NIST Technical Note 1256

Manipulator Primitive Level Task Decomposition

Albert J. Wavering

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 1256
54 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

1. Introduction 1

2. Overview of Prim Architecture 5

3. Prim Job Assignment Module Interfaces 8

3.1 E-move to Prim Job Assignment Interface 8
3.2 Prim Operator Control to Job Assignment Interface 14
3.3 Prim Job Assignment to World Modeling Interface 15

4. Prim Job Assignment Module Operation 15

5. Prim Planning Module Interfaces 15

5.1 Prim Job Assignment to Planning Interface 15
5.2 Prim Planning to World Modeling Interface 17

6. Prim Planning Module Operation 18

6.1 Joint Interpolated Motions 19
6.2 Linear Cartesian Space Motion 21
6.3 Optimal Trajectory Planning 24
6.4 Taking Advantage of Path Tolerance 26
6.5 Sensory-Interactive Trajectory Generation 27
6.6 Biologically-Motivated Trajectory Techniques 30
6.7 Trajectory Planning in a Non-Stationary Environment 30
6.8 Generalized Damper Motion 31
6.9 Hybrid Force/Position Motion 32
6.10 Kinematic Redundancy 34

7. Prim Execution Module Interfaces 34

7.1 Prim Planning to Execution Interface 34
7.2 Prim Execution to World Modeling Interface 37
7.3 Prim Execution to Servo Task Decomposition Interface 38

8. Prim Execution Module Operation 39

8.1 Executing Completely-Planned Trajectories 40
8.2 Executing Sensory-Interactive Trajectories 40
8.3 Negotiating Singular Configurations 41
8.4 Kinematic Redundancy 42
8.5 Reflexive Obstacle Avoidance 42
8.6 Operator Velocity Control 43
8.7 Operator Single-Step Control 43

9. Implementation Issues 43

9.1 Interface Size and Information Flow Rates 43
9.2 Processing Requirements 44

10. **Conclusion** 46

11. **References** 46

Manipulator Primitive Level Task Decomposition

1. Introduction

This document describes the structure, function, and interfaces of a trajectory generation module in a hierarchical manipulator control system. The overall framework of such a control system is described in [2], and it will be assumed that the reader is familiar with this reference. Figure 1 provides a pictorial representation of the initial concept of the control system hierarchy. The module described in this document is part of the *task decomposition* hierarchy, which subdivides high-level tasks into simpler and simpler subtasks. The decomposition performed by this module is to generate a time sequence of closely-spaced manipulator goal states from a static description of a desired motion. As such, it generates *primitive* trajectories, and is called the Primitive (Prim) task decomposition module. The relationship of this module to other elements of the control system is shown in figure 2. Figure 3 highlights the elements of the control system discussed in this document.

As shown in the first three figures, Prim receives commands from either the Elemental Move (E-move) level of the task decomposition hierarchy, which performs such functions as grasp planning and planning of gross and fine motions, or from the Operator Control. The output commands of the E-move level are time-independent descriptions of motions, for example static position or position and force *paths*, or directional *fields*. In the first case, the position and/or force commands are in the form of parameterized paths to be followed. In the case of a directional field description, the command takes the form of position-dependent fields which indicate the desired direction of motion or force application. In addition to these types of commands, E-move can also simply specify a set of *termination conditions,* or goal states, along with an *algorithm* specification which determines the strategy to be used to achieve them. This type of command is useful for sensory-interactive algorithms such as Cartesian end point servoing with vision. Motion commands from E-move may be large, as with gross motion paths specified for free space motions, or relatively small, as in fine motion segments for an assembly task.

Prim generates the time sequence of *attractor sets* needed to produce a *dynamic trajectory* from the E-move or Operator Control command, and sends these as commands to the Servo level of the task decomposition hierarchy. The Servo level, the lowest level in the hierarchy, controls the behavior of the manipulator in performing small motions between closely-spaced goals, or attractor sets. The function and interfaces of a Servo level for manipulators which accommodates a broad spectrum of known control algorithms is described in [20]. In addition to determining position, velocity, acceleration, and/or force trajectories to be commanded to Servo, Prim also has the task of determining appropriate manipulator *impedance*, stiffness, damping, and inertial characteristics which are controlled by adjusting the servo loop gains commanded to Servo.

The remainder of the document is arranged as follows. First, an overview of the architecture of the Prim level will be presented. As described in [2], the Prim task decomposition

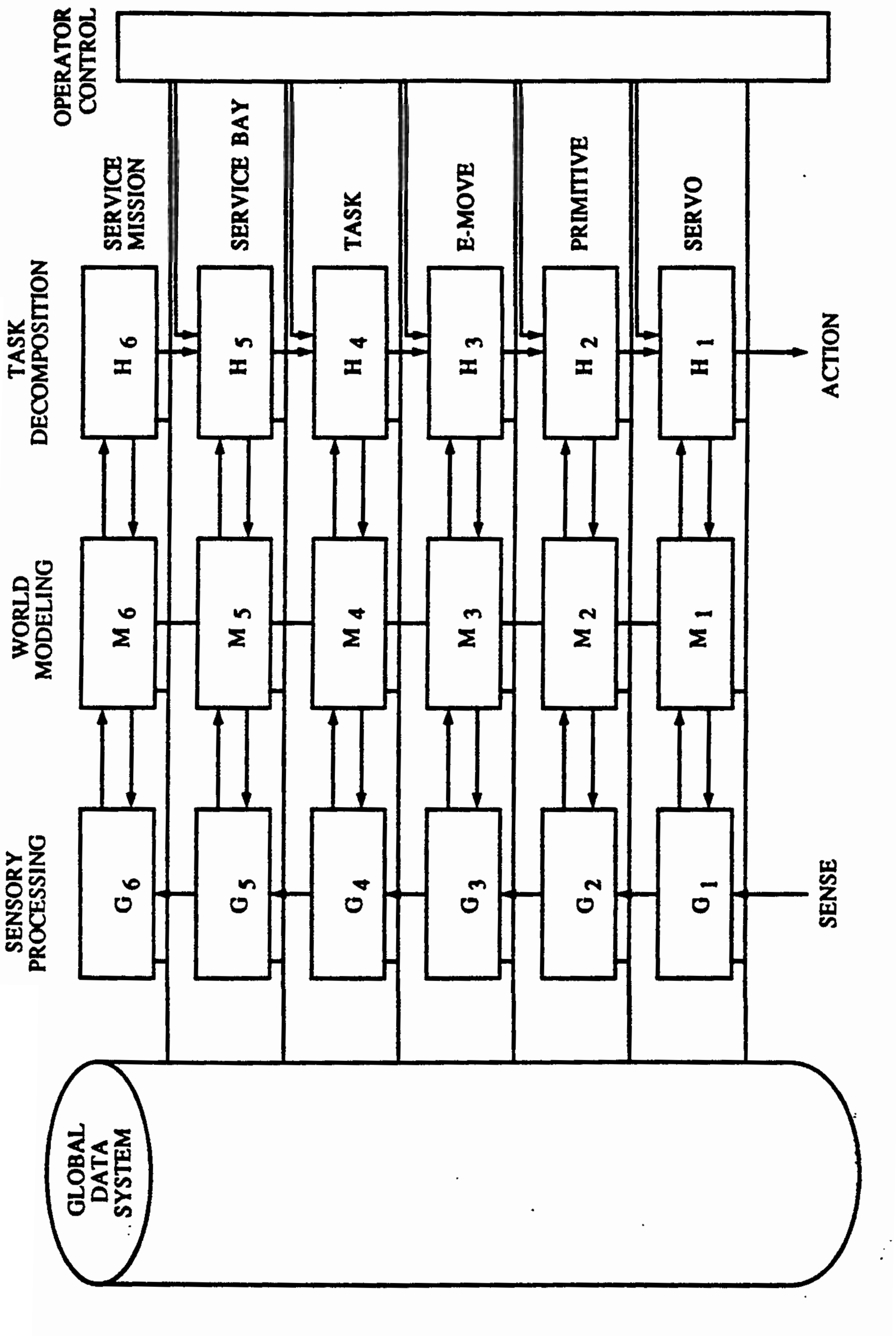

Figure 1. A hierarchical telerobot control system architecture.

