U.S. DEP ERCE
National and Technology
(formerly National Bureau of Standards)

NIST Technical Note 1255

Manipulator Servo Level Task Decomposition

John C. Fiala

NIST Technical Note 1255

Manipulator Servo Level Task Decomposition

John C. Fiala

Intelligent Controls Group
Robot Systems Division
Center for Manufacturing Engineering
National Institute of Standards and Technology
(formerly National Bureau of Standards)
Gaithersburg, MD 20899

Issued October 1988

NOTE: As of 23 August 1988, the National Bureau of Standards (NBS) became the National Institute of Standards and Technology (NIST) when President Reagan signed into law the Omnibus Trade and Competitiveness Act.

U.S. Department of Commerce
C. William Verity, Secretary

National Institute of Standards and Technology
(formerly National Bureau of Standards)
Ernest Ambler, Director

National Institute of Standards and Technology
Technical Note 1255
41 pages (Oct. 1988)

U.S. Government Printing Office
Washington: 1988

For sale by the Superintendent of Documents
U.S. Government Printing Office
Washington, DC 20402

Contents

Manipulator Servo Level Task Decomposition

1. Introduction

This document describes a servo level control structure for robotic manipulators. The discussion is limited to electric-powered manipulators with serial joints and unbranched kinematics. The terminology of the document originates in [3] and in the general references [17,47,54]. As much as possible, terminology will be defined as it is introduced. However, it is expected that the reader has some familiarity with control theory, and, in particular, with manipulator control.

The architecture for a hierarchical control system is outlined in [3]. This architecture contains a *task decomposition* hierarchy which decomposes complex tasks into simpler and simpler objectives, down to the Servo Level which computes the motor control signals for the actuators. The Servo Level is the lowest level of the task decomposition hierarchy in figure 1. Servo receives commands from the next higher level, the Primitive Level.

The function of Servo task decomposition is to handle motion "small in a dynamic sense." This means that the module executes a specified algorithm for approaching the command attractor set, the *attractor set* being the set of positions, velocities, accelerations, acceleration rates, forces, and force rates desired (by Primitive) as the state of the manipulator. Generally, the specified algorithm only gives valid behavior for the manipulator when the attractor is within a small neighborhood of the current manipulator state. Thus the input from Primitive is typically a point in much more complex trajectory planned by Primitive. Primitive handles the problem of generating trajectories with "smooth" accelerations. Primitive moves the manipulator along a trajectory by periodically updating the attractor set commanded to Servo. Primitive can also change the algorithm executing at the Servo Level with a new command.

Paralleling the task decomposition hierarchy are two more hierarchies, the *sensory processing* hierarchy and the *world modeling* hierarchy [3,22]. The sensory processing hierarchy obtains and processes data from system sensors. World modeling analyzes the sensory processing data and maintains an internal model of the world. World modeling is the part of the system which coordinates the sensory processing and task decomposition hierarchies, and maintains the *global data system.* As such, world modeling can be considered to have support modules that act as servers to sensory processing and task decomposition modules as depicted in figure 2. In particular, there is some component of world modeling at the Servo Level which acts as a server to task decomposition, providing model-based information relevant to servo control. This module is called Servo Support in figure 2. The Servo Support module and the Data Acquisition Support module, together with the connection between the modules through the global data system, form the world modeling concept as depicted by a single box in figure 1. The additional structure depicted in figure 2 for world modeling, allows task decomposition to be treated separately from sensory processing. This document discusses aspects of the system on the task decomposition side only. A detailed discussion of world modeling and sensory processing will given in subsequent documents.

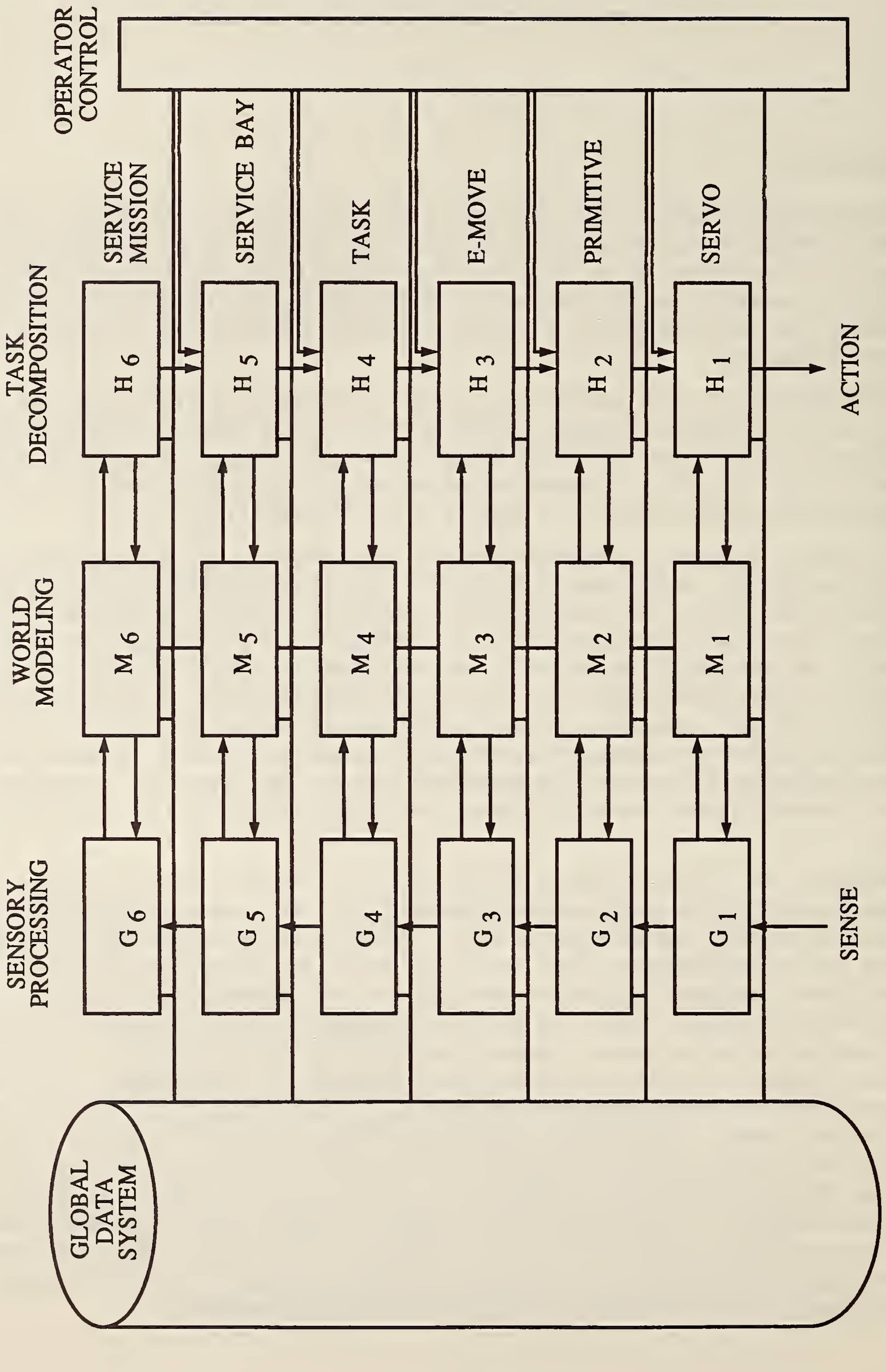

Figure 1. A hierarchical telerobot control architecture.

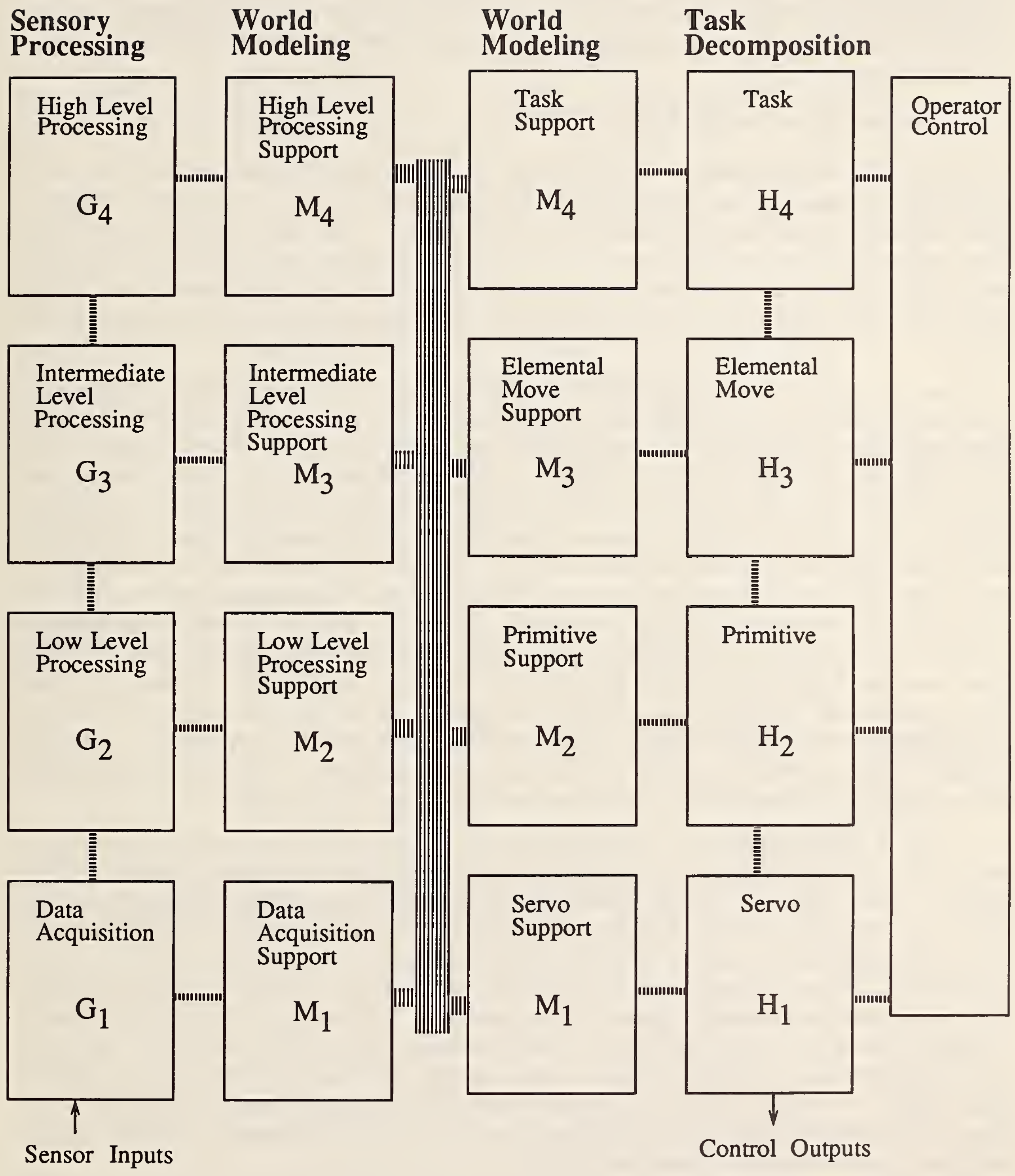
Sensory Processing
World Modeling
World Modeling
Task Decomposition
High Level Processing G_4
High Level Processing Support M_4
Task Support M_4
Task H_4
Operator Control
Intermediate Level Processing G_3
Intermediate Level Processing Support M_3
Elemental Move Support M_3
Elemental Move H_3
Low Level Processing G_2
Low Level Processing Support M_2
Primitive Support M_2
Primitive H_2
Data Acquisition G_1
Data Acquisition Support M_1
Servo Support M_1
Servo H_1
Sensor Inputs
Control Outputs

Figure 2. Principal components of architecture.

Also depicted in figures 1 and 2 is operator control. This part of the architecture provides the operator with a means to interact with the autonomous system at various levels of the hierarchy. The operator can issue commands at any level, thus performing teleoperation or supervisory control [3]. In special cases, a *shared control* mode can also be defined, as will be described in section 4.

