot{q})\dot{x}_d + N(q,\dot{q}) + K_v\dot{e} + K_pe \tag{11}$$

where K_v and K_p are symmetric positive definite. Using a Lyapunov stability argument, it can be shown that the actual trajectory of the robot converges to the desired trajectory asymptotically [18]. Extensions to the control law result in exponential rate of convergence [30], [28].

This PD control law has the advantage that for a quasi-static change in position Δx the resulting force is

$$\Delta F = K_p\Delta x \tag{12}$$

and thus we can achieve an arbitrary symmetric stiffness. Experimental results indicate that the trajectory tracking performance of this control law does not always compare favorably with the computed torque control law [23]. Additionally, there is no simple design criteria for choosing K_v and K_p to achieve good tracking performance. While the stability results give necessary conditions for stability they do not provide a method for choosing the gains. Nonetheless, PD control has been used effectively in many robot controllers and has some computational features that make it an attractive alternative.

Control of Redundant Manipulators: Since the dynamics for a redundant manipulator have the same form as our canonical robot system, it is easy to extend the previous control laws to handle this case. If a coordinatization of the internal motion manifold is available, the control laws are identical with the addition of the redundant states. If we do not have a set of coordinates for the internal motion, but rather, only the velocities, then we must be slightly more careful. For example, the computed torque control law becomes

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

Motion specification for such a control law would be in terms of a position trajectory $x_{e,d}(\cdot)$ and a velocity trajectory $\dot{x}_{i,d}(\cdot)$.

This control law will guarantee tracking of the given internal velocity. One method of calculating this velocity is to attempt to use the redundant degrees of freedom to minimize a cost function. Given the gradient of the cost function, we can project this vector (in joint space) onto the range of $K(\theta)$, which spans the internal motion directions. This determines $\dot{x}_{i,d}$, which can be passed to the controller. This type of cost criterion has been used with a computed torque-like controller by Hsu *et al.* [15].

III. 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 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 that can be assigned and retrieved by name. A typical attribute that we will use is the inertia of a robot. The existence of such an attribute implies the existence of a function that 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

$$M(\theta) := \begin{bmatrix} m_1 l_1^2 + m_2 l_2^2 & m_2 l_1 l_2 \cos(\theta_1 - \theta_2) \\ m_2 l_1 l_2 \cos(\theta_1 - \theta_2) & m_2 l_2^2 \end{bmatrix} \tag{13}$$

In order to evaluate the inertia attribute, we would call M with a vector $\theta \in \mathbb{R}^2$. This returns a 2×2 matrix as defined previously. 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 nonredundant robots. Additionally, we ignore the internal forces that are present in constrained systems. Extensions to these cases are presented in Section V.

A. The Robot Object

The fundamental object used by all primitives is a *robot*. Associated with a robot are a set of attributes that 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 force 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 wr by using the subscript e. Thus $\ddot{x}_e$ 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.

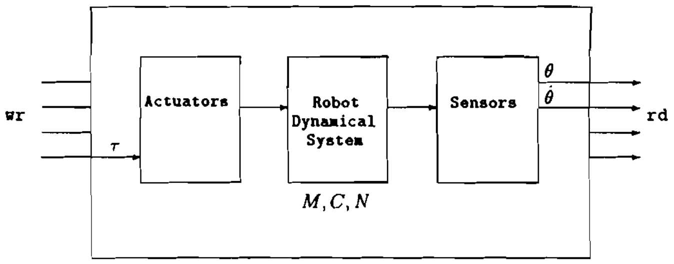

Fig. 3. Example of the `define` primitive. The robot shown here corresponds to a robot with torque driven motors and only position and velocity sensing.