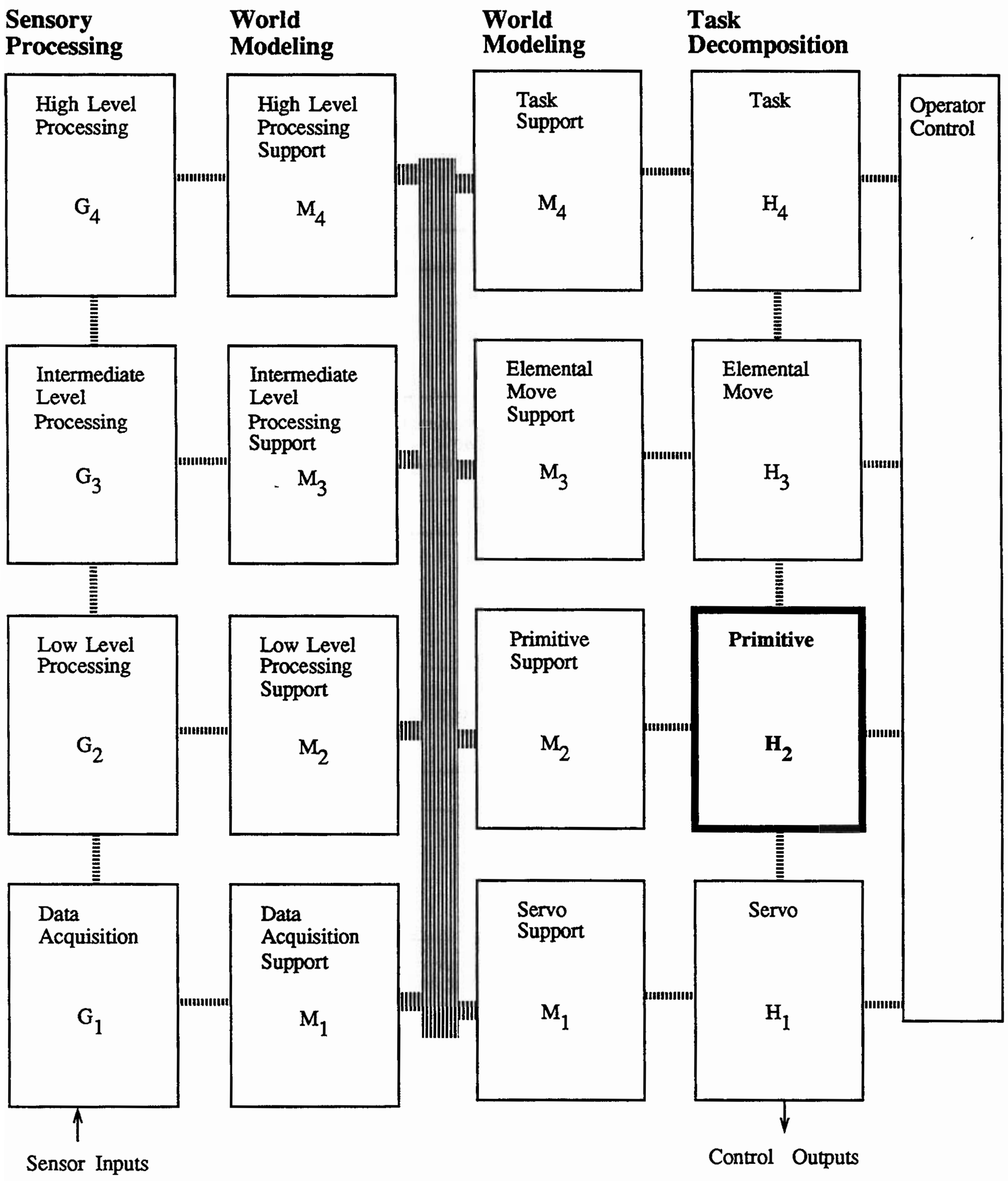

Figure 2. Relationship of primitive task decomposition module to other components.

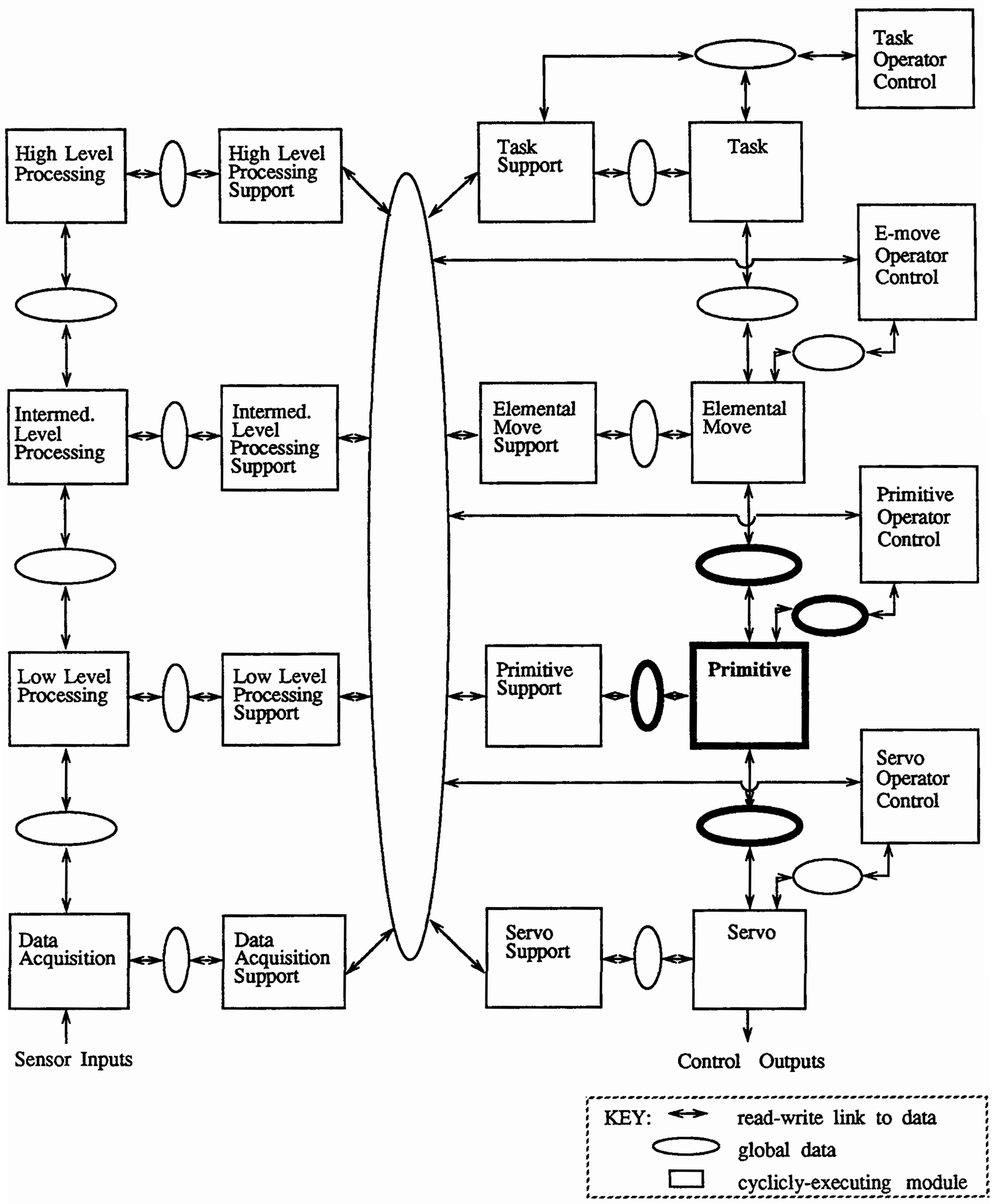

Figure 3. Primitive level task decomposition module and system interconnections.

module consists of *Job Assignment*, *Planning*, and *Execution* submodules. Following the discussion of the general operation of these components, each will be described in greater detail. First, the Job Assignment module will be discussed by identifying the interfaces (sec. 3) and describing the operation (sec. 4). Section 5 discusses the interfaces of the Planning module (sometimes called the Planner), while section 6 describes the operation of this module, including examples of trajectory generation for several types of algorithms. Following the same pattern, the Execution interfaces and operation are discussed in section 7 and section 8. Finally, some implications of the module design in terms of implementation are discussed in section 9. This document does not discuss the relative merit of various representations which may be used for the information in the interfaces. Also, the document does not directly address the issues involved with end-effector control or control configurations for dual-arm cooperation, although the proposed interfaces should be useful for these applications. Throughout the document, *position* refers to *position* and *orientation*, and *force* refers to *force* and *torque* unless it is clear from the discussion that a more restricted interpretation is implied.

2. Overview of Prim Architecture

As illustrated in figure 4, the Prim task decomposition module is composed of three subcomponents; the Job Assignment module, the Planning module, and the Execution module. These correspond to the processes JA(2), PL(2,s), and EX(2,s) (s=1,...,n, where n is the dimensionality of the trajectory generation problem), of [2]. The input commands to Prim are queued so that future commands will be available to the Planning module, and the Job Assignment module manages this command queue. The Prim Planning module handles the generation of a plan for a dynamic trajectory (for example, time functions of manipulator position, velocity, acceleration, and/or force). Plans may also take the form of trajectory parameters which specify the shape or characteristics of a trajectory, without indicating the exact path to be followed, as in the case of sensory-interactive trajectories. This planning is performed according to the algorithm specified in the command specification. The trajectory functions are then evaluated by the Execution module, which computes outputs to the Servo level at time intervals specified by the Planning module. Each of the Job Assignment, Planning, and Execution modules is a *cyclically-executing* process, in that each continually reads inputs, performs computation, and then writes output in a cyclic manner. The advantages to this type of execution and communication are discussed in [20].