2. The Detailed Servo Architecture

Figure 2 shows the principal components of a telerobot control system architecture. Also shown in the figure are the principal data pathways between these components. Figure 3 gives additional detail for the overall system architecture. This figure shows that the data pathways are actually links to data packets. Although in principle any module can link to any data packet [3], only the principal communication links of the system are shown here. Additional links come with some expense in terms of understandability and maintainability of the system.

There are many ways to implement the data pathways for communicating between system components; and although the different approaches will not be described here, there are some important features that any such approach must provide for the control system. First of all, the modules (depicted in the figure by rectangular boxes) are *cyclicly executing* processes. This means that each such module continually reads inputs, performs computation, and then writes output in a cyclic manner. (Note that the boxes shown in figure 3 will be further decomposed into separate processes but for illustrative purposes will be considered as single processes now.) For most processes of the system, the read-compute-write cycle is continually performed for the life of the system. Even when no new commands are issued to a module, cyclic execution is still being performed. This allows the robot to react to sensed changes in the environment, e.g., disturbances to the servo control loop. In order to insure that a module will be able to react quickly, the module should not get hung up waiting for communication with another module. Thus, reliable cyclic execution requires that a module be able to read or write data with as little waiting as possible. A process should always read and execute on the most current data, and not wait for new data to arrive.

A second feature required of the data pathways is that they allow access by many processes. This is particularly true for the main part of the global data system, the big oval in figure 3. In this area is stored the robot's best estimate of the state of the world [3], including the locations of objects in the environment and the location of the robot itself. Many different components of the system require access to this kind of information. However, if the components all keep such information locally, it is very difficult to maintain as things change. It is much better to centrally organize this information. This multiple access feature can be realized by using a common data packet that is protected against reads and writes while data is being written but allows multiple reads. Several processes can then read the data at once, with a wait required only for a period while data is being written. It should be mentioned that the central part of the global data may need a lot of organization. The big oval in the figure must have additional structure inside it.

Note that in figure 3 the operator control block has been broken into separate modules, one for each level of the task decomposition hierarchy. This is to show that the operator interacts with each level in a specific way. Operator control at the Servo Level is achieved by

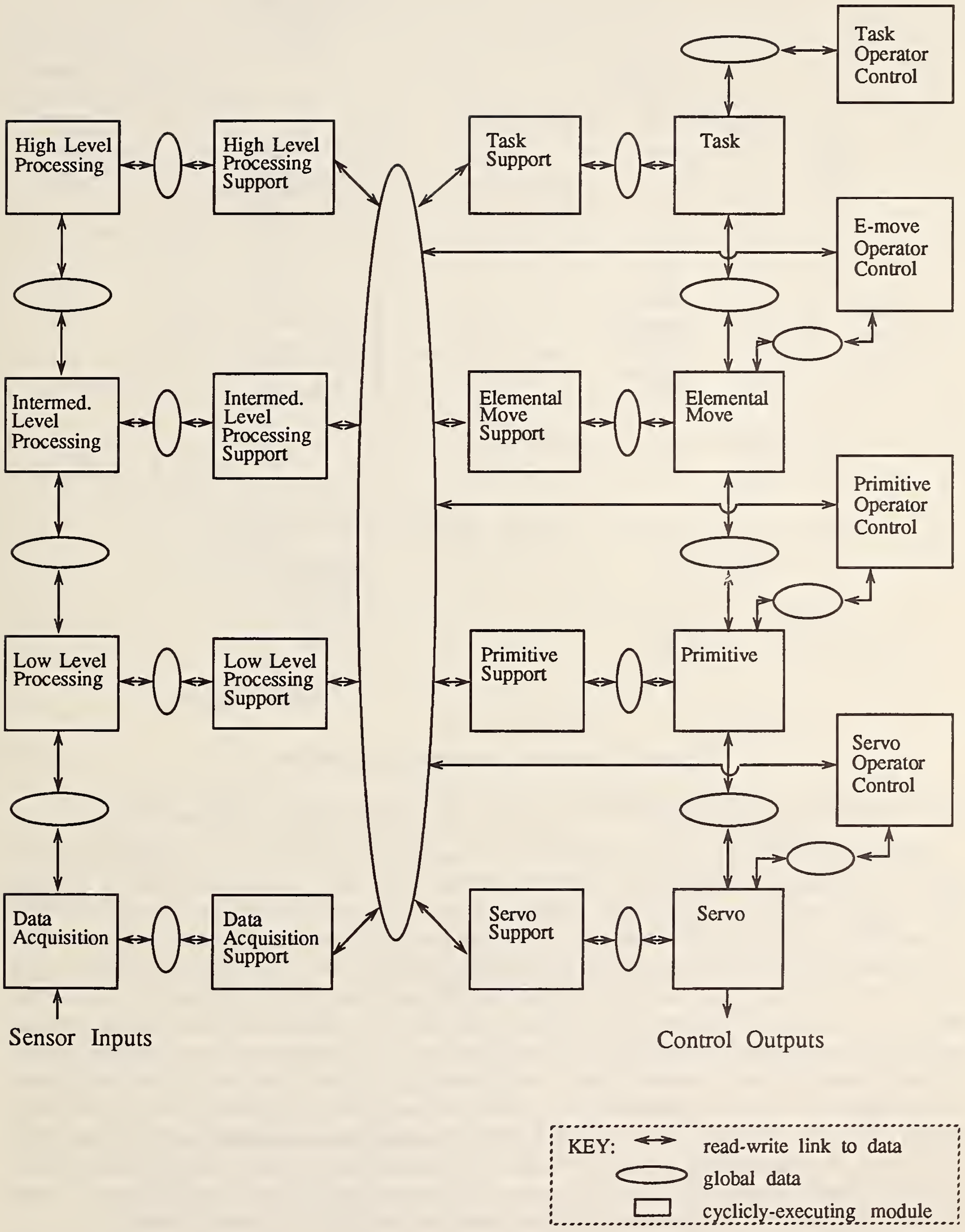

Figure 3. Principal system interconnections.

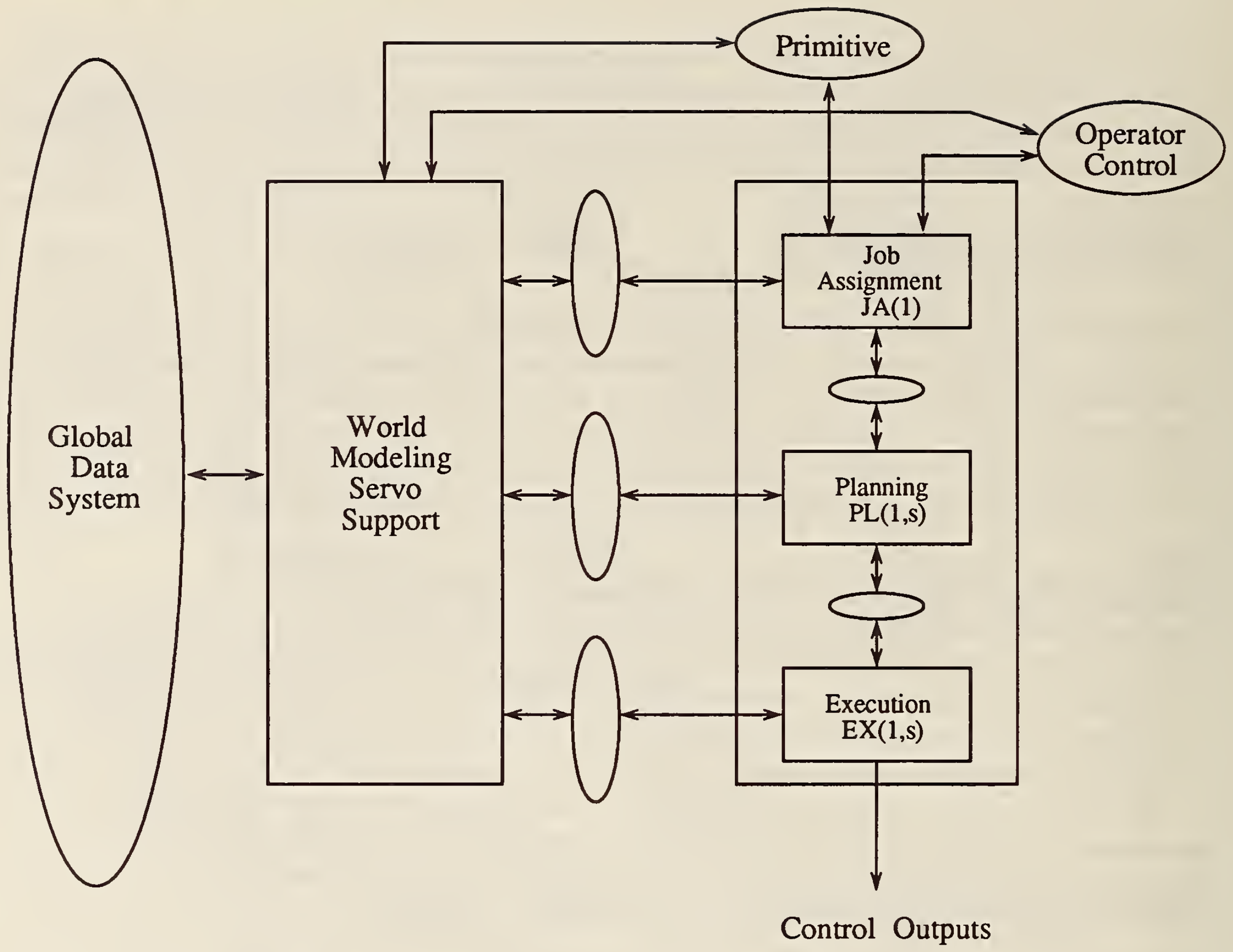

Figure 4. Servo task decomposition structure.

the Servo Operator Control module. This module is described in more detail elsewhere.

One important feature of the architecture not made clear in figures 1 - 3 is that the architecture hierarchically decomposes around equipment [3 p. 15]. This means that, in general, there is a Servo task decomposition module for each separate piece of equipment that must be controlled. A Primitive module will coordinate a set of Servo modules and an E-move level will coordinate a set of Primitives. Generally, there is one Task module for the entire robot. As an example of how the architecture decomposes around equipment consider an arm of the telerobot. There will be one Servo module for controlling the manipulator arm and another for controlling the end-effector (hand) of that arm. These two Servo modules would be coordinated by one Primitive level. An advantage of this approach is that the system becomes very modular. In addition, the system can easily make a process inactive and bring another online in its place. This is useful when changing the equipment configuration, such as changing the end-effector of the robot arm. The servo process used to control the old end-effector is made inactive and the servo process for controlling the new end-effector is made active. (It is likely that different end-effectors would require different types of control.)