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 robot object possessing those attributes is created:

$$\begin{aligned} \tilde{M}(\theta) &:= M(\theta) \\ \tilde{C}(\theta,\dot{\theta}) &:= C(\theta,\dot{\theta}) \\ \tilde{N}(\theta,\dot{\theta}) &:= N(\theta,\dot{\theta}) \\ \tilde{rd}() &:= rd() \\ \tilde{wr}(\theta_e,\dot{\theta}_e,\ddot{\theta}_e,\tau_e) &:= wr(\theta_e,\dot{\theta}_e,\ddot{\theta}_e,\tau_e). \end{aligned}$$

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 that 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 Fig. 3. On the other hand, a stepper motor actuated robot might use a wr function that ignores the torque argument and uses the position argument to move the actuator. Both robots would use a rd function that 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.

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

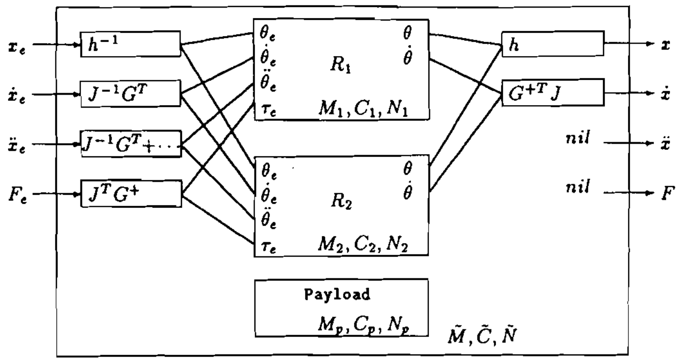

Fig. 4. Data flow in a two robot attach. In this example we illustrate the structure generated by a call to attach with 2 robots and a payload (e.g., a system like Fig. 2). The two large interior boxes represent the two robots, with their input and output functions and their inertia properties. The outer 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*.

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 that 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

$$J\dot{\theta}_R = G^T \dot{x} \tag{14}$$

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

$$\dot{\theta}_R = \mathcal{A}\dot{x} \tag{15}$$

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

$$\tilde{M} := M_p + \mathcal{A}^T M_R \mathcal{A} \tag{16}$$

$$\tilde{C} := C_p + \mathcal{A}^T C_R \mathcal{A} + \mathcal{A}^T M_R \dot{\mathcal{A}} \tag{17}$$

$$\tilde{N} := N_p + \mathcal{A}^T N_R \tag{18}$$

$$\begin{aligned}\tilde{rd}() := (&h(\theta_R), \mathcal{A}^+\dot{\theta}_R,\\ &\mathcal{A}^+\ddot{\theta}_R + \dot{\mathcal{A}}^+\dot{\theta}_R, \mathcal{A}^T\tau_R)\end{aligned} \tag{19}$$

$$\begin{aligned}\tilde{wr}(x_e, \dot{x}_e, \ddot{x}_e, F_e) := wr_R(&h^{-1}(x_e), \mathcal{A}\dot{x}_e,\\ &\mathcal{A}\ddot{x}_e + \dot{\mathcal{A}}\dot{x}_e, \mathcal{A}^{+T}F_e)\end{aligned} \tag{20}$$

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 Fig. 4.

The $\tilde{rd}$ attribute for an attached robot is a function that queries the state of all the robots in robot-list. Thus θ_R in (19) 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 that 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.

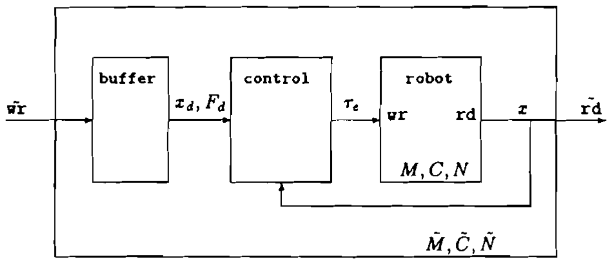

Fig. 5. Data flow in a typical controlled robot. Information written to the robot is stored in an internal buffer where it can be accessed by the controller. The controller uses this information and the current state of the robot to generate forces that cause it to follow the desired trajectory.

In principle, we could also define a DETACH primitive that would allow a composite robot system to be broken into components. This operation could be useful when a robot hand releases an object in its grasp, for example. An alternative approach is to respecify the new control structure that properly models the constraints after the object is released. Depending on the implementation, it might be possible to do this efficiently using previously defined daughter robots.