Information about the state of the manipulator and the world is needed to perform planning and execution tasks. This information is provided to the task decomposition module via the world modeling support module shown in figures 2, 3, and 4. This module accesses data stored in the *global data system* by the sensory processing side of the control hierarchy (figs. 2 and 3). The Prim world modeling support module may access information which has entered the data system through any level of the sensory processing hierarchy. This is indicated by the vertical data path between the two columns of world modeling support modules in figures 2 and 3. Note that this implies there is no direct correspondence between sensory processing levels and task decomposition levels. The generic term "world model" will often be used in this document to refer to the Prim world model support module and the global data system collectively.

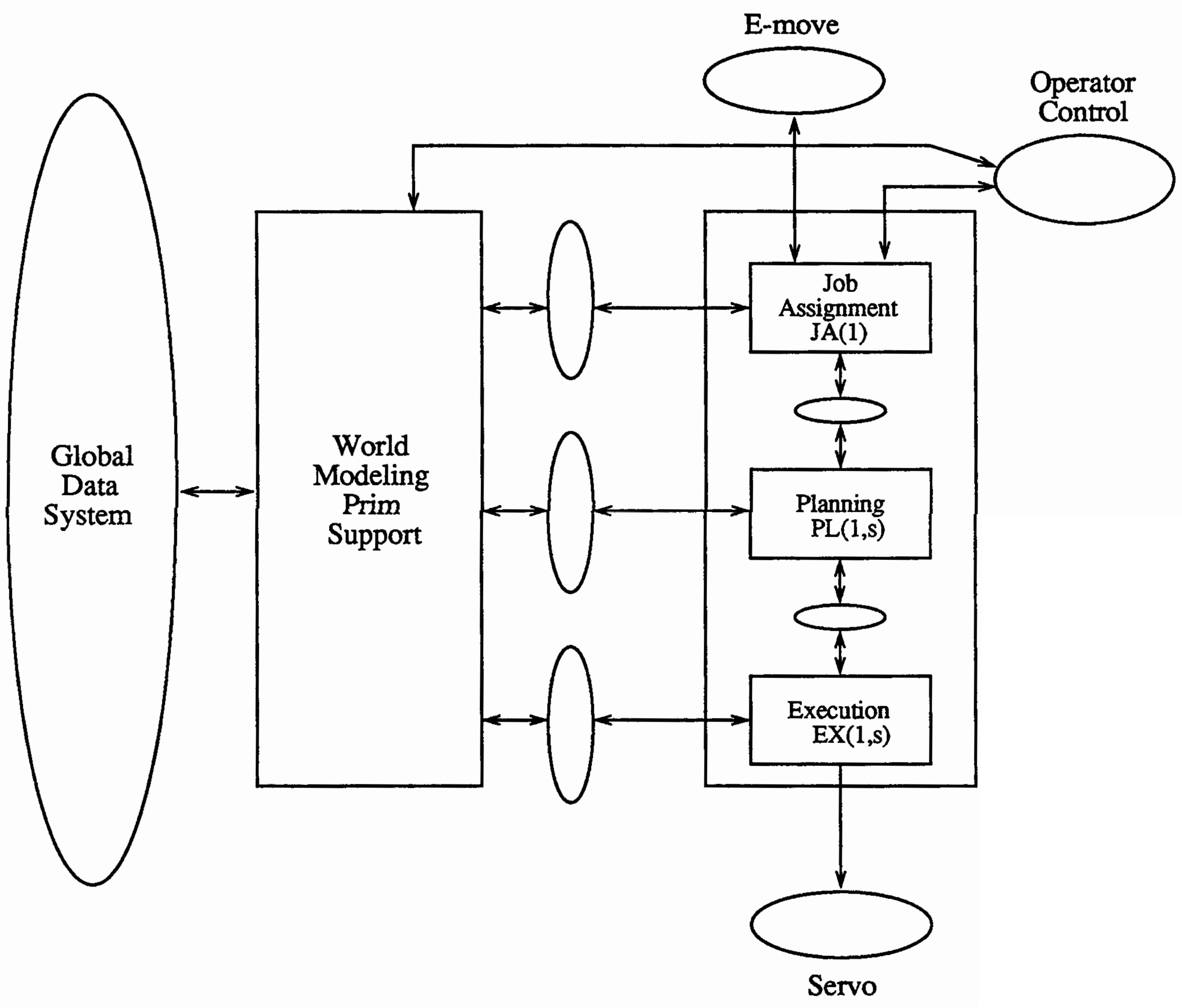

Figure 4. Prim task decomposition structure.

Also shown in the figures is an operator control module. The operator can enter commands into the command queue, control the velocity of the manipulator as it executes a planned trajectory, and single-step the operation of the Execution module. The operator always has ultimate control over the contents of the queue of input commands. The operator also has access to status information which is pertinent to controlling the system at the Prim level.

Figure 5 again shows the three components of the task decomposition module, with the addition of a detailed list of the types of information which may flow across the Prim task decomposition module boundaries. This information and the interfaces between the module subcomponents will be discussed further in the sections that follow.

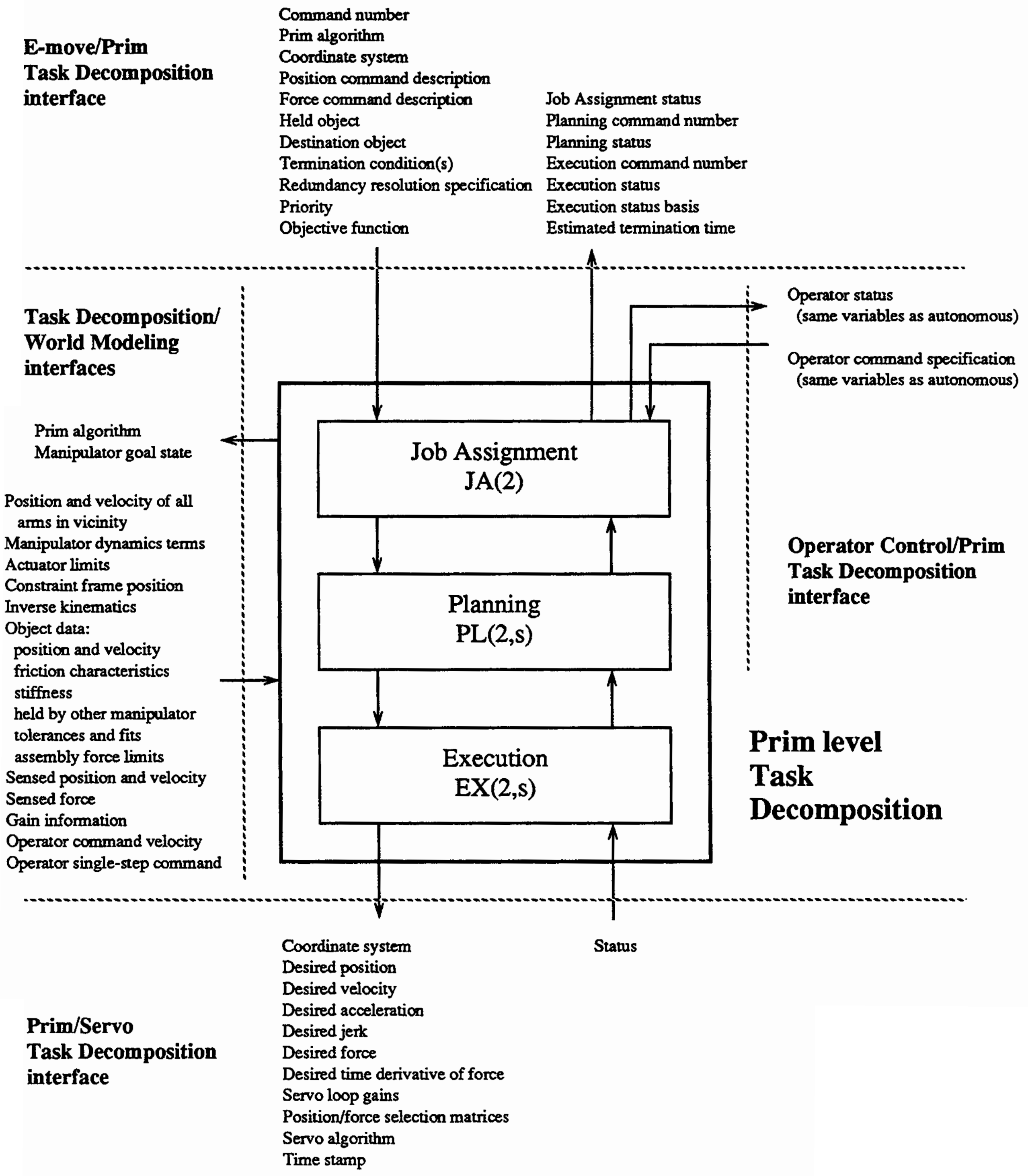

Figure 5. Prim task decomposition external interfaces.