The detailed architecture of Servo task decomposition for a manipulator arm is shown in figure 4. Task decomposition is broken into three major software modules, Job Assignment,

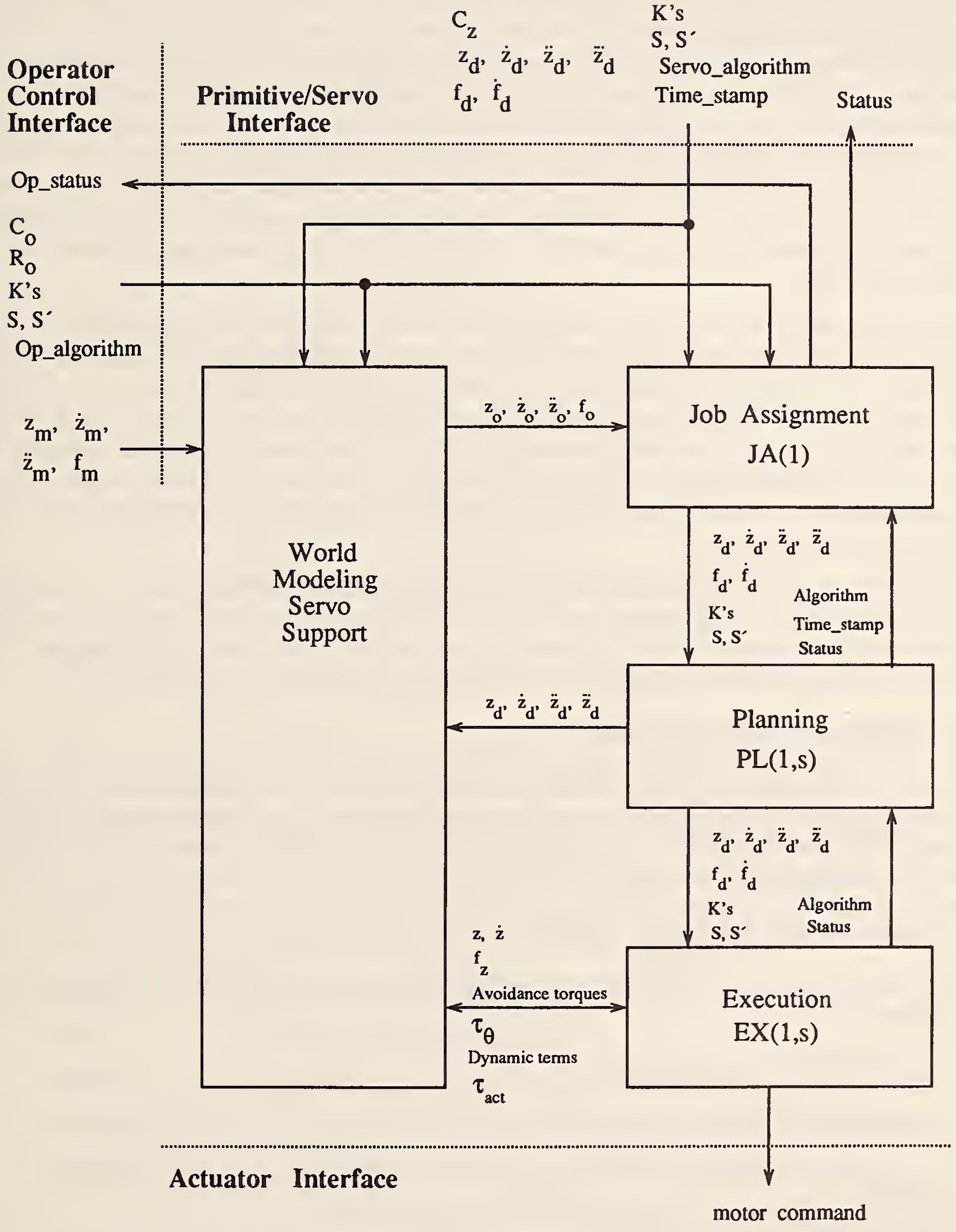

Figure 5. Servo Level task decomposition interfaces.

Planning, and Execution. The Job Assignment module defined here corresponds to the process JA(1) of [3]. The module determines whether the current job is to be assigned from the operator interface or the Primitive/Servo interface, or a combination of the two. The Planning module functionality encompasses that of the processes PL(1,s), s=1,...,n, of [3], n being the number of actuators in the manipulator. Likewise, the Execution module encompasses the EX(1,s), s=1...n, processes of [3]. The figure depicts the interfaces between modules as well as the interfaces to subsystems external to the Servo Level of the manipulator. Note that the interfaces between Job Assignment and Planning, and between Planning and Execution, have the same structure as previously described for the major system components. This is because the three modules are also cyclicly-executing processes. However, it is much less likely that these interfaces will be accessed by processes external to Servo task decomposition.

The Execution module directly computes the motor command for the manipulator. This usually means that the module computes a servo error in the coordinate system commanded by Primitive. To compute the appropriate control, the Execution module receives periodic attractor points for the manipulator from the Planning module and model-based terms (such as Jacobians, inertias, etc.) from world modeling. The Planning module feeds the periodic data points to the Execution module at the appropriate rate. The Job Assignment module receives the commands from Primitive and operator control, and provides a coordinated output command to the Planning module.

The interfaces to these modules are shown in much greater detail in figure 5. This figure shows the information that passes through the interfaces. The data ovals are not shown and the operator control interface is shown on the left in this figure for convenience. The information of the interfaces is described in the subsequent sections of this document.

3. Job Assignment Module Interfaces

The Job Assignment module interfaces to the upper task decomposition hierarchy, operator control, and to the Servo Level world modeling hierarchy. This section describes the data that compose those interfaces. The module also interfaces to the Planning module. This interface will be described in section 4.1.

3.1. Primitive to Job Assignment Interface

The components of the interface between the Job Assignment module and Primitive are listed below:

C_z = (coord. sys., T_w, T_e)

z_d, $\dot{z}_d$, $\ddot{z}_d$, $\dddot{z}_d$

f_d, $\dot{f}_d$

K_p, K_v, K_i, K_{pf}, K_{vf}, K_{if}, K_τ

S, S′

Servo_algorithm

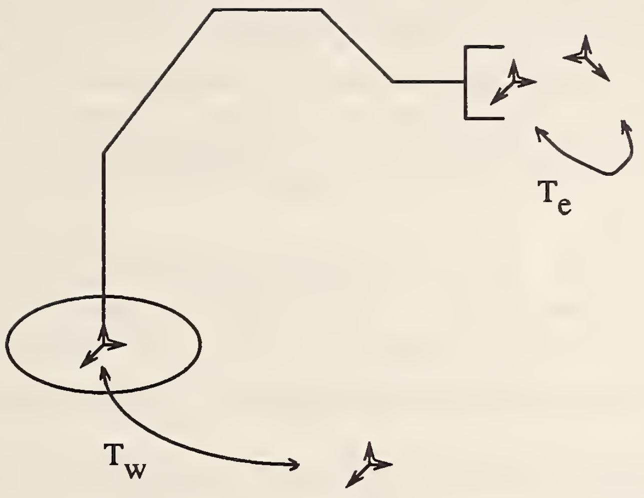

Figure 6. Cartesian coordinate system relationships.

Time_stamp = (synch_flag, t_p)

Status

Each element of this list will be described in detail in the following. The last element of the list is sent from Servo to Primitive. The rest are output from Primitive to Servo.

The parameter C_z is a coordinate system specifier. C_z indicates the coordinate system in which the position and force command vectors are expressed. If the command to Servo is taken from this interface, this coordinate system will be the coordinate system in which the servo errors are computed. The choices for C_z are

(motor)	=> commands in motor space
(joint)	=> commands in joint space
(world)	=> commands in Cartesian system fixed at manipulator base
(end-effector)	=> commands in Cartesian system fixed at the end-effector
(end-effector, T_e)	=> commands in Cartesian system with relation T_e to the system fixed at the end-effector
(world, T_w, T_e)	=> commands T_e with respect to end-effector in Cartesian system fixed at T_w from the manipulator base

The relationships T_w and T_e can be represented by rigid-body, homogeneous coordinate transformations. Figure 6 illustrates the relationships embodied in C_z. The use of T_w and T_e as shown in the figure allows Servo to execute control in a coordinate system relevant to the current task. As described in [42], certain types of control are done with respect to task coordinates, fixed either at the tip of the object held by the manipulator or in some object attached to the environment. The transformation parameters allow the Servo coordinates to be fixed in the world at T_w from the manipulator base, or fixed at T_e from the manipulator's end-effector. The T's can even change with each Primitive input, yielding a time-varying effect.

The vectors z_d, $\dot{z}_d$, $\ddot{z}_d$, $\dddot{z}_d$, specify the desired position, velocity, acceleration and jerk for the manipulator each control cycle. These values are expressed in the coordinate system chosen by C_z. Note that if the end-effector coordinates are chosen, pure positional information z_d is not valid. However, the other three elements can still be specified.

The vector f_d specifies the desired forces for the manipulator. The vector $\dot{f}_d$ specifies the desired force rates. These two vectors are specified with respect to the coordinate system of C_z. For example, if C_z indicates joint coordinates then the vectors are vectors of joint torques and torque rates. If C_z indicates world coordinates then the vectors are of forces and torques at the end-effector with respect to the world coordinate frame. Note that the desired force information and desired position information are always in the same coordinate system. The set of six vectors (z_d, $\dot{z}_d$, $\ddot{z}_d$, $\dddot{z}_d$, f_d, $\dot{f}_d$) forms the attractor set, (desired state,) of the manipulator for each control cycle.

The gain terms K_p,..., K_τ are the gain coefficients which multiply the error vectors in the control equations. The K's determine the impedance of the manipulator [28,33,51,59]. These may be full matrices, but in most cases take the diagonal form. See section 8 for more details.

The parameters S and S´ are selection matrices used to select the coordinate axes to be force controlled and position controlled. Thus, these parameters are used with hybrid position/force schemes. In the Raibert and Craig scheme [49], S is diagonal 1's and 0's. However, the parameters can also be used for task specification matrices in the manner of Khatib [35].

The next element of the Servo command input is the Servo_algorithm selector. This parameter indicates what algorithm is to be used for approaching the attractor set during subsequent control cycles. The possible types of algorithms are described in section 8.

The parameter Time_stamp in the command to Servo is used to synchronize Servo execution of the command with the Primitive Level execution module's time variable t_p. When the synch_flag is set, the Servo will resynchronize its internal clock with t_p. This lets the Primitive execution module reset its internal representation as necessary. The use of this parameter will be explained in greater detail in section 6.

The Status parameter, unlike the other elements of the interface, is passed from the Servo Job Assignment module to the Primitive Level execution module. The value of the parameter indicates the status of Servo operations. At least four important values can be identified.

"Executing"	=> Servo is processing the most recent command
"Ready_nxt_pt"	=> Servo Job Assignment is ready for additional data
"Cmd_error"	=> Servo could not process the last command
"Servo_error"	=> Servo encountered a fatal error during execution

The Primitive Level execution module regulates its behavior based on the current "Status" and the values of system sensors.

3.2. Job Assignment to Operator Control Interface

The control architecture of which Servo is a part must incorporate both teleoperated and autonomous operation. Thus, at each level there is an interface to operator control. It is the task of the Job Assignment module to determine whether Servo will execute from the autonomous commands (Primitive), or from operator commands given through some input device such as a joystick or master arm. In addition, the Job Assignment module must be able to coordinate operator and Primitive commands for shared control operation. The elements of the Job Assignment to operator control interface provide a means for teleoperated and shared control, as described below.

C_o = (coord. sys., T_w, T_e)

R_o

$K_p, K_v, K_i, K_{pf}, K_{vf}, K_{if}, K_\tau$

S, S′

Op_algorithm

Op_status

The coordinate system specification from the operator, C_o, indicates the coordinates in which the operator commands are to be interpreted. The possible values of C_o take the same form as those of C_z described for the Primitive to Job Assignment interface. The T_w and T_e have the same form as described for C_z. Note that the values of these parameters may be different from those appearing in C_z. In general, the values in the operator control interface are independent of the values in the Primitive/Servo interface, the same symbols are used here to stress that these variables convey the same type of information.

The actual command positions, velocities, accelerations, and forces, from the operator are obtained from the global data system via the servo world modeling module. The possible devices from which operator input could be taken include joysticks, control pendants of various forms, and master arms. Master arm input can give the system the functionality of classical master-slave teleoperation.

The redundancy resolution parameter R_o specifies how redundancy is to be resolved for the operator command. This is probably only required when doing shared control, i.e. the operator coordinates C_o are underspecified with respect to the autonomous coordinates C_z. Normally, the coordinates of the command input to Servo are the coordinates of the servo control algorithm. In this case, any redundancy resolution needed to transform the command into joint space is incorporated in the dynamics of the control algorithm. Servo does not make a transformation of coordinates from command to servo coordinates, because the command is in the desired servo coordinates. (This is an important feature of the operation of Servo task decomposition!) However, during shared control, a transformation may be required to bring the Primitive input and the operator input into the same coordinate system. This may require redundancy resolution independent of the servo algorithm.

The servo gains (K's) and selection matrices (S,S´) are input from the operator interface when the command to Servo is taken from the operator. These parameters are of exactly the same form as the corresponding parameters in the Primitive to Servo interface. During shared control, the Job Assignment module must choose one set of these parameters based on the Op_algorithm.

The overall control of Servo is achieved by the parameter Op_algorithm. When Op_algorithm specifies autonomous operation, then the Job Assignment module takes commands from Primitive, using Servo_algorithm to select the control algorithm. When Op_algorithm specifies a teleoperation algorithm, then the module carries out the teleoperation control algorithm, taking input from an operator device. The Op_algorithm can also specify a form of shared control to be executed by Servo.

The parameter Op_status is analogous to the Status returned to Primitive by the Job Assignment module. The Op_status informs the operator control of the current status of Servo.