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 represent 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 that saved the desired position, velocity, acceleration and force in a local buffer accessible to our controller. This configuration is shown in Fig. 5.

The dynamic attributes $\tilde{M}$, $\tilde{C}$ and $\tilde{N}$ are determined by the controller. At one extreme, a controller that 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 that 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 (3) has been preserved.

Computed Torque Controller: As we mentioned in Section II, the computed torque controller is an exactly linearizing controller that 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 $\tilde{M}$, $\tilde{C}$, and$\tilde{N}$ to zero and uses the rd and wr attributes as described previously. 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 [21] 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:

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

where rd and wr are attributes of the robot that 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 payloads attached at higher levels. In essence, a robot that is being controlled in this manner can be viewed as an ideal force generator that 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 IV.

IV. Examples

To make the use of the primitives more concrete we present some examples of a planar hand grasping a box (Fig. 2) using various control structures. For all of the controllers, we will use the following functions

M_b	inertia matrix for the box 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 Fig. 2.

High-Level Computed Torque Control

In this example we combine all systems to obtain a description of the dynamic properties of the overall system in box coordinates. Once this is done we can move the box by directly specifying the desired trajectory for the box. This structure is illustrated in Fig. 6.

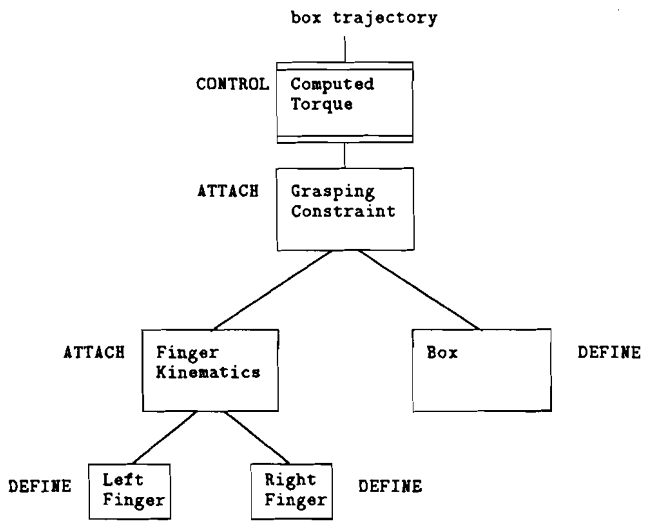

Fig. 6. High-level computed torque. The primitives listed next to the nodes in the graph indicate the primitive that was used to created the node. In this structure all dynamic compensation and error correction occurs at the top of the graph, using a complex dynamic model for the underlying system.

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

```
left     = DEFINE($M_l$,$C_l$, 0, rd_left, wr_left)
right    = DEFINE($M_r$,$C_r$, 0, rd_right, wr_right)
fingers  = ATTACH($J$, I,$f$, nil, left, right)

box   = DEFINE($M_b$,$C_b$, 0, nil, nil)
hand  = ATTACH(I,$G$,$g$, box, fingers)

ct_hand = CONTROL(hand, computed-torque).
```

It is useful to consider how the data flows to and from the control law running at the hand level. 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 $\dot{x}_b$
    fingers: ask for current state, $x_f$ and $\dot{x}_f$
        left: read current state, $\theta_l$ and $\dot{\theta}_l$
        right: read current state, $\theta_r$ and $\dot{\theta}_r$
    fingers: $x_f, \dot{x}_f \leftarrow f(\theta_l, \theta_r), J(\dot{\theta}_l, \dot{\theta}_r)$
hand: $x_b, \dot{x}_b \leftarrow g(x_f), G^T \dot{x}_f$
```

Similarly, when we write a set of hand forces using the wr attribute, it causes another chain of events to occur:

```
box: generate a box force $F_b$
    hand: generate finger force $G^+ F_b$
        fingers: generate joint torques $J^T G^+ F_b$
            left: output torques conjugate to $\theta_l$
            right: output torques conjugate to $\theta_r$.
```

Using the transformations given above it is straightforward to calculate the torque generated by the control law by expanding the structure of Fig. 6 using the definition of the primitives. Let the subscript d denote desired quantities at each level and let the subscripts b, h, f refer to the body, hand, and fingers attributes,

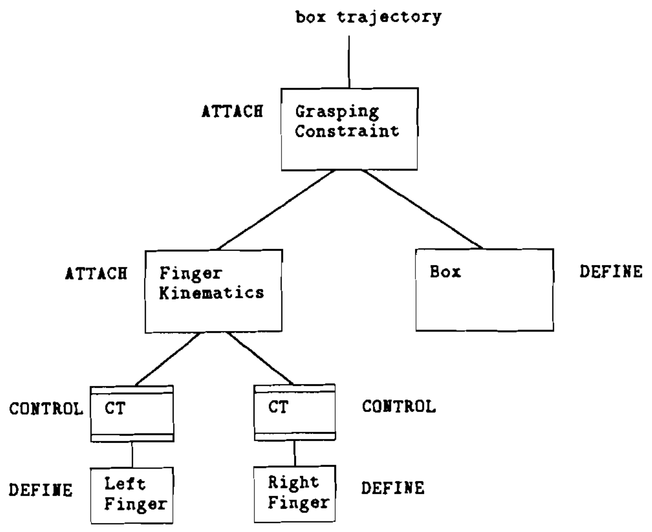

Fig. 7. Low-level computed torque. Computed torque controllers are used for the individual fingers to provide trajectory following capability in joint space. Since no controller is positioned above the box, the dynamics of the box are ignored even though the path is given in the box's frame of reference.

respectively.

$$\begin{aligned}\begin{pmatrix}\tau_l\\ \tau_r\end{pmatrix} &= J^T F_{f,d}\\ &= J^T G^+ F_{b,d}\\ &= J^T G^+[M_h(\ddot{x}_{b,d} + K_v\dot{e} + K_p e) + C_h\dot{x}_b]\\ &\;\vdots\\ &= J^T G^+\Big[\Big(M_b + GJ^{-T}M_\theta J^{-1}G^T\Big)(\ddot{x}_{b,d} + K_v\dot{e} + K_p e)\\ &\quad + \Big(C_b + GJ^{-T}C_\theta J^{-1}G^T\Big)\dot{x}_b\\ &\quad + GJ^{-T}M_\theta(d/dt)\Big(J^{-1}G^T\Big)\dot{x}_b\Big]\end{aligned}$$

M_θ and C_θ refer the block diagonal matrices formed by the individual finger attributes. This control law corresponds exactly to the generalized computed torque control algorithm presented by Li, *et al.* [19], with the omission of internal forces (see Section V-A).

A. Low-Level Computed Torque Control

Another common structure for controlling robots is to convert all trajectories to joint coordinates and perform computed torque at that level. In a crude implementation one might assume the dynamic effects of the box were negligible and construct the following structure shown in Fig. 7.

The primitives used to define this structure are

```
left  = DEFINE(M_l, C_l, 0, rd_left, wr_left)
right = DEFINE(M_r,C_r, 0, rd_right, wr_right)
ct_left = CONTROL(left, computed-torque)
ct_right = CONTROL(right, computed-torque)

fingers = ATTACH(J,I, f, nil, ct_left, ct_right)
box     = DEFINE(M_b,C_b, 0, nil, nil)