3. Prim Job Assignment Module Interfaces

The Prim Job Assignment module receives commands from the Task Decomposition module of the E-move level and from the Operator Control. Figure 6 shows the information which passes in and out of the Job Assignment module. The data which comprise the interfaces between these units will be described in the following sections. The Job Assignment module takes the commands and places them in a command queue, which is accessible to the Planning module. The input command queue and operation of the Job Assignment module are discussed in section 4.

3.1 E-move to Prim Job Assignment Interface

The nature of Prim input commands from E-move is indicated in figure 6. The command specification consists of an algorithm and a set of parameters. The parameters which have been included represent commonly-used means of describing manipulator motions in a time-independent manner, and expressing what factors are important in transforming the command into a dynamic movement. It is important to recognize, therefore, that the particular combination of parameters for a given command will depend on the nature of the algorithm. Also, in many cases a particular command will only be a segment of a longer trajectory, where different algorithms may be used for different segments. After the command interface has been described in the following, a table will be presented which shows typical combinations of parameters for different types of commands. Examples of input command specifications and more details on how they are used are also presented in section 6, Prim Planning Operation.

The command interface from the E-move Task Decomposition module to the Prim Job Assignment module consists of the following information; the symbol (if any) which may be used to refer to a given element is also given:

Command number

The command number identifies the current command. Since Prim can accept a number of commands to be working on, it needs a way to associate the status it provides to E-move with a particular command.

Prim algorithm

The Prim algorithm specifies the algorithm to be used to generate a trajectory. The algorithm determines the *type* of manipulator behavior desired in performing a command. It is up to Prim to determine specifics of the behavior, such as velocity, acceleration, stiffness, and the like. Several example algorithms are discussed in section 6.

Coordinate system, C_z

The coordinate system indicates what frame of reference is to be used for motion descriptions. The format of C_z is (coordinate system, T_w, T_e), where T_w represents a transformation from the robot base to the desired reference coordinate system and T_e represents a transformation from end-effector coordinates to the frame which is desired to follow the specified motion (see fig. 7). The possibilities for C_z are:

(joint, -, -)	=> path description in joint space

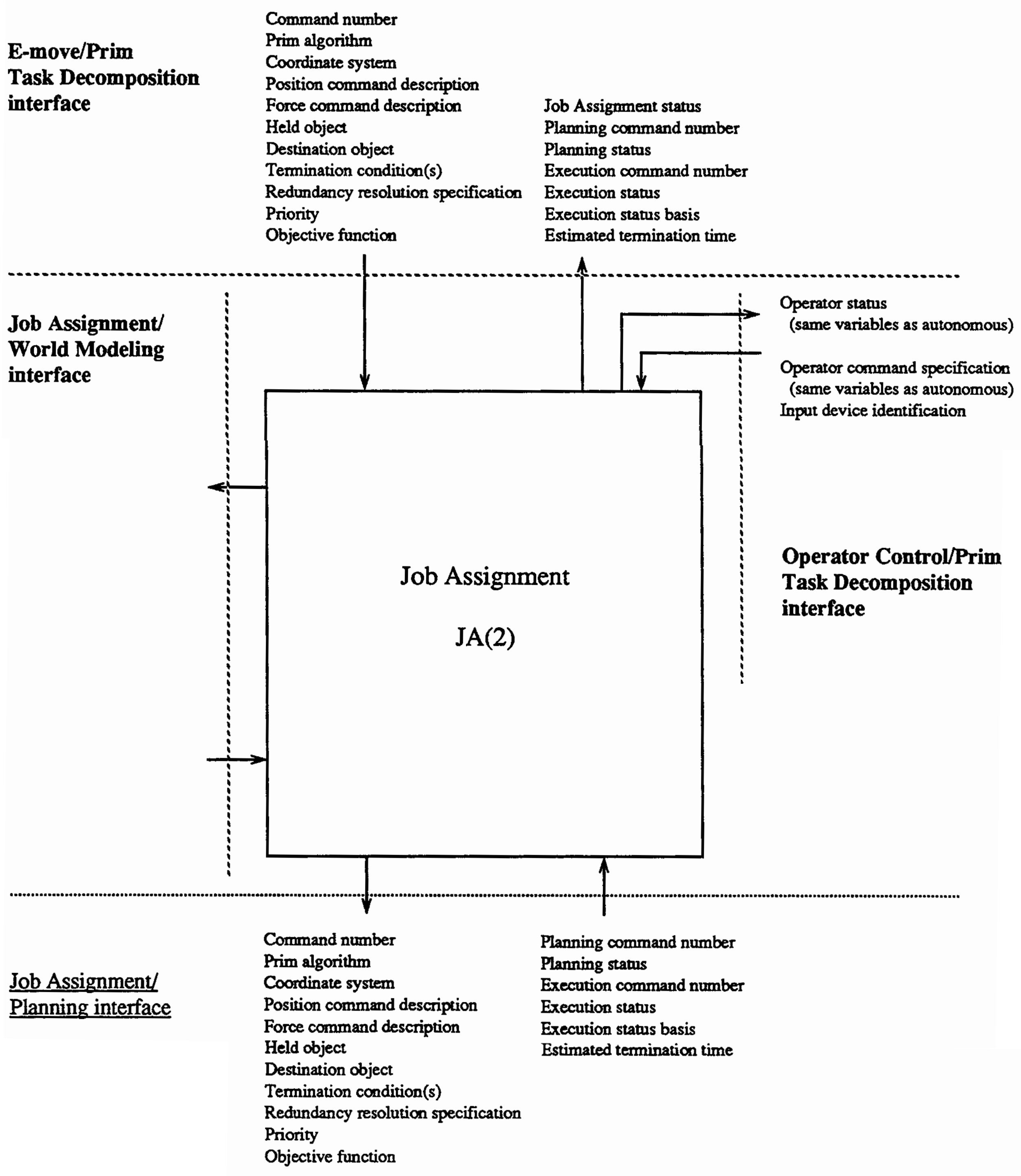
E-move/Prim Task Decomposition interface
Command number
Prim algorithm
Coordinate system
Position command description
Force command description
Held object
Destination object
Termination condition(s)
Redundancy resolution specification
Priority
Objective function
Job Assignment status
Planning command number
Planning status
Execution command number
Execution status
Execution status basis
Estimated termination time
Job Assignment/ World Modeling interface
Operator status (same variables as autonomous)
Operator command specification (same variables as autonomous)
Input device identification
Job Assignment
JA(2)
Operator Control/Prim Task Decomposition interface
Job Assignment/ Planning interface
Command number
Prim algorithm
Coordinate system
Position command description
Force command description
Held object
Destination object
Termination condition(s)
Redundancy resolution specification
Priority
Objective function
Planning command number
Planning status
Execution command number
Execution status
Execution status basis
Estimated termination time

Figure 6. Prim job assignment module interfaces.

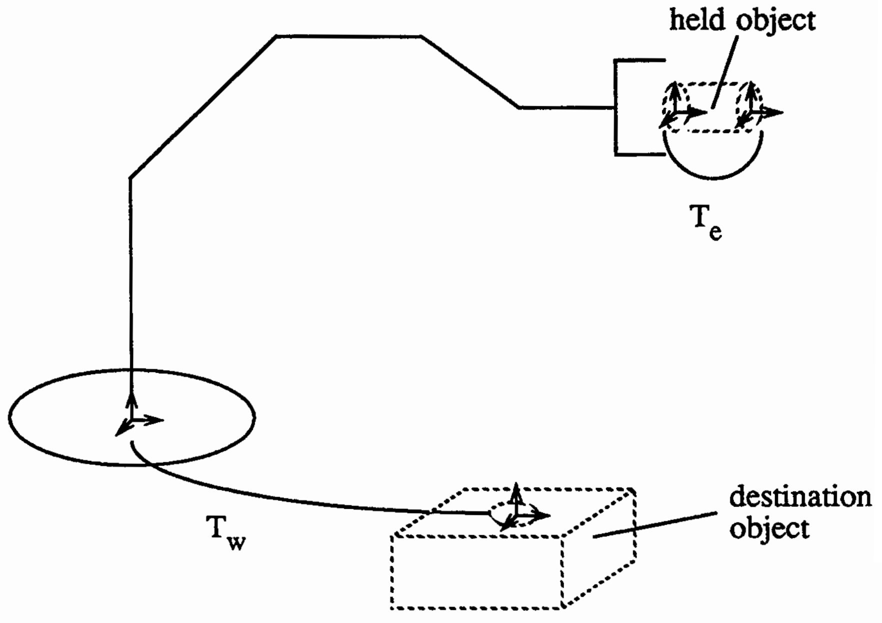

Figure 7. Coordinate frames for C_z.