3.3. Job Assignment to World Modeling Interface

Note that certain elements of the Primitive and operator control interfaces are transmitted directly to the world modeling module of the Servo Level. These items could be considered as coming from the Job Assignment module, but, in any case, they should be discussed as they are vital to Servo operation. The elements needed by world modeling are listed here.

C_z = (coord. sys., T_w, T_e)

Servo_algorithm

C_o = (coord. sys., T_w, T_e)

R_o

Op_algorithm

World modeling needs to know these for several reasons. World modeling must know the selected control coordinates C_z (C_o) since the feedback variables will have to be put into these coordinates. Also, if dynamic terms are required for the particular algorithm, they will have to be in these coordinates. Of course, the relevance of any of the world modeling computations depends on the control algorithm determined by both Op_algorithm and Servo_algorithm. As stated previously, for shared control world modeling may need R_o to resolve transformations between C_z and C_o.

The Job Assignment module receives from world modeling the command data from the operator's input device.

$$z_o,\ \dot{z}_o,\ \ddot{z}_o, f_o$$

These vectors give the desired position, velocity, acceleration, and force for the manipulator when Servo is in teleoperation mode. Note that these vectors are with respect to the coordinates selected by C_o for pure teleoperation, and with respect to the coordinates selected by C_z for shared control. The values giving the state of the teleoperation device in C_o,

$$z_m,\ \dot{z}_m,\ \ddot{z}_m,\ f_m,$$

may be transformed by world modeling to get the final values sent to Job Assignment. Thus, the values z_o, $\dot{z}_o$, $\ddot{z}_o$, and f_o can represent the state of a master arm, in the teleoperation case. The state of the master serves as an attractor set for the slave arm controlled by Servo. In shared control mode, the vectors can represent incremental modifications to be made to the autonomous command vectors coming from Primitive. This wil! be discussed in greater detail below.

4. Job Assignment Operation

The purpose of the Job Assignment module is to receive the command inputs from the operator control and Primitive, and provide a coordinated output command to the Servo Planning module. Thus, the output from Job Assignment could reflect either a purely autonomous command (received from Primitive) or a purely teleoperated command (received from the operator input) or a combination of both operator and Primitive inputs (shared control). The Planning module is not told the original source of the command data; it acts the same for all three types of control.

The Job Assignment module, upon receiving a new command, first examines the Op_algorithm. This parameter indicates the desired control mode selected by the operator. The control mode could be autonomous, teleoperated, or shared control. For an autonomous control mode, the Job Assignment module takes the input strictly from the Primitive/Servo interface. No control data is obtained from the operator. Thus, as stated previously, the control algorithm is given by Servo_algorithm. Also, in autonomous mode, no data is obtained from world modeling to modify the Primitive command vectors.

The Job Assignment module takes the Op_algorithm as the control algorithm in a teleoperated control mode. In this mode, the control data comes from the world modeling interface. World modeling obtains the data from the control device specified in the command. A number of controls are possible, including unilateral master-slave control, unilateral joystick control, and generalized bilateral control [6,12]. Teleoperation of the Servo Level will be discussed in much greater detail in a subsequent document.

The Op_algorithm could also indicate that some kind of shared control [12,27] is to be executed, where the Primitive command and the operator would both contribute to the final control. In this mode, the control algorithm would be determined from both Op_algorithm and Servo_algorithm. The servo gains and selection matrices will be chosen from one of the two interfaces. Also, the final attractor set for the manipulator would be a combination of the attractors specified by world modeling and Primitive.

A Job Assignment algorithm allowing the operator input to incrementally modify the Primitive command would be an example of shared control. Suppose that Primitive is issuing rate commands to Servo in end-effector coordinates. The Job Assignment module could compute the rate to be sent to the Planning module as

$$\dot{z}_s = \dot{z}_d + \dot{z}_o ,$$

where $\dot{z}_s$ is the new rate output to the Planning module, and $\dot{z}_d$ is the original rate commanded by Primitive. The vector $\dot{z}_o$ is a modification of the Primitive rate command by the operator.

Another type of shared control could be achieved by choosing elements from both Primitive and the operator to create a new attractor set [10]. For instance, in a hybrid position/force control scheme, the commands computed by Primitive could be used for the position-controlled subspace, and the force-controlled subspace could be given by the operator. This would allow an operator to control the force exerted by the manipulator while it is moving on a surface.

5. Planning Module Interfaces

The Servo Planning module has interfaces to the Job Assignment module, the Execution module, and world modeling, as shown in figure 5. This section describes the interfaces to Job Assignment and world modeling. The Planning to Execution interface is described in section 7.1.

5.1. Job Assignment to Planning Module Interface

The elements of the Job Assignment to Planning module interface are listed below.

$z_d, \dot{z}_d, \ddot{z}_d, \dddot{z}_d$

$f_d, \dot{f}_d$

$K_p, K_v, K_i, K_{pf}, K_{vf}, K_{if}, K_\tau$

S, S'

Algorithm

Time_stamp = (synch_flag, t_p)

Status

These items are of the same form as the corresponding items in the Primitive to Job

Assignment interface. The vectors z_d, $\dot{z}_d$, $\ddot{z}_d$, $\dddot{z}_d$, f_d, and $\dot{f}_d$, form an attractor set for the manipulator. This set may be the combination of Primitive and operator inputs, as discussed above. The servo gains (K's) and the selection matrices (S,S´) are passed through to the Planning module by the Job Assignment module. The Job Assignment module does not modify these parameters. The Time_stamp is also passed directly to the Planning module. The Algorithm parameter indicates the type of servo algorithm to be executed during the control cycle. This parameter is a distillation of the two algorithm parameters, Op_algorithm and Servo_algorithm. Finally, the Status returns the status of the Planning module to the Job Assignment module.

5.2. Planning to World Modeling Interface

The Planning module communicates the following values to world modeling.

$$z_d,\ \dot{z}_d,\ \ddot{z}_d,\ \dddot{z}_d$$

These vectors are the final command vectors for the control. They represent the desired position, velocity, acceleration, and jerk of the manipulator. These values are transmitted to world modeling to be used in computing feedforward compensation terms as described in section 8.2.

6. Planning Module Operation

The purpose of the Planning module is to translate commands from Job Assignment into actions performed by the Execution module during the control period. Thus, the module plans for future action.

In normal operation, the Primitive Execution module would generate path points from its path equations. These path points are generated by substituting desired values of t into the equations $z(t)$, $\dot{z}(t)$, $\ddot{z}(t)$, $\dddot{z}(t)$, generated by the Primitive Planning module. Evaluating the equations for periodic t yields individual inputs for Servo. The Servo Planning module must make sure that the actual (real) time that these points are executed corresponds to the time parameter t used for evaluation in Primitive's Execution module. This correspondence is achieved by the Time_stamp parameter. This parameter tells the Planning module the time for which the current command point was calculated. It is the Planning module's job to see that the Servo Execution module begins executing on the point at the correct real time. To do this the Planning module determines the correspondence between the Execution module's real-time clock and the Primitive Execution module's time parameter t_p. This correspondence is called *synchronization.*

The rate at which the Planning module updates the Servo Execution module depends on the algorithm. Obviously, the Execution module need only be updated when new data is commanded by Primitive. It is possible that the Planning module could linearly interpolate between Primitive commands if they are too sparse. However, the Primitive Execution module should be able to deliver new data at the required rate.

In addition to delivering commands to the Servo Execution module at the proper times, the Planning module must inform the Servo Execution module of new operating modes (algorithms), and see that these are initiated at the correct times. (See the Planning interface description below.) The module must also update world modeling to insure that world modeling can provide the Execution module with the data it needs for control computations, as described above.

7. Execution Module Interfaces

The Execution module has interfaces to the Planning module, world modeling, and the actuator motors. Each of these is described in the following.

7.1. Planning to Execution Interface

The elements of the interface between the Servo Execution module and the Planning module are given below.

z_d, $\dot{z}_d$, $\ddot{z}_d$, $\dddot{z}_d$

f_d, $\dot{f}_d$

K_p, K_v, K_i, K_{pf}, K_{vf}, K_{if}, K_τ

S, S´

Algorithm

Status

These items are passed exactly as they are described in the Job Assignment to Planning interface in section 5.1. The parameters z_d, $\dot{z}_d$, $\ddot{z}_d$, $\dddot{z}_d$, f_d, $\dot{f}_d$ may be interpolated values from those originally received by the Planning module. The rest of the elements, except the last element of the list, are simply passed through to the Execution module. The Status parameter is passed from the Execution module to the Planning module. This parameter indicates when there is a servo error, e.g., excessive command torques.

7.2. World Modeling to Execution Interface

World modeling provides the Execution model with feedback variables and model-based terms to be used in the control computation. Detailed control computations are given in section 8.2. An examination of these may help clarify the nature of the interface parameters given below.

z, $\dot{z}$

f_z

Avoidance torques

τ_θ

Dynamic terms

τ_{act}

The vectors z and $\dot{z}$ are the position and velocity feedback vectors of the manipulator. World modeling determines these parameters by translating sensory processing results into the coordinate system of C_z. Thus, if C_z = (joint) then z is the vector of joint positions, and if C_z = (Cartesian), z is a Cartesian manipulator position. Note that these feedback variables are used to compute the servo error terms of the control equation. Therefore world modeling must update the values provided to the Execution module at the servo rate.

The parameter f_z is the force feedback vector. Again, the vector can be in either Cartesian or joint space, depending on the value of C_z.

The Avoidance torques are vectors of joint torques that are added to the control to achieve avoidance of some condition. One example of the use of this type of parameter is for obstacle avoidance [7,34]. In [34], a hypothetical force pushing away from an obstacle is given by

$$f^* = \begin{cases} \eta\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)\frac{1}{\rho^2}\frac{\partial\rho}{\partial x} & \text{if } \rho \le \rho_0 \\ 0 & \text{if } \rho > \rho_0 \end{cases},$$

where ρ is the distance to the obstacle, η and ρ_0 are constants. This force can be converted to a torque that will add to the control vector and provide motion that tends to avoid the obstacle,

$$\tau_{avd} = J^t(\theta)\Lambda(x)f^* .$$

Here, $J^t(\theta)$ is the transpose of the Jacobian of the manipulator and $\Lambda(x)$ is the Cartesian inertia matrix.

Note that the above formulation generates torques that affect the entire manipulator motion. When the manipulator contains some redundancy, i.e. seven or more degrees of freedom, this can be used to advantage in performing avoidance without modifying the endpoint trajectory. For redundant devices, the relationship

$$\dot{x} = J\,\dot{\theta}$$

still holds, where $\dot{x}$ is the six-dimensional Cartesian velocity and $\dot{\theta}$ is the n>6 -dimensional joint velocity. J is the manipulator Jacobian, which in this case is non-square and has no inverse. The inverse solution in this case would take the form

$$\dot{\theta} = J^*\dot{x} + (I - J^*J)\,\phi ,$$

where $J^* = J^t(J\,J^t)^{-1}$ and ϕ is an arbitrary joint vector. The operator $(I - J^*J)$ maps ϕ into the null space of the Jacobian. The null space vectors correspond to self-motion of the linkage which does not affect the endpoint motion. Thus, avoidance torques can be generated that operate only in this null space. For example, to avoid extremes of torque [11,31], the vector

$$\tau_{avd} = H[\, H\,(I - J^*J)]^*\,(\frac{\tau^+ + \tau^- - 2\tau_\theta}{2}),$$

can be added to the control. Here, H is the inertia matrix, τ^+ and τ^- are the torque extremes, and τ_θ is the sensed torque vector. This vector only acts in the null space of the Jacobian.

In general, avoidance torques can be generated by computing torque vectors for vectors of the form

$$k\frac{\partial g(z)}{\partial z}\ .$$

Such torques will produce motion along the gradient of g(z), thus reducing g(z) through subsequent motion [31,55]. The avoidance torques can operate on the entire motion of the manipulator or can be restricted to null space motions. This technique has been used for avoiding joint limits [34,36,61] and singularities [35,61], as well as obstacles and torque extremes. Note, however, that such avoidances can only be based on "local" criteria. Global techniques are required for the general solution to these problems.