hand = ATTACH(I,G,g, box, fingers)
```

This controller is provably exponentially stable when the mass of the box is zero. However, this controller does not compensate for the mass of the box. As a result, we expect degraded performance if the mass of the box is large. Experimental results on a system of this form confirm our intuition [23].

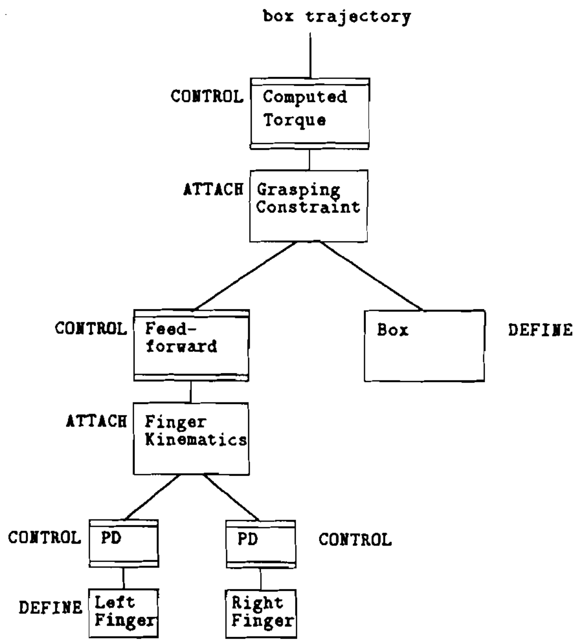

Fig. 8. Multilevel computed torque and stiffness (PD). Controllers are used at each level to provide a distributed control system with biological motivation (cf. Fig. 1), desirable control properties, and computational efficiency.

B. Multilevel Computed Torque/Stiffness Control

As a final example, we consider a control structure obtained by analogy with biological systems in which controllers to run at several different levels simultaneously (see Fig. 8).

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 [14]. 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. Here 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 the box's motion and correct for errors.

The structure in Fig. 8 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. Furthermore, it allows a degree of device independence since the underlying dynamics of the fingers are masked from higher level controllers. 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. Another feature is that if we ignore the additional torques due to the PD terms, the joint torques generated due to an error in the box position are the same as those of the high-level computed torque scheme presented earlier.

V. Extensions to the Basic Primitives

Having presented the primitives for nonredundant 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 II 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 as 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 (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 that 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 that 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 that lie in these subspaces.

A. Internal Forces

Consider first the case where J is square (and invertible) but G has a null space. If J is not square, then we can assume an extended Jacobian has been defined, as described in Section II-C. 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

$$wr(x_e, \dot{x}_e, \ddot{x}_e, F_e, F_i) \tag{21}$$

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.

Let $F_{i,1}$ parameterize the internal forces generated by the current constraint and let $F_{i,2}$ be the collection of internal forces present in the daughter robots.

The force transformations that describe this relationship are

$$\tau_R = \begin{pmatrix} \tau_{R,e} \\ \hline \tau_{R,i} \end{pmatrix} = \left(\begin{array}{ccc} J^T G^+ & J^T H^T & 0 \\ \hline 0 & 0 & I \end{array}\right) \begin{pmatrix} F_e \\ F_{i,1} \\ \hline F_{i,2} \end{pmatrix} \tag{22}$$

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 (5) in Section II-B. 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 (22) we see the appropriate definition for the new wr attribute generated by `attach` is

$$\tilde{wr}(x_e, \dot{x}_e, \ddot{x}_e, F_e, F_i) := wr_R(\cdots, J^T G^+ F_e + J^T H^T F_{i,1}, F_{i,2}) \tag{23}$$

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 (22):

$$F = \begin{pmatrix} F_e \\ F_{i,1} \\ \hline F_{i,2} \end{pmatrix} = \left(\begin{array}{cc} GJ^{-T} & 0 \\ HJ^{-T} & 0 \\ \hline 0 & I \end{array}\right) \begin{pmatrix} \tau_{R,e} \\ \hline \tau_{R,i} \end{pmatrix} \tag{24}$$

It follows that

$$\tilde{rd}() := \left(\cdots, GJ^{-T}\tau_{R,e}, \begin{pmatrix} HJ^{-T}\tau_{R,e} \\ \tau_{R,i} \end{pmatrix}\right). \tag{25}$$

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 that 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 most efficiently resolved at this level. Unlike external motions and forces, internal forces are not subject to coordinate change and so leaving such forces unresolved causes higher level controllers to use low-level coordinates.

B. Internal Motions

Internal motions are also created by the `attach` primitive, this time due to a nonsquare 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 andwr primitives, but just specify the internal and external motion components.