(end-effector, -, T_e)	=> motion direction or force in Cartesian system fixed with respect to the end-effector
(world, T_w, T_e)	=> path description or other motion goal for Cartesian system fixed with respect to the end-effector, in terms of Cartesian system related to the manipulator base by T_w

The transformations T_w and T_e may be fixed, rigid-body, homogenous coordinate transformations. Alternatively, the transformations may be defined symbolically by specifying the name of an object, in which case the exact transform may be time varying. Object names (which may designate specific features of objects), may be used to periodically obtain either absolute or relative position information about the objects from the world model, depending on the trajectory algorithm.

Position command description, P

Force command description, F

The position and force command descriptions are used when it is desired to command fairly specifically *how* Prim is to move to achieve the goal conditions (given by the termination conditions, below). The format of each of the command descriptions P and F is (vector function, tolerance function). The vector functions may represent two different types of commands. In one case, the commands represent position and force paths parameterized in the variable s, which represents the fraction of the path traversed. In this case, the force path specifies the desired force at each point along the commanded position path through space. The other possibility is to have the functions represent position-dependent fields of desired position and force directions. The type of function (path or field) is indicated in the representation of the function. The two possibilities will now be discussed in more detail.

If the functions represent paths, they must have continuous first derivatives and the common path parameter s must increase monotonically. If it is desired to command a position path composed of non-smooth segments (for example, a series of straight lines), these segments may be commanded as consecutive commands. The position path description does not represent an exact position goal for the manipulator; it must be considered along with the position path tolerance. The position tolerance defines a sphere at each point along the path within which the trajectory generated by Prim must fall. For Cartesian paths, the position tolerance consists of two separate components; one for translation and one for rotation. For joint space motions, there is only one component, which gives the allowable distance from the desired path in terms of the Euclidean norm of the individual joint deviations. Together, the position path description and tolerance define a generalized cylinder [12], which Prim is free to plan a trajectory anywhere within. Similarly, there is a tolerance function associated with the commanded force path. Note that a path description cannot be used if the end-effector coordinate system is selected, as the reference coordinate system would move as the path was traversed.

If the command descriptions specify position and force directional fields, they must again have continuous first partial derivatives. This type of motion specification may be used when it is desired to command the manipulator to move in a particular direction (which may depend on the position relative to some object). Such a specification would be used with generalized damper control [57], for instance, where the manipulator is commanded to move in some direction, and slides along the surface of an object when contact is made. Another example would be resolved motion rate control. In either case, the Prim Planning module must determine an appropriate nominal velocity to command in the specified direction. The directional fields also have associated tolerances, which may also be position-dependent. The tolerance functions in this case specify the amount of allowable deviation between the direction of force and motion specified by E-move and the actual direction of the force and velocity commanded by Prim to Servo.

Examples of both types of command descriptions may be found in section 6. The reader should keep in mind that it is not always necessary to use the position and force command descriptions; they are used when it is desired to command motion along a specific path or in a particular direction to achieve the goal indicated in the termination condition.

Held object, HO

Destination object, DO

The held object and destination object specifications indicate what object is currently held by the end-effector and what object, if any, the held object will contact during the commanded motion. This information is included in the input command so that Prim may easily access information about the contacting objects, such as stiffness, friction, and inertial characteristics, from the world model. These characteristics are needed by Prim to determine appropriate values for stiffnesses, damping coefficients, and velocities to be used during contact operations. The object names include the names of the specific features which will be contacting. If the Prim Planning is not smart enough to figure out reasonable gains and velocities given the object characteristics, it should at least be able to look up appropriate values to use given the names of the contacting features.

Termination condition(s), TC

The termination conditions are the desired goal states which should be achieved by the end of command execution. They specify the criteria to be used to evaluate when a Prim command is "done". The termination conditions may be given in terms of desired values of sensed forces, positions, velocities, elapsed time, or desired states of other world model variables. The termination conditions therefore provide a means of implementing the *guarded moves* used in many manipulation strategies [4,17,37,59].

Redundancy resolution specification, R

The redundancy resolution specification indicates what technique should be used to convert a 6 degree of freedom (dof) Cartesian input command into a 7 or more dof output command.

Priority

The priority specification indicates which manipulator is to have priority if there is a conflict for path space. The Prim level may use this information to decide whether to alter the current trajectory if collision with another manipulator is imminent [18,21]. Since the priorities for individual manipulators depend on the task and must be set with all manipulators in mind, they should be set by the E-move level.

Objective function, OF

The objective function is used to communicate what factors are important in generating a dynamic trajectory from the static motion description. For example, it may be important that time or expended energy be minimized for a particular motion.

As mentioned previously, different combinations of parameters are used for different types of input commands. Table 1 lists typical sets of input command parameters that might be used with some different types of algorithms. Note that these parameter combinations are only examples; particular algorithms within a group may use somewhat different combinations.

The Job Assignment Module returns the following status information to the E-move Task Decomposition module:

Job Assignment status

The Job Assignment status lets E-move know whether or not Prim has operator commands in the queue. The job assignment status may take at least the following values:

accepting autonomous	=> the Job Assignment module can accept new commands from E-move; no operator commands are in the command

Table 1. Parameter sets for some types of input commands.

	Free-space	Hybrid force/pos.	Gen. damper	Sens. Interactive
Command number	x	x	x	x
Prim algorithm	x	x	x	x
Coordinate system	x	x	x	x
Position cmnd. descr.	x	x	x	-
Force cmnd. descr.	-	x	-	-
Held object	x	x	x	x
Destination object	-	x	x	x
Termination cond.	x	x	x	x
Red. resol. spec.	x	x	x	x
Priority	x	x	x	x
Objective function	x	x	x	x

queue

accepting operator	=>	the Job Assignment module has commands from the operator in the queue and will ignore any commands sent from E-move

Planning command number

The Planning command number indicates which command is currently being processed by the Planning module.

Planning status

The Planning status element indicates the current status of the Prim Planning module. The Planning status may indicate the following conditions:

ready for next command	=>	the Planner is ready to receive the next command specification
need next command	=>	the Planner needs to have a command added to the queue in order to plan the trajectory for the next command scheduled for execution. If a command is not added to the queue then the manipulator will have to stop.

Execution command number

The Execution command number indicates which command is currently being processed by the Execution module.

Execution status

The Execution status indicates the status of the command currently being processed by the Execution module. The execution status may have the following values:

executing	=>	the command currently being processed is executing
done	=>	the command currently being processed has finished
error	=>	an error has occured in executing the current command

Execution status basis

The Execution status basis indicates more fully the meaning of the execution status. For example, if the status of the current command is done, the Execution status basis indicates what termination conditions were met (i.e. was position goal achieved, or was maximum time reached). If there is an error status, it may indicate the nature of the error.

Estimated termination time

The estimated termination time provides E-move with Prim's best estimate of how long it will take to complete the current command. This is not practical to do with all algorithms, but for those that it is, the estimated termination time should be a useful piece of information for E-move scheduling. A lack of an estimate of the termination time may be indicated by setting this parameter to 0.

3.2 Prim Operator Control to Job Assignment Interface

In order to support teleoperation mode where commands are entered by an operator rather than the level above, there is an interface between the Prim Job Assignment Module and the Operator Control, through which operator interactions take place. Complete Prim input commands may be submitted to the Job Assignment module through the Operator Control. Another important aspect of teleoperation or manual modification of autonomous operation at the Prim level is the capability for an operator to manually override the planned velocity for an otherwise autonomous trajectory. The operator may do this by operating a joystick potentiometer or similar device. Of course, the particular device must have been previously identified. The information which passes from the Operator Control to the Prim Job Assignment module to provide these capabilities consists of the following:

Operator command specification

The operator command specification consists of an algorithm and complete set of parameters specified by the operator. The components of the operator command specification are of the same type as those supplied by E-move, as discussed above.

The Job Assignment module returns the following information to the Operator Control:

Operator status

The operator status includes the same types of information as provided to E-move.

3.3 Prim Job Assignment to World Modeling Interface

At this time it appears that the Job Assignment module need not share an interface with the world model. All the information the Job Assignment module needs to perform its functions are provided by either the E-move Execution module or the Operator Control.