The parameter τ_θ is also a joint space vector. This vector is feedback of sensed torques at the actuators. This type of feedback could be used to create a torque loop around the actuators as described in [39]. However, current digital hardware may be insufficient for the computation speeds required in such loops [47]. This value may also be needed for detecting torque limit violations.

The Dynamic terms provided by world modeling include the model-based terms of the manipulator dynamic equations, used in the Execution module control algorithm. Detailed descriptions can be found in section 8.2, however, the terms include Jacobians, gravity compensation vectors, inertia coefficients, and centrifugal and Coriolis compensations. The computation of these values may depend on either desired position values or on feedback values; world modeling has access to both sets of data. The exact nature of the terms provided to the Execution module depends largely on the algorithm. World modeling should be programmed to provide the terms needed by the algorithms to be used in a given application. The dynamic terms generally do not have to be updated at the servo rate. Updates to these values may often occur at intervals much longer than the servo rate.

The final element of the interface is sent from the Execution module to world modeling. This vector, τ_{act}, is the control output computed by the Execution module. World modeling will need this value for adaptation computations [43,44]. An example of this is the technique used by Miller [44], for learning the relationship between the manipulator state and the command torque.

7.3. Execution to Motor Control Interface

The Execution module outputs a new control vector to the motors at the servo rate. This output may be thought of as armature current or voltage, however, there is most likely additional analog circuitry between the Execution module computer and the motors themselves. At minimum this circuitry must include power amplifiers to drive the motors, but it may also include analog control loops of various types. For example, torque control loops around the individual actuators are usually implemented with analog components [39,47]. Also, some types of motors, such as brushless DC motors, require that the control current be distributed based on motor position. Thus, these devices must have current control loops. For the purposes of this discussion, analog control circuitry such as this will be considered outside the domain of the control architecture.

8. Execution Operation

The Servo Level supports most algorithms for manipulator control. Indeed, the command parameters are general enough to admit a wide variety of servo behaviors. These behaviors can be collected into modes that are selected via the "Algorithm" command specifier. This section describes the mainpulator control problem at the Servo Level, many well-known servo control algorithms, and their implementation in the Servo Executor.

8.1. The Manipulator Control Problem

Each joint of a manipulator is powered by an actuator. For the purposes of this discussion, an actuator consists of an electric motor and, possibly, a gear reducer. The gear reducer is used to achieve low speed, high torque output from high speed, low torque motors. In considering the control of a manipulator one should begin with motor dynamics.

For DC motors, the developed torque is directly related to motor current [10,53],

$$\tau = K_\tau i,$$

where K_τ is the torque constant, and i can be either armature or field current. For the purpose of discussion, consider i to be armature current. Characterizing the motor as having certain inertia, J, and damping, B, the motor dynamics can be written [53]

$$\tau = J \ddot{\phi} + B \dot{\phi},$$

where ϕ is the angle of the motor output shaft. (For a linear actuator, τ can be interpreted as force, and ϕ as linear position. However, for purposes of discussion, the following simply refers to rotational actuators.) Armature current can be used as the control variable via

$$\frac{1}{K_\tau} (J \ddot{\phi} + B \dot{\phi}) = i .$$

The voltage equation of the armature circuit is given [53] by

$$u = R i + L \frac{di}{dt} + K_e \dot{\phi} .$$

Here, u is the voltage across the armature winding and $K_e \dot{\phi}$ is the back emf. Large DC motors used for manipulators can have massive windings with large inductances. For these motors it is important to consider the inductance, L, in the motor dynamics [25]. By plugging the previous equation and its derivative into the armature voltage equation, a third-order motor dynamics equation is obtained.

$$u = \left(\frac{LJ}{K_\tau}\right)\dddot{\phi} + \left(\frac{RJ+LB}{K_\tau}\right)\ddot{\phi} + \left(\frac{RB+K_eK_\tau}{K_\tau}\right)\dot{\phi}$$

In this dynamical system, the control variable would be the voltage u. Note that the open-loop characteristics of voltage control may make it easier to achieve accurate position control, while making force control more difficult [37]. However, with the appropriate feedback terms in the control, comparable behavior should be achievable with either approach.

Similar dynamical equations can be developed for brushless DC motors and variable reluctance (stepping) motors. However, control of these devices typically involves distribution of stator current over three phases. The distribution is based on rotor position so that closed loop current control with a rotor position sensor must be employed [10].

The dynamics of other electric motors can be described with similar equations. For example, the dynamics of the two-phase induction (AC) motor can be described by

$$u = \left(\frac{J}{K_e}\right)\ddot{\phi} + \left(\frac{B-K_n}{K_e}\right)\dot{\phi}$$

as a first approximation [53]. Here, u is the control field voltage.

So far, the discussion has not included the external load of the actuator. The external load torque appears in the dynamical equation of the DC motor actuator as

$$J\ddot{\phi} + B\dot{\phi} + \frac{\tau_{ext}}{n} = K_\tau i\,,$$

where n is the gear reduction ratio of the actuator [25]. For a direct-drive actuator, i.e. an actuator with no gear reducer between the motor and the link, n is unity. In this type of actuator it may be quite important to consider the armature winding inductance. The complete dynamical equation is

$$u = \left(\frac{LJ}{K_\tau}\right)\dddot{\phi} + \left(\frac{RJ+LB}{K_\tau}\right)\ddot{\phi} + \left(\frac{RB+K_eK_\tau}{K_\tau}\right)\dot{\phi} + \left(\frac{L}{nK_\tau}\right)\dot{\tau}_{ext} + \left(\frac{R}{nK_\tau}\right)\tau_{ext}\,.$$

This equation can be quite complicated. The external load torques are, in general, dependent on the entire manipulator's position and velocity. The equation of external load torques is given [17,54] by

$$\tau = A(\theta)\,\ddot{\theta} + b(\theta,\dot{\theta}) + g(\theta)\;,$$

where τ is the vector of external load torques for all the actuators of the manipulator and θ is the vector of actuator (joint) positions. $A(\theta)$ is the position-dependent inertial coefficient matrix, $b(\theta,\dot{\theta})$ is the contribution of centrifugal and Coriolis effects, and $g(\theta)$ is the loading due to gravity. Thus, the complete dynamical equation for an actuator including winding inductance effects is 21st order for a seven-actuator manipulator.

Fortunately, it is not always necessary to consider the effects of all terms of the equation. For example, for smaller gear-drive DC motor actuators, the motor inductance is negligibly small [10]. Thus, its effects can be ignored. However, many of the nonlinear effects must be considered and this motivates the discussion of complex control algorithms in the next section. The discussion will treat various components of the control separately, and will concentrate mainly on the external torque terms. Note that the motor inertia and damping terms can usually be subsumed in the external torque equation. The discussion will generally assume that an actuator can deliver a desired torque for some control input. As discussed above, for DC motors there is a direct relationship between input current and output torque. However, gear reducers may contain significant friction and nonlinearities that prevent this relationship from being realized at the actuator output. In these cases the actuators may be equipped with low-level torque servos [39] so that a desired output torque is reliably achieved.

8.2. Manipulator Control Schemes

The problem of Servo is to provide control of the manipulator actuators described by the above equations. This involves providing stable and accurate positioning of the end-effector at fixed points, accurate tracking of desired trajectories x(t), and characterizable interaction with the environment.

A common control algorithm used in commercial robots is proportional-derivative (PD) position control [15,17]. This algorithm does not provide good path control, although it is simple and globally stable for an appropriate choice of gains [10]. To implement this algorithm in the Executor of the Servo Level, a control equation of the form

$$K_p (\theta_d - \theta) + K_v (\dot{\theta}_d - \dot{\theta}) + \ddot{\theta}_d = \tau_{act}$$

would have to be executed every control cycle. Here, τ_{act}, the vector of joint torques, represents the desired outputs of the actuators. The vectors θ and $\dot{\theta}$ are the feedback positions and velocities from sensors at the manipulator's joints. These values are provided to the Executor by world modeling. All the remaining components of the control equation are available in the command specification from Primitive. These are passed to the Executor by the Servo planner. Thus, one line of code should realize an implementation of the PD algorithm in the Servo Executor.

The PD algorithm is often augmented with an integral term to eliminate steady-state position errors. This PID algorithm is realized by the equation

$$K_p (\theta_d - \theta) + K_v (\dot{\theta}_d - \dot{\theta}) + K_i \int (\theta_d - \theta) + \ddot{\theta}_d = \tau_{act}.$$

The symbol $\int$ is shorthand notation for a call to a function which implements a digital integrator, e.g.,

$$\int_{t_0}^{t} (\theta_d - \theta) \Rightarrow \quad sum = sum + \Delta T(\theta_d - \theta)\,,$$

where ΔT is the sample period.

Since it is often easier to visualize a task in Cartesian coordinates, many control algorithms are formulated in this domain. The PID servo algorithm in Cartesian coordinates is given by the control equation [17,55]

$$J^t(\theta)[\, K_p(x_d - x) + K_v(\dot{x}_d - \dot{x}) + K_i \int (x_d - x) + \ddot{x}_d\,] = \tau_{act}\,.$$

The vectors x in this equation represent Cartesian positions and orientations. To get this Cartesian interpretation of the servo variables, Primitive must specify C_z= (world) in the command specification. This input, coupled with the "Algorithm" specifier, results in the Executor selecting code that implements the above equation. The term $J^t(\theta)$ is the transpose of the manipulator Jacobian matrix, value of which is dependent on joint positions, which is used to transform the control from the Cartesian domain into the actuator command τ_{act}.

The problem with PID controllers is that no one choice of gains can provide good control throughout the manipulator's work volume. This is because linear gains can not model the nonlinear coupling between joints. It is possible with the proposed control architecture to alleviate this problem by using different gains for different regions of the work volume. Since Primitive commands these gains, it could select appropriate gains for a given position. However, a better solution is to compensate for the nonlinear dynamics in the control.

World modeling can generate compensating torques based on a model of the manipulator's dynamics. For example, to eliminate gravity loading disturbances on a PD algorithm, a control [17,55]

$$K_p\,(\theta_d - \theta) + K_v\,(\dot{\theta}_d - \dot{\theta}) + \ddot{\theta}_d + \tau_{gravity}(\theta) = \tau_{act}$$

can be used. The vector $\tau_{gravity}(\theta)$ is composed of the n torques necessary to compensate for the gravity loading of each joint. World modeling computes this vector less frequently than the servo rate, and provides the values to the Executor in the same manner as it provides the feedback variables.

At higher operating speeds it may also be desirable to compensate for inertial and centrifugal effects [4,23,38,54]. These terms are also made available by world modeling so that

$$M(\theta)[\, K_p\,(\theta_d - \theta) + K_v\,(\dot{\theta}_d - \dot{\theta}) + \ddot{\theta}_d\,] + \tau_{cent.}(\dot{\theta},\theta) + \tau_{gravity}(\theta) = \tau_{act}$$

can be easily implemented in the Executor. Here, the matrix $M(\theta)$ is the inertia coefficient and $\tau_{cent.}(\dot{\theta},\theta)$ is the vector of centifugal and Coriolis compensation torques, both computed from the manipulator model. Compensation for other types of disturbances, such as friction, can be included in the control provided a model of the disturbance exists. In some cases, such models can be learned [18,24,43].

The above control can be seen as a direct implementation of the dynamic equation

$$\tau = A(\theta)\,\ddot{\theta}^* + b(\theta,\dot{\theta}) + g(\theta)$$

for computing the joint torques, (for this reason it is often referred to as the "computed torque" algorithm). The nonlinear terms act to linearize and decouple the dynamics so that $\ddot{\theta}^*$ can be replaced by a linear control law,

$$\ddot{\theta}^* = K_p\,(\theta_d - \theta) + K_v\,(\dot{\theta}_d - \dot{\theta}) + \ddot{\theta}_d\,.$$

This algorithm will provide good performance when the dynamic parameters are known, as the linearization will be effective. Determining the parameters accurately, however, is quite difficult. For this reason the control is often augmented to achieve more robustness. One technique for doing this is by adding a switching term to the linear control $\ddot{\theta}^*$ so that the system is forced to remain near the desired trajectory no matter the uncertainty in the dynamic parameters [9,62].