Given a constraint that 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

$$\left(\begin{array}{cc} J & 0 \\ K & 0 \\ \hline 0 & I \end{array}\right) \begin{pmatrix} \dot{\theta}_{R,e} \\ \hline \dot{\theta}_{R,i} \end{pmatrix} = \left(\begin{array}{ccc} G^T & 0 & 0 \\ 0 & I & 0 \\ \hline 0 & 0 & I \end{array}\right) \begin{pmatrix} \dot{x}_e \\ \dot{x}_{i,1} \\ \hline \dot{x}_{i,2} \end{pmatrix}. \tag{26}$$

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

$$\bar{J} = \begin{pmatrix} J \\ K \end{pmatrix} \qquad \bar{G}^T = \begin{pmatrix} G^T & 0 \\ 0 & I \end{pmatrix} \qquad (27$$

we see that $\bar{J}$ is full rank and so we can use it to define $\mathcal{A} = \bar{J}^{-1}\bar{G}$ in (16)–(20). This then defines the dynamics attributes created attach. Note that the dimension of the constrained subspace (whe internal forces act) is unchanged by this extension.

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

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

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

$\ddot{\theta}_{R,e}$ 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

$$x_e = h(\theta_{R,e}) \tag{30}$$

$$\dot{x}_e = G^+ J \dot{\theta}_{R,e} \tag{31}$$

$$\dot{x}_i = \begin{pmatrix} K\dot{\theta}_{R,e} \\ \dot{\theta}_{R,i} \end{pmatrix} \tag{32}$$

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

Controllers must also be extended to understand internal 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

$$F = M(q)\begin{pmatrix} \ddot{x}_{e,d} + K_v \dot{e}_e + K_p e_e \\ \ddot{x}_{i,d} + K_v \dot{e}_i \end{pmatrix} + C(q,\dot{q})\begin{pmatrix} \dot{x}_e \\ \dot{x}_i \end{pmatrix} + N(q,\dot{q}) \tag{33}$$

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.

C. Other Issues

Control laws commonly use the position of the object as part of the feedback term. This may not always be available for systems with nonintegrable constraints (such as grasping with rolling contacts). If the object position cannot be calculated from θ alone, then we must retrieve it from some other source. One possibility is to use an external sensor that 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 that do not have well defined inverse kinematic functions. There are some robot control laws that 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.

VI. Conclusion

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.

There are many problems, both general and specific, which remain open. We discuss a few of these below.

One of the primary challenges in robotic control is the integration of large numbers of manipulators performing a coordinated task. The primitives given in this paper allow specification of controllers that can be used for such tasks. A typical task consists of many different phases, each potentially requiring a different type of control. An example of this is a hand grabbing an object, in which the initial move and grab phases require position control of the fingers, while the manipulation phase requires control of the composite grasping system.

For each phase of the task, the control primitives can be used to specify hierarchical control structures appropriate to the motion being performed. Methods for smoothly changing from one control structure to another, for example when contact is being made with an object, need to be developed. If we represent the system dynamics as a finite state machine, with each state corresponding to a different set of constraints on the overall system, we can associate with each state a controller defined in terms of the control primitives. In this context, we are interested in control structures that facilitate the transition from one state to another. It particular, it seems important to insure that the controller does not force the system to return to the previous state and begin an oscillation between states and the associated controllers. The idea of representing the system as a graph of dynamical systems is from a suggestion by Brockett [4] and is worthy of more study.