4. Prim Job Assignment Module Operation

The primary function of the Prim Job Assignment module is to manage the queue of input commands. An input command queue is necessary because the Prim Planning module needs information about future commands in order to plan smooth transitions between trajectory segments. During autonomous operation, the Job Assignment module accepts commands from E-move and places them in the queue as they are sent down. The Job Assignment module erases a command from the queue after it has finished executing and shifts the other commands forward. When the operator wishes to take control at the Prim level, he or she may do so by editing the queue to insert and delete commands. There are a number of ways in which the operator might be allowed to edit the command queue, but it seems reasonable to require that whenever the operator enters a command in the queue, the Job Assignment module will empty any autonomous commands scheduled to be performed after the operator command(s). In addition, the Job Assignment module should let E-move know (via the Job Assignment status) that there are operator commands in the queue and that any commands sent from E-move will be ignored, until the operator commands have finished executing. This approach seems reasonable since the task should be replanned from the top down after operator intervention, as conditions may have changed which will affect task execution. Since the operator is allowed to edit the queue, it would be convenient to have a mechanism associated with the queue (e.g. a counter for each queue position) that allows the Planning module to determine when a change has occurred in a planned trajectory or one that is being planned.

5. Prim Planning Module Interfaces

The interfaces between the Prim Planning module, the Job Assignment module, the world model, and the Execution module are shown in figure 8. The first two of these interfaces are described in this section. The interface to the Execution module is discussed in section 6.

5.1 Prim Job Assignment to Planning Interface

The input to the Planning module consists of the commands in the Job Assignment queue, which have come from either E-move or the Operator Control. The format of the commands does not change when they are entered in the queue; therefore the interface to the Planning module consists of the same information as contained in the Job Assignment input command specification, described in section 3.1.

The Planning module returns the following to the Job Assignment module:

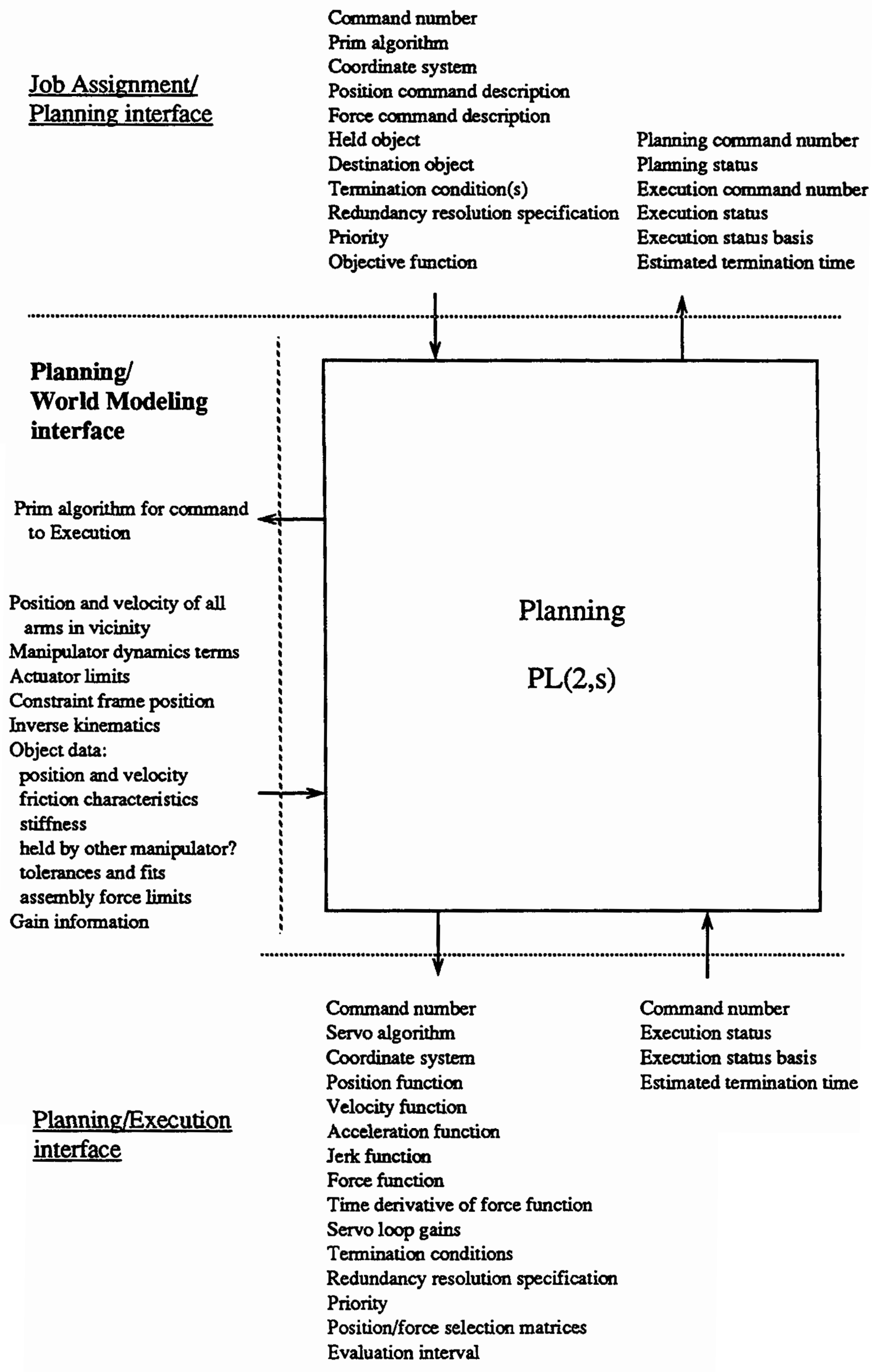

Figure 8. Prim planning module interfaces.

Planning command number

Planning status

Execution command number

Execution status

Execution status basis

Estimated termination time

All of these command and status elements correspond to the like-named items in the E-move to Prim Job Assignment interface. The execution-related parameters are passed straight through from the Execution module to the Job Assignment module.

5.2 Prim Planning to World Modeling Interface

In order to plan reasonable and effective trajectories the Prim Planner requires a substantial amount of information from the world model. The Prim Planner must be able to obtain the following information from the world model (as a minimum):

Inverse kinematics

Given the manipulator position and velocity in one coordinate system, the world model must be able to provide manipulator position and velocity in the desired coordinate system. The world model can also transform forces from one coordinate system to another.

Actuator limits

The world model must be able to provide actuator torque and acceleration limits as a function of joint position and speed. Joint position limits must also be available.

Manipulator dynamics terms

World modeling must also have the capability of providing manipulator dynamics terms, including inertial, Coriolis, friction, and gravity terms; given the manipulator position and velocity.

Object data

A variety of information relating to objects must be available from the world model. This includes stiffness, friction, mass, and tolerance information, in addition to geometry data. The position and velocity of objects may also be useful for trajectory planning if the object motion is predictable. The amount and type of data required by the Prim Planning module depends greatly on the reasoning capability of the Planner. For example, the Planning module may determine appropriate servo gains for an assembly operation based on the object characteristics mentioned above, or it may simply use the names of the objects (and, more specifically, of the mating features), as keys to retrieve pre-stored parameters to be used for the operation.

Gain information

Information about appropriate gains to use for different types of motion is available from the world model. The Planning module may obtain this information and specify gains when fixed gains are to be used for an entire motion segment.

The Prim Planning module provides the world model with the Prim algorithm used for the command sent to Execution. This is included to assist the sensory processing and the world model in selecting appropriate sensors for use in sensory-interactive trajectories.

6. Prim Planning Module Operation

This section discusses a number of ways in which Prim can plan a trajectory from the motion specification supplied from E-move and/or the Operator Interface. The nature of the planning performed depends to some extent on the type of motion commanded. For instance, the planning for a free space motion using a simple trajectory generation algorithm may be quite different from that for more complex sensory-interactive motions. This section discusses planning for several trajectory generation algorithms.

For preplanned free space motions, the Planning module must determine manipulator position, velocity, acceleration, and/or jerk as a function of time to satisfy the commanded path constraints. In doing so, it must in some way take into consideration the dynamics of the task, and the limitations of the joint actuators. Joint actuators have limitations on maximum speed, and maximum force or torque output. The torque limitation places configuration-dependent bounds on the maximum acceleration of a joint. In addition, joint travel limits must not be exceeded. One approach is to use worst-case bounds on joint accelerations to generate a trajectory. This is the tack taken by the algorithms discussed in section 6.1 and section 6.2. This approach is straightforward and simple since the manipulator dynamics need not be computed. However, it does not allow maximum utilization of the actuator capabilities to obtain minimum-time trajectories, nor does it enable the generation of the most energy-efficient trajectories. To obtain maximum manipulator performance, the complete dynamics of the manipulator and payload, and the joint actuator limits must be taken into account simultaneously in determining a trajectory. Section 6.3 and the second part of section 6.4 discuss algorithms that incorporate manipulator dynamics in trajectory planning.