In the Cartesian domain the computed torque control algorithm takes the form [17,35,40]

$$J^t(\theta)M_x(\theta)[\, K_p(x_d-x) + K_v(\dot{x}_d-\dot{x}) + \ddot{x}_d\,] + \tau_{cent.x}(\dot{\theta},\theta) + \tau_{gravity}(\theta) = \tau_{act}\,.$$

Here the dynamic terms $M_x(\theta)$ and $\tau_{cent.x}(\dot{\theta},\theta)$ are computed with respect to the Cartesian coordinate system. The relationship of these terms to the joint space terms used above is given [35] by

$$M_x(\theta) = J^{-t}(\theta)M(\theta)\,J^{-1}(\theta)$$

$$\tau_{cent.x}(\dot{\theta},\theta) = \tau_{cent.}(\dot{\theta},\theta) - M(\theta)\,J^{-1}(\theta)\;\dot{J}(\theta)\,\dot{\theta}\,.$$

Here, $J^{-t}(\theta)$ is the transpose of the inverse Jacobian. Note that the computed torque algorithm could also include compensation for friction in the actuators (provided a good model of friction exists) by including a term $\tau_{friction}(\dot{\theta},\theta)$ in the control [17].

In some algorithms, a feedforward technique is used to decouple the actuators. This process is generally less sensitive to errors in the dynamic model. World modeling can also provide these terms for use in control algorithms. For example, a common feedforward control [4,8,19,44] can be performed in the Executor by

$$\tau_{model}(\,\ddot{\theta}_d, \dot{\theta}_d, \theta_d\,) + K_p\,(\theta_d - \theta) + K_v\,(\dot{\theta}_d - \dot{\theta}) + \ddot{\theta}_d = \tau_{act}\,.$$

In this case, all of the dynamic compensating terms are obtained as one torque vector. This is possible because of the feedforward nature of the control. However, individual feedforward terms, e.g., gravity, friction, etc., could be obtained as well.

Another type of feedforward algorithm can also be implemented by using this model torque vector. In the linear modern control approach [16,57], it may be possible to compute the control vector without feedback. This approach uses the desired state vector $(\theta_d \mid \dot{\theta}_d)$ and the linear state vector model of the manipulator to compute the control τ_{model}. The servo algorithm in this case reduces to

$$\tau_{model} = \tau_{act}\,.$$

Obviously, the control can be made more robust using feedback [57]. Linear models, however, often represent gross approximations to nonlinear systems such as robots.

An even simpler control

$$f_d = \tau_{act}$$

might be invoked in the Execution module. Here C_z= (joint), so that the command to Servo is the desired actuator torque. This type of operation might arise when Primitive needs to plan torque profiles for the actuators, as in the time-optimal control [13].

These particular feedforward algorithms have a number of limitations [25]. One significant problem is that they don't take into account motor dynamics. It may be advantageous to consider a control in motor coordinates, such as [24,25]

$$u = K_p(z_d - z) + v(\dot{z}_d, \ddot{z}_d, \dddot{z}_d),$$

where z represents motor positions, as specified by C_z= (motor), and u is the control input to the motors, which in this case would be the applied armature voltage. The feedforward term v( ) is a linear function of the desired velocity, acceleration and jerk. The coefficients of the function are based on the parameters in the third-order motor model. Thus, as in the above, world modeling computes the feedforward term for the Execution model.

Chen [15] describes a lag-lead compensator for use with PD controllers, including PD controllers used in the scheme just mentioned. Such compensators serve to increase the gain margin of the system, allowing for increased stability with high gains. The compensator acts to modify u based on the error signal. The equations of compensation,

$$w_1(k+1) = k_1(l_1 - d_1)\, e(k) - d_1 w_1(k),$$

$$w_2(k+1) = k_2(l_2 - d_2)(w_1(k) + k_1 e(k)) - d_2 w_2(k),$$

$$\Delta u(k) = b[\, k_2(w_1(k) + k_1 e(k)) + w_2(k)\,],$$

can be easily implemented in the Executor with a few lines of code. In these equations, w_1 and w_2 are state variables of the compensator, and e is the error signal (z_d-z). The remaining symbols, k_1, k_2, l_1, l_2, d_1, d_2, and b, are constants determined in the z-domain derivation of the lag-lead compensator. See [15] for more details.

This type of compensator also appears in other control schemes. The stiffness control algorithm of Salisbury [51] has the form

$$K_p(\theta_d - \theta) + f_d + K_v M(\theta)(\dot{\theta}_d - \dot{\theta}) + \tau_{friction}(\dot{\theta}, \theta) + \tau_{gravity}(\theta)$$
$$+ K_{pf}\,\text{Lead-lag}[\, K_p(\theta_d - \theta) + (f_d - f_\theta)] + K_{if} \int \text{Dblmt}[\, K_p(\theta_d - \theta) + (f_d - f_\theta)] = \tau_{act}.$$

The f's are vectors of Cartesian forces transformed to joint space, i.e. $f_\theta = J^t(\theta) f_x$. (In the same way as z is specified as x or θ in the position control, the force feedback f_z is written as f_x or f_θ to distinguish the selected coordinate system as Cartesian or joint coordinates. The desired force f_d is obviously in the same coordinates as this feedback term and will appear only as f_d in the notation.) The function Lead-lag( ) has the form described above, and the function Dblmt( ), a deadband and limiting function, can be easily implemented as well.

There are a number of other ways of incorporating force feedback into manipulator control algorithms [59]. Force control known as damping or accommodation involves relating sensed forces to velocities linearly. This type of control is achieved by [58]

$$K_v[\dot{\theta}_d - K_{pf} f_\theta - \dot{\theta}] + K_p[\int (\dot{\theta}_d - K_{pf} f_\theta) - \theta] = \tau_{act}\,.$$

Another scheme for including force information is hybrid force/position control [4,42,49]. The algorithm for the Servo Executor is

$$\begin{aligned} &K_p J^{-1}(\theta)S'(x_d-x) + K_v J^{-1}(\theta)S'(\dot{x}_d-\dot{x}) + K_i \int J^{-1}(\theta)S'(x_d-x) \\ &\qquad + J^t(\theta)Sf_d + K_{pf} J^t(\theta)S(f_d-f_x) + K_{if} \int \mathrm{Lmt}[\, J^t(\theta)S(f_d-f_x)\,] = \tau_{act}\,, \end{aligned}$$

where the S and S′ are selection matrices for selecting certain axes as position controlled and others as force controlled. Lmt( ) is a limiting function which is required to prevent the destabilizing influence of the integrator. Stabilization might also be achieved by incorporating a control term for force rate feedback if it is available [5,49]. It may also be desirable to eliminate the Jacobian inverse from the control, giving something of the form

$$\begin{aligned} &J^t(\theta)[\, K_p S'(x_d-x) + K_v S'(\dot{x}_d-\dot{x}) + K_i \int S'(x_d-x) \\ &\qquad + Sf_d + K_{pf}\, S(f_d-f_x) + K_{vf}\, S(\dot{f}_d - \dot{f}_x) + K_{if} \int S(f_d-f_x)\,] = \tau_{act}\,. \end{aligned}$$

In some systems the position error gain is set according to how much stiffness (compliance) is desired [48,51,52]. A more general approach is to set all the gains so that the manipulator exhibits a certain overall impedance [5,26,28,33]. This can be implemented by having Primitive set the values of the K's appropriately in the command specification to Servo.

It is possible to do model-based compensation with a combined position and force control scheme [29,35]. This approach requires that the dynamics of the manipulator be formulated with respect to the Cartesian frame in which errors are to be computed. The dynamics can be reformulated for a new coordinate system C_z= (world, T_w, T_e) as follows.

Let x be the end-effector coordinates for which the relationship

$$\delta x = J(\theta)\delta\theta$$

is already established. Given $x_w = T_w\, x$, J_w can be computed such that

$$\delta x_w = J_w \delta x = J_w J(\theta)\delta\theta = J_z(\theta)\delta\theta\,.$$

Then, given the manipulator dynamics of

$$A(\theta)\,\ddot{\theta} + b(\theta,\dot{\theta}) + g(\theta) = \tau\,,$$

the dynamics can be reformulated in the chosen coordinates [35] as

$$\Lambda(x_w)\,\ddot{x}_w + \mu(x_w,\dot{x}_w) + p(x_w) = f_w\,,$$

where

$$\Lambda(x_w) = J_z^{-t}(\theta)A(\theta)J_z^{-1}(\theta)\,,$$

$$\mu(x_w,\dot{x}_w) = J_z^{-t}(\theta)b(\theta,\dot{\theta}) - \Lambda(\theta)J_w\dot{J}(\theta)\dot{\theta}\quad,$$

$$p(x_w) = J_z^{-t}(\theta)g(\theta)\,.$$

The control algorithm using this reformulation would take the form

$$J^t(\theta)\{\ S[\ f_d + K_{pf}\,(f_d\text{-}f_x)\ ] + \ \Lambda(\theta)SK_{vf}\,\dot{x} +$$
$$\Lambda(\theta)S'[\ K_p(x_d\text{-}x) + K_v(\dot{x}_d\text{-}\dot{x}) + \ddot{x}_d\ ] + \mu(\theta,\dot{\theta})\ \} + g(\theta) = \tau_{act}\,.$$

The selection matrices S, S′ in this control have the role of Khatib's task specification matrices. They can be modified to handle the new end-point of the manipulator specified by the transformation T_e. Given a known task specification matrix S_0 for C_z= (world, T_w, I),

$$S_0 = \begin{pmatrix} S_f^{\,t}\,\Sigma_f\,S_f & 0 \\ 0 & S_\tau^{\,t}\,\Sigma_\tau\,S_\tau \end{pmatrix}$$

a task specification matrix for C_z= (world, T_w, T_e) can be computed by

$$S = \begin{pmatrix} S_f^{\,t}\,R_e^{\,t}\,\Sigma_f\,R_e S_f & 0 \\ 0 & S_\tau^{\,t}\,R_e^{\,t}\,\Sigma_\tau\,R_e\,S_\tau \end{pmatrix}$$

where R_e is the rotation matrix corresponding to the transformation T_e.

The position error gain terms in the algorithms given so far can be thought of as attractor fields, i.e. the manipulator position is compelled toward the desired position to a degree specified by the position gain [54,55]. The idea can be extended to repellent fields surrounding obstacles [7,34,41], joint limits [31,34,36,61], and singularities [35,61], as discussed in section 6.2. The resulting inputs are avoidance torque vectors with magnitudes proportional to the strength of the repellent force fields at the current manipulator position. Since they are additive, they can be easily introduced in a control such as

$$J^t(\theta)[\ K_p(x_d\text{-}x) + K_v(\dot{x}_d\text{-}\dot{x}) + \ddot{x}_d\ ] + \tau_{avd} = \tau_{act}\,,$$

where τ_{avd} is the avoidance torque vector.

With regard to collisions with obstacles and other undesirable behavior, there is a way to take immediate (reflexive) action within the Servo Level. This type of action can be used to minimize damage between the time the accident is detectable and the time Primitive can act or the system can be shutdown. Reflexive behavior is implemented in the Servo Level as a "conditional" algorithm. For example, the algorithm

$$\begin{cases} J^t(\theta)[\, K_p(x_d - x) + K_v(\dot{x}_d - \dot{x})\,] = \tau_{act}, & |\tau_\theta| < L, \\ \\ -\, K_{pf}\, J^t(\theta)\, f_x = \tau_{act}, & \text{after } |\tau_\theta| \geq L, \end{cases}$$

where L is some prespecified constant vector. This particular algorithm would try to servo to a position and a velocity unless a torque limit was reached. After the torque limit, the manipulator would accommodate any forces at the end-effector. The reader should note that the parameter L is fixed and can not be modified by Primitive, i.e. there is no provision for the parameter in the command specification. The reason for this is that Primitive should be monitoring the sensors and changing the command input to Servo anyway. The conditional algorithm is only meant for emergency situations, Primitive being able to detect and modify the Servo input under normal conditions. Note that this type of construct must be used to handle servo errors, such as excessive joint torques, for safety reasons.