It would be very useful to determine broad stability properties for the system with controllers at various levels. This is particularly difficult when there are constraints in the system, since some low-level controllers may not be aware of the constraints and the resulting model mismatch could cause instability. A weak version of the desired result is the observation that if we place a computed torque controller at the highest level and feedforward controllers at lower levels of the hierarchy, the commanded torques are identical to those obtained by a single computed torque controller at the top of the hierarchy. Thus the overall system is stable. Similar results with PD controllers distributed throughout the structure seem plausible, and there has been some work in this area in other contexts, such as decentralized control [29]. By using the structure inherent in controllers constructed using the primitives, it may be possible to adapt these results and strengthen them.

Finally, 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.

Related to the issue of on-line versus off-line computation, is the level at which we decide to label a mechanism as a simple robot object, created by the `define` primitive. It is certainly possible to define the individual links of a robot manipulator as robot objects, and then attach these robots into a larger structure, such as a robotic finger. However, since the links of the robot remain attached throughout the execution of a task, it is likely that a more efficient representation of the dynamics can be achieved by modeling the entire finger as a single mechanism. This corresponds to computation of the finger dynamics in an off-line fashion.

We have made an initial implementation of the primitives using the Mathematica symbolic manipulation program [32]. This implementation supports the basic primitives described in Section III and corresponds to a completely off-line implementation. The Mathematica source code is available via anonymous ftp from united.berkeley.edu in the directory "pub/RobotPrimitives." The listings for the Mathematica package can be found in a recent technical report [9].

Acknowledgment

The authors would like to thank the reviewers for their remarks, which have helped them provide a clearer description of the motivation and goals of this work.

References

[1] J. Baillieul, "Kinematic programming alternatives for redundant manipulators," in *Int. Conf. on Robotics and Automation,* 1985, pp. 722–728.
[2] A. K. Bejczy, "Robot arm dynamics and control," Tech. Rep. 33-699, Jet Propulsion Lab., 1974.
[3] R. W. Brockett, "On the computer control of movement," Tech. Rep. CICS-P-31, Center for Intelligent Control Systems, Harvard Univ., Nov. 1987.
[4] R. W. Brockett, personal communication, 1990.
[5] D. Clark, "HIC: An operating system for hierarchies of servo loops," in *IEEE Int. Conf, Robotics Automat.*, 1989, pp. 1004–1008.
[6] A. B. A. Cole, J. E. Hauser, and S. S. Sastry, "Kinematics and control of multifingered hands with rolling contact," *IEEE Trans. Circuits and Systems*, vol. 34, pp. 398–404, 1989.
[7] J. J. Craig, *Introduction to Robotics: Mechanics and Control*, second ed. Reading, MA: Addison-Wesley, 1989.
[8] M. R. Cutkosky, R. D. Howe, and A. P. Witkin, "Object-oriented modeling of robot hands," in *Knowledge-Based Expert Systems for Manufacturing*, 1986, presented at the Winter Annual Meeting of the ASME, pp. 177–186.
[9] D. C. Deno, R. M. Murray, K. S. J. Pister, and S. S. Sastry, "Control primitives for robot systems," Tech. Rep. UCB/ERL M90/39, Electronics Res. Lab., Univ. California at Berkeley, May 1990.
[10] V. Eng, *The MDL Programmer's Reference Manual.* Harvard Robotics Lab., 1988, first revision.
[11] K. S. Fu, R. C. Gonzalez, and C. S. George Lee, *Robotics: Control, Sensing, Vision, and Intelligence.* New York: McGraw-Hill, 1987.
[12] C. Ghez, "Introduction to the motor system," in *Principles of Neural Science*, second ed., E. R. Kandel and J. H. Schwartz, Eds. New York: Elsevier, 1985, ch. 33.
[13] G. Hinton, "Some computational solutions to Bernstein's problems," in *Human Motor Actions — Bernstein Reassessed*, H. T. A. Whiting, Ed. New York: Elsevier, 1984, ch. 4b.