It is important to realize that the terms "trajectory planning" and "trajectory generation algorithm" as used here do not always refer to methods of preplanning the exact position, velocity, and acceleration of the manipulator at every instant during the motion. Instead, the Planning module may only determine functions or parameters which determine what the general profile should look like for the motion, and the exact path the robot takes during execution is determined by sensory or other external inputs. Such is the case for certain sensory-interactive trajectory generation algorithms, as discussed in section 6.5. Another way of incorporating sensory interaction is to plan an explicit trajectory based on sensory inputs available at the time of planning, but then replan the motion frequently during execution to incorporate new sensor information [2,5]. A replanning interval of 50–150 msec appears to be appropriate for such algorithms [2,5].

For planning all types of motions, the Prim Planning module looks ahead by an amount of time which depends on several factors, including the length of the motion itself and the nature of the objective function to be optimized during the motion (i.e. minimum-time vs. minimum energy). The planning horizon is not fixed, therefore; but it will typically be on the order of 1 to a few seconds.

In addition to planning a suitable trajectory, the Planning module must also select an appropriate servo algorithm and servo loop gains. In most cases, the Prim algorithm will imply the servo algorithm to be used. Otherwise, Prim can choose an appropriate algorithm based on heuristics. The position, force, and torque servo loop gains determine the behavior of the manipulator in response to new position and force goals, and to external disturbances. The Prim Planning module may determine the apparent manipulator impedance by adjusting these gains. A constant set of gains that works reasonably well for a variety of manipulator configurations and payloads might be used for free space moves. Such gains result in compromised performance, however, since the manipulator dynamics are configuration-dependent. Alternatively, gain scheduling may be used to vary the gains as a function of the manipulator state during trajectory execution [3, 41]. In this case, on-line gain modification is performed by the Execution module. Time-varying gains may also be used to generate trajectories with a constant position [13], as discussed briefly in section 6.6. For constrained motions, the gains might be set using heuristics or by optimizing an objective function specified from E-move; to minimize or maximize work done on the environment, for instance [23]. The Planning module also determines the intervals of time for which the position and force trajectory functions should be evaluated.

In order to plan smooth transitions between consecutive path segments, the Planning module must have some knowledge of future commands. The Prim Planning module has a command queue to buffer a number of future input commands. For many algorithms which have been suggested for splining together manipulator motion segments, the current goal point plus the next two future points are sufficient [14,30,43,55]. When the Planning module needs an additional future command in order to plan the current one, it sets the planning status to "need next command".

6.1 Joint Interpolated Motions

It is often useful to command motions in joint space. Linear joint space motions have been shown to be close to time-optimal in certain situations where the manipulator dynamics are dominant [48], and the problem of excessive joint velocities at singular configurations is often avoided (depending on the length of the path in Cartesian space). In addition, at least one path planning algorithm is performed in joint space [33], in which case joint space would be a natural choice for the command to Prim. A typical command specification to Prim for a linear joint space motion with a cubic polynomial trajectory function is given below. For clarity, the path specification $z_d(s)$ is represented by $\theta_d(s)$ when C_z = (joint, -, -).

Command number	=	n	
Prim algorithm	=	cubic polynomial	
Coordinate system	=	(joint, -, -)	<= joint space
Pos. cmnd. descr.	=	$(\theta_d(s) = \theta_0 + s(\theta_1 - \theta_0), e_{01})$	<= straight line, constant tol.
Force cmnd. descr.	=	null	
Held object	=	null	
Destination object	=	null	

Termination cond.	= θ_f within e_{01} of θ_1, $\dot{\theta}_f = 0$	<= at goal position, zero velocity
Red. resol. spec.	= default	
Priority	= 1	
Objective function	= minimize $OF = (T_{seg} - T_{segd})^2$	<= minimize difference between desired and actual traversal times

Such a path results in all joints reaching the final position at the same time. E-move must of course check that the joint space path is free of obstacles—not a trivial task in a cluttered environment.

Once a parameterized path has been specified, Prim must generate a dynamic trajectory according to the specified algorithm. One method of generating joint positions as a function of time for linear joint space motions is presented by Taylor [55]. With this algorithm specified, the motion from P_0 to P_1 is given by

$$\theta(t) = \theta_1 - \frac{T_1 - t}{T_1} (\theta_1 - \theta_0)$$

and the transition between the path segment from P_0 to P_1 and the segment from P_1 to P_2 is calculated using

$$\theta(t) = \theta_1 - \frac{(t_{acc} - t')^2}{4\, t_{acc} T_1} (\theta_1 - \theta_0) + \frac{(t_{acc} + t')^2}{4 t_{acc} T_2} (\theta_2 - \theta_1)$$

where $t' = T_1 - t$

t_{acc} = half of the transition time

T_1 = traversal time for the segment from P_0 to P_1

T_2 = traversal time for the segment from P_1 to P_2.

In this case, a constant acceleration is used to accelerate the manipulator from the first segment velocity to the second segment velocity, resulting in a parabolic transition path. Suitable segment traversal and transition times must be determined by the Prim Planning module.

There are other possibilities for joint interpolated motions, of course. For example, for a single-segment cubic polynomial joint space path the trajectory functions for each joint may be determined as follows [15]:

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

$$\dot{\theta}(t) = a_1 + 2a_2 t + 3a_3 t$$

$$\ddot{\theta}(t) = 2a_2 + 6a_3 t$$

where $a_0 = \theta_0$

$a_1 = \dot{\theta}_0$

$a_2 = \frac{3}{T^2}(\theta_1 - \theta_0) - \frac{2\dot{\theta}_0}{T} - \frac{\dot{\theta}_1}{T}$

$a_3 = -\frac{2}{T^3}(\theta_1 - \theta_0) + (\dot{\theta}_1 + \dot{\theta}_0),$

$\theta_0, \dot{\theta}_0$ = initial position, velocity

$\theta_1, \dot{\theta}_1$= final position, velocity

T = traversal time.

In the case of the example command specification given above, the desired traversal time T_{segd} would be used for the parameter T in the above equations. The resulting trajectory equations would then be checked and modified to make sure joint velocity and acceleration bounds will not be exceeded. If the desired traversal time is not indicated in the objective function, then the Prim Planning module must determine an appropriate value, probably based on some heuristic. The planned trajectory functions are sent to the Execution module (see sec. 8), which evaluates them for the desired intervals and commands the results to Servo. Similarly, higher order polynomials may also be used, if continuous acceleration is also desired [11,15]. Parabolic blends may also be used between trajectory points, as can transcendental and exponential functions [11,15]. Still other trajectory generation functions for splining together points in joint space are discussed in [30,31,34].

6.2 Linear Cartesian Space Motion

In many cases, it is desirable to command motion segments in Cartesian space. Cartesian straight line paths are particularly useful for controlled trajectories in a cluttered environment and when the payload dynamics are dominant. Although Cartesian straight line motions cannot be performed exactly (for articulated manipulators), a reasonable approximation may be achieved by interpolating the manipulator through a large number of closely-spaced points in joint space such that the resulting motion is sufficiently close to the desired path. The command parameters for free space Cartesian motions are the same as those used for joint space motions, with C_z selecting one of the Cartesian frames mentioned previously. For Cartesian space motions, the commanded path $z_d(s)$ must specify Cartesian translation and rotation. It is also possible to command Cartesian motions using the "field" position command description. In this case, the (possibly position-dependent) directions in which the manipulator is to move are commanded in the form of a directional field.

The translation portion of the commanded Cartesian path consists of a three-component vector function, which will be represented by $X_d(s)$. To command a Cartesian straight line path segment from position X_0 to position X_1, for example, the position function would be

$$X_d(s) = X_0 + s(X_1 - X_0).$$

The orientation specification for Cartesian space motions is somewhat more difficult to visualize. Consider an initial orientation given by a vector n_0 and an angle θ_0, written as $\mathrm{Rot}(n_0, \theta_0)$, and a goal orientation $\mathrm{Rot}(n_1, \theta_1)$. For the moment, it will be assumed that the end-effector is to be smoothly reoriented about the single axis n_r by the angle θ_r which will take it from the initial orientation to the final orientation as in [55]. The orientation as a function of s may then be expressed as the initial rotation multiplied by a parameterized rotation:

$$\mathrm{Rot}(s) = \mathrm{Rot}(n_0, \theta_0)\ \mathrm{Rot}(n_r, s\theta_r).$$

Alternatively, the reorientation may be specified by a combination of two simultaneous rotations: one, $\mathrm{Rot}(k, \theta_k)$ that rotates n_0 to n_1 about a mutually perpendicular axis k, and another, $\mathrm{Rot}(z, \phi_z)$, that gives the rotation about the instantaneous approach axis z [44]. In this case, the rotation may be specified as:

$$\mathrm{Rot}(s) = \mathrm{Rot}(n_0, \theta_0)\ \mathrm{Rot}(k, s\theta_k)\ \mathrm{Rot}(z, s\phi_z).$$

Still other Cartesian orientation functions are possible, although they may be difficult to visualize.