Encountering singularities can cause problems with excessive joint rates if the control is not handled properly. It may not be possible to avoid excessive joint rates through avoidance torques [11], and it is not desirable to shutdown the robot if some type of correction can be made. A number of alternate ways of dealing with singularities is described in [1]. These generally involve limiting the excessive terms to some maximum value and continuing with the distorted control. Ideally, Primitive should not command motions along a singular direction. This means that in the region of a singularity, Cartesian commands to Servo should be properly scaled with respect to manipulability [61]. At the Servo Level, the control algorithm should monitor the singular expressions in the determinant of the Jacobian and modify the control when a singular region is entered. In the region near a singularity, the manipulator can be treated as redundant with respect to end-effector motion in the subspace orthogonal to the singular direction [35,55]. A gradient term on the manipulability measure can then be used to control motion in the associated null space [35,61].

Although the variety of control algorithms presented thus far include algorithms with appropriate terms for compensating the nonlinear, coupled dynamics of manipulators, these compensation techniques require that an accurate model of the manipulator be determined a priori. Such models can rarely be determined. Manipulator systems have parameters which are difficult to identify. These parameters are often likely to be time-varying, as well. The very nature of manipulator activity requires that the dynamics of the system is continually changing, e.g., variation in payloads as objects are manipulated. Therefore, it is important to consider adaptive techniques for determining model-based compensations.

A variety of adaptive control algorithms for manipulators are given in the literature [32]. Some type of adaptation is possible with almost all of the control algorithms mentioned above. As an example, consider one form of model reference adaptive control [20,21]. This control is based on decoupled joint dynamics of the form

$$\alpha_i(t)\,\ddot{\theta}_i + \beta_i\,\dot{\theta}_i + \theta_i = r_i(t)\,.$$

Here, the inertia coefficient $\alpha_i(t)$ of a joint is time-varying. This is often called the "effective" inertia of the joint. It represents the inertia seen by the i^{th} joint, including the coupling effects between joints of the manipulator, at time t. The function $r_i(t)$ is the reference input to the manipulator, the desired position. It is desired that each joint of the manipulator exhibit a specific second-order behavior that is known a priori. Although, $\alpha_i(t)$ and β_i are not known, due to uncertainty in the dynamic model, the desired behavior can be specified by the model

$$a_i \ddot{y}_i + b_i \dot{y}_i + y_i = r_i(t) .$$

The model following approach involves making each joint exhibit the behavior of the model by adjusting the proportional-derivative gains of the control feedback. This is to be done such that

$$f = \frac{1}{2} (q_0^i e_i + q_1^i \dot{e}_i + q_2^i \ddot{e}_i)^2$$

is minimized, where $e_i = y_i - \theta_i$ is the error between model and system. As derived in [21], the adaptation rules for the PD gains are given by

$$\dot{K}_p^i = - \frac{K_p^i}{a_i} (q_0^i e_i + q_1^i \dot{e}_i + q_2^i \ddot{e}_i) (q_0^i u_i + q_1^i \dot{u}_i + q_2^i \ddot{u}_i)$$

and

$$\dot{K}_v^i = K_p^i (q_0^i e_i + q_1^i \dot{e}_i + q_2^i \ddot{e}_i) (q_0^i w_i + q_1^i \dot{w}_i + q_2^i \ddot{w}_i) - \frac{K_v^i}{K_p^i} \dot{K}_p^i ,$$

where u_i and w_i are the solutions to the equations

$$a_i \ddot{u}_i + b_i \dot{u}_i + u_i = - \ddot{y}_i ,$$

$$a_i \ddot{w}_i + b_i \dot{w}_i + w_i = - \dot{y}_i .$$

There are a number of techniques for digitally generating the solutions to these equations [50]. One simple technique would be to transform them into difference equations by using the formula

$$\frac{d}{dt} x(t) \approx \frac{x(n) - x(n-1)}{\Delta T} .$$

An important limitation of the model reference adaptive control scheme described above is that it only adapts to perturbations in the assumed second-order model. Although this works quite well for a given trajectory, the control must be readapted for a new trajectory. In other words, the control never learns the true dynamic model of the manipulator. There are schemes which involve learning the true dynamic parameters of the system [18,44,45]. The approach of [18] uses the computed torque algorithm for control, with adaptation of model parameters based on the servo error. This particular type of learning (also called "identification") may belong in the world modeling module, since it involves improving the internal knowledge of the world. Adaptation with a control algorithm that uses the feedforward form of the dynamics is given in [44]. Dynamic coefficients can also be learned in the feedforward algorithm that includes motor inductances [25].

8.3. Implementational Issues

The most crucial implementational consideration at the Servo Level is speed. The Execution module must execute at the servo rate. That is, the module must read its inputs, from the Planning module and world modeling, perform the necessary computations, and update the output control vector in a fixed interval of time. This interval is typically on the order of 1 to 5 milliseconds. Failure to compute at the required rate may lead to system instability. This section attempts to give a feel for the computational load suggested by the discussion of section 8.2. In addition to the computational rate issue, the implementation must consider the data rates suggested by the proposed interfaces. This section examines this issue as well.

As a example of required computational loads consider Khatib's operational space control [35]. This is one of the more complicated controls presented in this paper. The control algorithm can be reformulated, as shown in [35], to minimize the amount of computation. The reformulated algorithm is

$$J^t(\theta)\{\ S[\ f_d + K_{pf}(f_d - f_x)\ ] + \\ \Lambda(\theta)(\ SK_{vf}\dot{x} + S'[\ K_p(x_d - x) + K_v(\dot{x}_d - \dot{x}) + \ddot{x}_d\ ])\ \} + \tilde{b}(\theta,\dot{\theta}) + g(\theta) = \tau_{act},$$

where $\tilde{b}(\theta,\dot{\theta})$, $g(\theta)$, $J^t(\theta)$, and $\Lambda(\theta)$ are supplied directly by world modeling. The amount of computation required of the Execution module for a six degree-of-freedom manipulator can be computed as follows.

$K_{vf}\dot{x}$	=>	6 multiplies	
$f_d + K_{pf}(f_d - f_x)$	=>	6 multiplies and	12 adds
$K_p(x_d - x) + K_v(\dot{x}_d - \dot{x}) + \ddot{x}_d$	=>	12 multiplies and	24 adds
matrix mult. by S and S′	=>	54 multiplies and	36 adds
vector additions inside { }	=>		12 adds
matrix mult. by $\Lambda(\theta)$	=>	36 multiplies and	30 adds
matrix mult. by $J^t(\theta)$	=>	36 multiplies and	30 adds
remaining vector additions	=>		12 adds
TOTAL		150 multiplies and	156 adds

This analysis uses the fact that the K's are diagonal matrices and the S's are block diagonal to avoid always doing full matrix multiplies. The result is that the Execution module must perform approximately 300 real operations each control cycle. For a control cycle of 1 ms, this implies that the control processor must be capable of at least 300 K real operations per second. Actually, Khatib [35] uses a servo rate of 5 ms, so that only 60 K real operations per second is required. In terms of floating point calculations, this is well within the rate of current microprocessors with floating point coprocessors. The 68020/68881 system is capable of 1.24 M Whetstones per second [60] (one Whetstone being a set of floating point operations). In terms of fixed point calculations (which can be accomplished by a 68020 alone), the algorithm can be computed in approximately 11880 clock cycles, assuming 60 cycles per multiply and 20 cycles per addition [43]. For a 20 MHz clock rate, the algorithm could compute in 0.59 ms. Of course, there will be overhead associated with Execution module operation. However, the total time should still be well within the required range.

A similar algorithm for a seven degree-of-freedom arm is given [35] by

$$J_r^t(\theta)\{\ S[\ f_d + K_{pf}\,(f_d - f_x)\ ] + \Lambda_r(\theta)(\ SK_{vf}\,\dot{x} + S'[\ K_p(x_d - x) + K_v(\dot{x}_d - \dot{x}) + \ddot{x}_d\ ] + K_{vq}\,\dot{x}\ )\ \} - K_{vq}A_r(\theta)\,\dot{\theta} + \tilde{b}_r(\theta,\dot{\theta}) + g_r(\theta) = \tau_{act}\,.$$

Here $\tilde{b}_r(\theta,\dot{\theta})$, $g_r(\theta)$, $J_r^t(\theta)$, and $\Lambda_r(\theta)$ are the dynamic terms of the redundant system, supplied directly by world modeling. The additional K_{vq} terms are to determine joint torques for components of the Jacobian null space. Reasoning as before to obtain the amount of computation required of the Execution module, the values shown below are obtained.

$K_{vf}\,\dot{x}$	=>	6 multiplies	
$f_d + K_{pf}\,(f_d - f_x)$	=>	6 multiplies and	12 adds
$K_p(x_d - x) + K_v(\dot{x}_d - \dot{x}) + \ddot{x}_d$	=>	12 multiplies and	24 adds
$K_{vf}\,\dot{x}$	=>	6 multiplies	
matrix mult. by S and S′	=>	54 multiplies and	36 adds
vector additions inside { }	=>		18 adds
matrix mult. by $\Lambda_r(\theta)$	=>	36 multiplies and	30 adds
$K_{vq}A_r(\theta)\,\dot{\theta}$	=>	56 multiplies and	42 adds
matrix mult. by $J_r^t(\theta)$	=>	42 multiplies and	35 adds
remaining vector additions	=>		28 adds
TOTAL		218 multiplies and	225 adds

Thus, for this highly complex algorithm, approximately 450 K real operations per second are required for a 1 ms servo rate, and 90 K real operations for a 5 ms servo rate. This seems within the capabilities of current computers. Simpler control algorithms may be used when only limited computational facilities are available.

The majority of the computation required for the Servo Level is contained in computing the dynamic terms such as inertia coefficients, Jacobians and inverse Jacobians [14,30,35].

Depending on the algorithm, additional processors may need to be provided for these computations. This approach is used in many manipulator control implementations [35,46]. It has been demonstrated that, with currently available processing elements, it is possible to compute the entire dynamic equation of a six d.o.f. manipulator in less than a millisecond [46]. Memory look-up schemes could possibly provide even faster dynamics [2,44]. However, as stated earlier, it may only be necessary to compute these terms at rates of 10 to 16 ms. Thus, these computations should also be well within the capabilities of current computers.

The amount data in the interfaces to the Execution module can be estimated as follows. For the Planning module interface,

$z_d, \dot{z}_d, \ddot{z}_d, \dddot{z}_d$	=>	4 x 7 = 28 words
$f_d, \dot{f}_d$	=>	2 x 7 = 14 words
$K_p, K_v, K_i, K_{pf}, K_{vf}, K_{if}, K_\tau$	=>	7 x 7 = 49 words
S, S'	=>	2 x 2 x 9 = 36 words
Algorithm	=>	1 word ,

where the same assumptions for the K's and S's are made as before. The most appropriate size for a "word" in this interface would probably be 32 bits. Thus, the interface requires that about 128 words or 512 bytes be passed every 5 ms or so. Note that only the first 42 words of the interface will need to be passed most of the time. Since current microprocessor buses support data rates of at least 5 MHz [46], this entire interface could be transmitted in 25.6 μs.

For the world modeling to Execution module interface consider,

$z, \dot{z}, f_z$	=>	3 x 7 = 21 words
$\tau_{obs}, \tau_{jl}, \tau_\theta$	=>	3 x 7 = 21 words
$J^t(\theta)$	=>	7 x 6 = 42 words
$M(\theta)$	=>	7 x 7 = 49 words
$\tau_{cent.}(\dot{\theta},\theta)$	=>	7 words
$\tau_{gravity}(\theta)$	=>	7 words
$\tau_{friction}(\dot{\theta},\theta)$	=>	7 words.

Here, the τ_{obs} and τ_{jl} are avoidance torque vectors for obstacle avoidance and joint limit avoidance, respectively. This interface represents the amount of information required for something like operational space control. The entire interface requires about 154 words, which are again probably 32 bits wide. However, only the first 21 words of the interface need to be passed at the servo rate. Thus, the 30.8 μs transfer time for a 5 MHz bus transfer of the whole interface is easily achievable.

9. Conclusion

This document has given a description of Servo Task Decomposition for a manipulator.

This description should motivate an implementation of the three principal modules: Job Assignment, Planning, and Execution. In addition, the document should serve as a companion guide to the Servo Level world modeling and Primitive Level documents, giving the basic output requirements of those system components.

The reader should note that the references which follow form a comprehensive review of the basic concepts of manipulator servo control. The references should be consulted to answer questions of applicability and implementation not discussed in this document.