[14] N. Hogan, "Stable execution of contact tasks using impedance control," in *IEEE Int. Conf, Robotics Automat.*,1987, pp. 1047–1053.
[15] P. Hsu, J. Hauser, and S. Sastry, "Dynamic control of redundant manipulators,". *J. Robotics Syst.*, vol. 6, no. 2, pp. 133–148, 1989.
[16] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE J. Robotics and Automation*, vol. RA-3, pp. 43–53, Feb. 1987.
[17] O. Khatib, "Augmented object and reduced effective inertia in robot systems," in *American Control Conf.*, 1988, pp. 2140–2147.
[18] D. Koditschek, "Natural motion for robot arms," in *IEEE Control and Decision Conf.*, 1984, pp. 733–735.
[19] Z. Li, P. Hsu, and S. Sastry, "Grasping and coordinated manipulation by a multifingered robot hand," *Int. J. Robotics and Contr.*, vol. 8, no. 4, pp. 33–50, 1989.
[20] Z. Li and S. Sastry, "A unified approach for the control of multifingered robot hands," *Contemporary Mathematics*, vol. 97, pp. 217–239, 1989.
[21] J. Y. S. Luh, M. W. Walker, and R. P. Paul, "Resolved acceleration control of mechanical manipulators,"*IEEE Trans. Circuits Syst.*, vol. CAS-25, pp. 468–474, 1980.
[22] R. M. Murray, *Robotic Control and Nonholonomic Motion Planning.* Ph.D. dissertation, Univ. California at Berkeley, 1990.
[23] R. M. Murray and S. S. Sastry, "Control experiments in planar manipulation and grasping," in *IEEE Int. Conf. Robotics Automat.*, pp. 624–629, 1989.
[24] ——, "Grasping and manipulation using multifingered robot hands," in *Robotics: Proceedings of Symposia in Applied Mathematics, Vol. 41*, R. W. Brockett, Ed. Providence, RI: Amer. Math. Soc., 1990, pp. 91–128.
[25] R. P. Paul, *Robot Manipulators: Mathematics, Programming, and Control.* Cambridge, MA: MIT Press, 1981.
[26] R. P. Paul and V. Hayward, "Robot control and computer languages," in *Theory and Practice of Robots and Manipulators*, pp. 187–193, 1984,; also, in *Proc. RoManSy '84: The Fifth CISM-IFTOMM Symp.*.
[27] R. M. Rosenberg, *Analytical Dynamics of Discrete Systems.* New York: Plenum, 1977.
[28] N. Sadegh, "Adaptive control of mechanical manipulators: stability and robustness analysis," Ph.D. dissertation, Dept. Mech. Eng., Univ. California, Berkeley, 1987.
[29] N. R. Sandell, Jr., P. Varaiya, M. Athans, and M. G. Safonov, "Survey of decentralized control methods for large scale systems," *IEEE Trans. Automat. Contr.*, vol. AC-23, pp. 108–128, 1978.
[30] J. E. Slotine and W. Li, "On the adaptive control of robot manipulators," in *Int. J. Robotics and Contr.*, vol. 6, 1987, pp. 49–59.
[31] M. W. Spong and M. Vidyasagar, *Dynamics and Control of Robot Manipulators.* New York: Wiley, 1989.
[32] S. Wolfram, *Mathematica: A System for Doing Mathematics by Computer.* Reading, MA: Addison-Wesley, 1989.

Optimal Grasps for Planar Multifingered Robotic Hands

Gongliang Guo, William A. Gruver and Qixian Zhang

Abstract—**A fast and simple algorithm for determining optimal grasps of a planar multifingered robotic hand is presented. Using a representation for the stiffness of a planar grasp system, we derive conditions for static equilibrium, task-oriented grasping, and stability. An optimization model for determining optimal grasps of a planar multifingered hand is established using a quality measure, which includes the effect of the external disturbing forces and moments. A constrained gradient descent technique is used to compute the optimal grasps. An example demonstrates the applicability and effectiveness of the theory.**

I. Introduction

There is increasing interest in multifingered robotic hands that have the capability to grasp objects of different shapes and to manipulate them with a variety of actions to accomplish a task [1]. This interest has been driven by a need to extend the flexibility, dexterity, and precision of industrial robots, and a desire to improve prostheses for

Manuscript received March 6, 1990; revised December 22, 1990 and May 22, 1991.

G. Guo and W. A. Gruver are with the Center for Robotics and Manufacturing Systems University of Kentucky, Lexington, KY 40506-0108

Q. Zhang is with the Institute for Robotics, Beijing University of Aeronautics and Astronautics, Beijing, China

IEEE Log Number 9104108..