10. References

[1] Aboaf, E. W., Paul, R. P., "Living with the Singularity of Robot Wrists," IEEE Conf. Robotics & Automation, Raleigh, NC, March 1987.

[2] Albus, J. S., "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)," Jour. Dyn. Sys., Meas., and Control, September 1975.

[3] Albus, J. S., McCain, H. G., Lumia, R., NASA/NBS Standard Reference Model Telerobot Control System Architecture (NASREM), NASA Document SS-GSFC-0027, December 1986.

[4] An, C. H., Trajectory and Force Control of a Direct Drive Arm, Ph.D. Thesis, MIT Artificial Intelligence Lab, 1986.

[5] Anderson, R. J., Spong, M. W., "Hybrid Impedance Control of Robotic Manipulators," IEEE Conf. Robotics & Automation, Raleigh, NC, March 1987.

[6] Andre, G., Fournier, R., "Generalized End Effector Control in a Computer Aided Teleoperation System with Application to Motion Coordination of a Manipulator Arm on an Oscillating Carrier," '85 ICAR, Tokyo, September 1985.

[7] Andrews, J. R., Impedance Control as a Framework for Implementing Obstacle Avoidance in a Manipulator, Master's Thesis, Dept. of Mech. Eng., MIT, February 1983.

[8] Asada, H., Kanade, T., Takeyama, I., "Control of a Direct-Drive Arm," Jour. Dyn. Sys., Meas., and Control, Vol. 105, No. 3, 1983.

[9] Asada, H., Slotine, J.-J. E., Robot Analysis and Control, John Wiley & Sons, New York, 1986.

[10] Asada, H., Youcef-Toumi, K., Direct-Drive Robots: Theory and Practice, MIT Press, Cambridge, MA, 1987.

[11] Baillieul, J., et. al., "Kinematically-redundant Robot Manipulators," Workshop on Space Telerobotics, Pasadena, CA, January 1987.

[12] Bejczy, A. K., "Robots as Man-Extension Systems in Space," IFAC Ninth Triennial World Congress, Budapest, Hungary, 1984.

[13] Bobrow, J. E., Dubowsky, S., Gibson, J. S., "Time-Optimal Control of Robotic Manipulators Along Specified Paths," Int. Jour. Robotics Research, Vol. 4, No. 3, 1985.

[14] Burdick, J. W., "An Algorithm for Generation of Efficient Manipulator Dynamic Equations," IEEE Conf. Robotics & Automation, San Fransisco, CA, April 1986.

[15] Chen, Y. L., "Frequency Response of Discrete-time Robot Systems - Limitations of PD Controllers and Improvements by Lag-Lead Compensation," IEEE Conf. Robotics & Automation, Raleigh, NC, March 1987.

[16] Coiffet, P., Modeling and Control, Kogan Page, London, 1983.

[17] Craig, J. J., Introduction to Robotics: Mechanics and Control, Addison-Wesley, Reading, MA 1986.

[18] Craig, J. J., Hsu, P., Sastry, S. S., "Adaptive Control of Mechanical Manipulators," IEEE Conf. Robotics & Automation, San Fransisco, CA, April 1986.

[19] Desa, S., Roth, B., "Synthesis of Control Systems for Manipulators Using Multivariable Robust Servomechanism Theory," Int. Jour. Robotics Research, Vol. 4, No. 3, 1985.

[20] Donalson, D. D., Leondes, C. T., "A Model Referenced Parameter Tracking Technique for Adaptive Control Systems," Trans. IEEE Appl. & Industry, Vol. 82, No. 68, September 1963.

[21] Dubowsky, S., DesForges, D. T., "The Application of Model-referenced Adaptive Control to Robotic Manipulators," Jour. Dyn. Sys., Meas., and Control, Vol. 101, September 1979.

[22] Fiala, J. C., Lumia, R., Albus, J. S., "Servo Level Algorithms for the NASREM Telerobot Control System Architecture," SPIE Vol. 851 Space Station Automation III, Cambridge, MA, November 1987.

[23] Freund, E., "Fast Nonlinear Control with Arbitrary Pole-placement for Industrial Robots and Manipulators," Int. Jour. Robotics Research, Vol. 1, No. 1, 1982.

[24] Goor, R. M., "A New Approach to Robot Control," Proc. Amer. Control Conf., June 1985.

[25] Goor, R. M., "A New Approach to Minimum Time Robot Control," in Robotics and Manufacturing Automation - PED - Vol. 15, Donath, M., Leu, M., eds., ASME, New York, 1985.

[26] Hanafusa, H., Asada, H., "Adaptive Control of a Robot Hand with Elastic Fingers for Mechanical Assembly," Third Symp. Theory and Practice of Robots and Manipulators, Udine, Italy, 1978.

[27] Hirzinger, G., "Robot Learning and Teach-In Based on Sensory Feedback," Third Symposium Robotics Research, Gouvieux, France, 1985.

[28] Hogan, N., "Impedance Control: An Approach to Manipulation," Jour. Dyn. Sys., Meas., and Control, March 1985.

[29] Hogan, N., Cotter, S. L., "Cartesian Impedance Control of a Nonlinear Manipulator," ASME Winter Meeting, Phoenix, 1982.

[30] Hollerbach, J. M., "A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity," IEEE Trans. Sys., Man, Cyber., Vol. SMC-10, No. 11, November 1980.

[31] Hollerbach, J. M., Suh, K. C., "Redundancy Resolution of Manipulators through Torque Optimization," IEEE Jour. Robotics & Automation, Vol. RA-3, No. 4, August 1987.

[32] Hsia, T. C., "Adaptive Control of Robot Manipulators - A Review," IEEE Conf. Robotics & Automation, San Fransisco, CA, April 1986.

[33] Kazerooni, H., Sheridan, T. B., Houpt, P. K., "Robust Compliant Motion for Manipulators," IEEE Jour. Robotics & Automation, Vol. RA-2, No. 2, June 1986.

[34] Khatib, O., "The Potential Field Approach and Operational Space Formulation in Robot Control," in Adaptive and Learning Systems: Theory and Application, Narendra, K. S., ed., Plenum Press, New York, 1986.

[35] Khatib, O., "A Unified Approach for Motion and Force Control of Robot Manipulators: The Operational Space Formulation," IEEE Jour. Robotics & Automation, Vol. RA-3, No. 1, February 1987.

[36] Klein, C. A., Huang, C.-H., "Review of Pseudoinverse Control for Use with Kinematically Redundant Manipulators," IEEE Trans. Sys., Man, Cyber., Vol. SMC-13, No. 3, March 1983.

[37] Koren, Y., Ulsoy, A. G., "Control of DC Servo-Motor Driven Robots," Robots VI Conf., Detroit, Michigan, March 1982.

[38] Luh, J. Y. S., "Design of Control Systems for Industrial Robots," in Handbook of Industrial Robotics, Nof, S. Y., ed., John Wiley & Sons, New York, 1985.

[39] Luh, J. Y. S., Fisher, W. D., Paul, R. P. C., "Joint Torque Control by a Direct Feedback for Industrial Robots," IEEE Trans. Auto. Control, Vol. AC-28, No. 2, February 1983.

[40] Luh, J. Y. S., Walker, M. W., Paul, R. P., "Resolved-Acceleration Control of Mechanical Manipulators," IEEE Trans. Auto. Control, Vol. AC-25, No. 3, June 1980.

[41] Maciejewski, A. A., Klein, C. A., "Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments," Int. Jour. Robotics Research, Vol. 4, No. 3, 1985.

[42] Mason, M. T., "Compliance and Force Control for Computer Controlled Manipulators," IEEE Trans. Sys., Man, Cyber., June 1981.

[43] MC68020 32-bit Microprocessor User's Manuel, Second Edition, Prentice-Hall, Englewood Cliffs, NJ, 1985.

[44] Miller, W. T. III, Glanz, F. H., Kraft, L. G. III, "Application of a General Learning Algorithm to the Control of Robotic Manipulators," Int. Jour. Robotics Research, Vol. 6, No. 2, 1987.

[45] Neuman, C. P., Khosla, P. K., "Identification of Robot Dynamics: An Application of Recursive Estimation," in Adaptive and Learning Systems: Theory and Application, Narendra, K. S., ed., Plenum Press, New York, 1986.

[46] Nigam, R., Lee, C. S. G., "A Multiprocessor-based Controller for the Control of Mechanical Manipulators," IEEE Jour. Robotics & Automation, Vol. RA-1, No. 4, December 1985.

[47] Paul, R. P., Robot Manipulators: Mathematics, Programming, and Control, MIT Press, Cambridge, MA, 1981.

[48] Paul, R., Shimano, B., "Compliance and Control," Proc. Joint Auto. Control Conf., 1976.

[49] Raibert, M. H., Craig, J. J., "Hybrid Position/Force Control of Manipulators," Jour. Dyn. Sys., Meas., and Control, Vol. 102, June 1981.

[50] Rosko, J. S., Digital Simulation of Physical Systems, Addison-Wesley, Reading, MA, 1972.

[51] Salisbury, J. K., "Active Stiffness Control of a Manipulator in Cartesian Coordinates," Proc. 19th IEEE Conf. Decision & Control, December 1980.

[52] Salisbury, J. K., Craig, J. J., "Articulated Hands: Force Control and Kinematic Issues," Int. Jour. Robotics Research, Vol. 1, No. 1, 1982.

[53] Shinners, S. M., Modern Control System Theory and Application, Addison-Wesley, Reading, MA, 1978.

[54] Skowronski, J. M., Control Dynamics of Robotic Manipulators, Academic Press, Orlando, 1986.

[55] Takegaki, M., Arimoto, S., "A New Feedback Method for Dynamic Control of Manipulators," Jour. Dyn. Sys., Meas., and Control, Vol. 102, June 1981.

[56] Vukobratovic, M., Stokic, D., "Is Dynamic Control Needed in Robotic Systems, and if So, to What Extent?" Int. Jour. Robotics Research, Vol. 2, No. 2, 1983.

[57] Vukobratovic, M., Stokic, D., Kircanski, M., "Contribution to Dynamic Control of Industrial Manipulators," Eleventh Int. Symp. Industrial Robots, Tokyo, October 1981.

[58] Whitney, D. E., "Force Feedback Control of Manipulator Fine Motions," Jour. Dyn. Sys., Meas., and Control, June 1977.

[59] Whitney, D. E., "Historical Perspective and State of the Art in Robot Force Control," IEEE Conf. Robotics & Automation, St. Louis, March 1985.

[60] Wilson, R., "New Coprocessors Head Toward Supermini Speeds," Computer Design, April 1987.

[61] Yoshikawa, T., "Manipulability and Redundancy Control of Robotic Mechanisms," IEEE Conf. Robotics & Automation, St. Louis, March 1985.

[62] Young, K.-K. D., "Controller Design for a Manipulator Using Theory of Variable Structure Systems," IEEE Trans. Sys., Man, Cybern., Vol. SMC-8, No. 2, February 1978.

INDEX

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NIST/TN-1255	2. Performing Organ. Report No.	3. Publication Date October 1988

4. TITLE AND SUBTITLE

Manipulator Servo Level Task Decomposition

5. AUTHOR(S)

John C. Fiala

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
(formerly NATIONAL BUREAU OF STANDARDS)
U.S. DEPARTMENT OF COMMERCE
GAITHERSBURG, MD 20899

7. Contract/Grant No.

8. Type of Report & Period Covered

Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

S/A

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

This document details the functionality of the Servo task decomposition modules for electric motor-powered manipulators with serial joints and unbranched kinematics. The treatment is strictly for rigid body links. In addition, the discussion is directed primarily toward autonomous mode operation. However, this document does describe the operator control interface at the Servo level. The document also describes the Servo interfaces to World Modeling, as well as the interfaces between the three task decomposition components: Job Assignment, Planning and Execution. The command interface from the Primitive task decomposition module to the Servo task decomposition module is specified, as is the interface to operator control.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

control; manipulator dynamics; nonlinear control; robot manipulators; servo control; shared control; task decomposition; teleoperation; telerobot control system

13. AVAILABILITY

[✓ XX] Unlimited
☐ For Official Distribution. Do Not Release to NTIS
☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
☐ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

41

15. Price

☆ U. S. GOVERNMENT PRINTING OFFICE: 1988--242-311--92592