A widely known approach to Cartesian motion planning is that of Paul [42,43,44,45], who uses a time-varying "drive" transform to take the manipulator from the initial to the final configuration along a straight path. The drive transform is a parametrically-defined matrix that relates the goal position to the current position.

For constant-velocity motion from X_0 to X_1, Paul uses the functions

$$X(t) = X_0 + \frac{t}{T}(X_1 - X_0)$$

and

$$\mathrm{Rot}(t) = \mathrm{Rot}(n_0, \theta_0)\ \mathrm{Rot}(k, \frac{t}{T}\theta_k)\ \mathrm{Rot}(z, \frac{t}{T}\phi_z).$$

To *transition between path segments*, Paul uses a polynomial interpolation function for both translations and rotations. Assuming that the transition time $2\,t_{acc}$ required to accelerate from the velocity of one segment to the velocity of the next may be determined based on some available acceleration and/or the amount of allowable path deviation at the intermediate point, the manipulator position during the transition from the segment from P_0 to P_1 to the segment from P_1 to P_2 may be given by

$$X(t) = X_0 + f(\Delta X_1, \Delta X_2, T_1, T_2, t_{acc}, t)$$

$$k(t) = \mathrm{Rot}(k_1 x\, k_2, \frac{t + t_{acc}}{2\, t_{acc}} \cos^{-1}(k_1 \cdot k_2))\, k_1$$

$$\mathrm{Rot}(t) = \mathrm{Rot}(n_0, \theta_0)\mathrm{Rot}(k, f(\Delta\theta_1, \Delta\theta_2, T_1, T_2, t_{acc}, t))\mathrm{Rot}(z, f(\Delta\phi_1, \Delta\phi_2, T_1, T_2, t_{acc}, t))$$

for $-t_{acc} < t \leq t_{acc}$, where k_1 and k_2 are unit vectors and $f(\Delta_1, \Delta_2, T_1, T_2, t_{acc}, t)$ is an appropriate interpolation function. For example, a polynomial function may be used to connect the segments:

$$X(t) = X_0 + \Delta X_1 \frac{T_1 - t_{acc}}{T_1} + [(\Delta X_2 \frac{t_{acc}}{T_2} - \Delta X_1)\, h + 2\, \Delta X_1]\, h$$

where $\Delta X_1 = (X_1 - X_0)$

$\Delta X_2 = (X_2 - X_1)$

$h = \dfrac{t + t_{acc}}{2\, t_{acc}}$

$-\, t_{acc} < t < t_{acc}.$

Determining the drive transform defines the manipulator motion as a function of time (when used in Paul's "fundamental equation of manipulation"), and this is performed by the Prim Planning. The drive transform is then evaluated by the Prim Execution module at a fixed sample rate, which is set fast enough to yield a sufficiently accurate Cartesian path. The result is used to generate the string of Servo command positions required to perform the motion.

Taylor [55] has suggested several modifications to Paul's technique. First, he has suggested the use of quaternions as a more efficient representation of rotations. In addition, Taylor suggests using a rotation about a single axis, rather than two separate rotations, to achieve new orientations. The third modification is to compute the instantaneous position by subtracting a shrinking value from the destination rather than adding a growing value to the initial position. That is, form P_0 to P_1, the motion would be defined by

$$X(t) = X_1 - \frac{T - t}{T}(X_1 - X_0)$$

and

$$R(t) = R_1\, \mathrm{Rot}(n_R, -\frac{T - t}{T}\theta_R).$$

In this manner, small changes in the destination location may be automatically compensated for. To transition between segments, a constant acceleration may be assumed, from which an expression of position is obtained through integration, as an alternative to a matching polynomial. In this case, the trajectory function for the transition is the same as that given in the previous section for constant acceleration transitions between straight line joint space segments.

Another approach to trajectory planning is to use energy methods to set up artificial dynamic potential functions, or fields, which will be evaluated as the manipulator performs the motion [40]. Such an approach is particularly consistent with the directional field command description. The commanded field can be thought of as an attractor field, but without magnitudes assigned. The magnitude of the attractor field can be assigned by Prim based on the desired dynamic performance, maximum kinetic energy, and actuator capabilities. Similarly, braking fields can be devised which will result in the manipulator stopping as it reaches the goal. The superimposed fields can be logically combined, and the gradient of the resultant field used to determine the commanded manipulator forces. Such an approach is applicable to minimum-time motion planning, obstacle avoidance, and time-optimal interception of moving targets without impact [40].

6.3 Optimal Trajectory Planning

There has been a good deal of interest recently in developing algorithms that, given a parameterized path specification, will determine the time sequence of torques required to travel the path in minimum time or some other optimal fashion [10,16,27,46,51,52]. These algorithms all use the manipulator dynamics and actuator torque constraints in computing the optimal trajectory. The required input from E-move is a parameterized path description, an algorithm specification, and a performance index, all of which are provided by the proposed interface.

As an example of this type of algorithm, the minimum-cost trajectory planning (MCTP) technique [51] will be discussed. The function of MCTP is to determine the control signals that will drive a given robot along a specified curve in joint space with minimum cost, given constraints on the magnitudes and derivatives of the control signals. For MCTP the commanded geometric path is assumed to have the form

$$\theta^i = f^i(s), \quad 0 < s < s_f$$

where θ^i represents the position of the ith joint for an n-jointed manipulator. The path parameter may be limited to $0 < s < 1$ without loss of generality. Either E-move commands a *joint space path directly* or E-move commands Cartesian path segments, which must be splined into a joint space path by the Prim Planner before applying MCTP.

If symmetrical actuator torque constraints are assumed, they may be given in terms of the robot's position and velocity as

$$F \in F(\theta,\dot{\theta})$$

where F is the vector of actuator torques/forces and $F(\theta,\dot{\theta})$ is the set of realizable actuator efforts. If the actuator torque constraints are direction-dependent, then two sets of constraints may be provided. In addition, it may be desirable to limit the derivatives of the joint torques. This constraint may be expressed as

$$|\dot{F}| < K$$

where K is a constant.

The manipulator dynamics are needed in terms of the parameterized path, which results in the form

$$F_i = J_{ij}\frac{df^j}{ds}\dot{\mu} + (J_{ij}\frac{d^2f^j}{ds^2} + C_{ijk}\frac{df^j}{ds}\frac{df^k}{ds})\mu^2 + R_{ij}\frac{df^j}{ds}\mu + G_i$$

where J_{ij} is the inertia matrix

C_{ijk} is the array of centrifugal and Coriolis coefficients

R_{ij} is the viscous friction matrix

G_i is the gravitational loading vector

$\mu = \dot{s}$ is the path velocity

The cost function for MCTP takes the form

$$C = \int_0^1 L(s,\theta,F)ds.$$

For example, the cost function

$$C = r_t \int_0^1 \frac{1}{\mu}ds + r_e \int_0^1 \mu^2 R_{ij}\frac{df^i}{ds}ds$$

minimizes a combination of traversal time and frictional losses. The MCTP problem is then to minimize the cost function subject to the actuator torque constraints and the manipulator dynamics.

For the Prim level currently discussed, the bounds on torque and the time derivative of torque may be accessed from the world model. The world model is also the logical place to retrieve the J_{ij}, C_{ijk}, R_{ij}, and G_i terms of the dynamic equations. The commanded path and cost function are provided by the input command from E-move. Once so defined, the MCTP problem can be solved, for example, using dynamic programming as shown in [51]. The solution is based on calculating the cost associated with discrete points on a grid which divides the phase plane (s - μ plane), and then applying the dynamic programming technique to find the optimal positions and velocities.

As an example of how an MCTP motion is commanded, consider the following input command specification:

Command number	=	n
Prim algorithm	=	MCTP
Coordinate system	=	(world, -, -)

Pos. cmnd. descr.	=	$(X_d(s) = (1 - s)^3X_0 + 3s(1 - s)^2X_1 + 3s^2(1 - s)X_2 + s^3X_3, e_{01})$
Force cmnd. descr.	=	null
Held object	=	null
Destination object	=	null
Termination cond.	=	X_f within e_{01} of X_3, $\dot{X}_f = 0$
Red. resol. spec.	=	default
Priority	=	1
Objective function	=	minimize OF = T_{seg}

The position path description in this case defines a cubic Bezier spline segment where the points X_0 through X_3 are control points which describe the start, intermediate shape, and end of the curve [16,19]. The path is described in world coordinates. The termination conditions are that the manipulator be within tolerance of the goal point and stopped. The objective function indicates that the cost to be minimized is proportional to segment traversal time.

6.4 Taking Advantage of Path Tolerance

The algorithms discussed up to this point have not addressed the issue of taking maximum advantage of the positional freedom allowed by the positional path tolerance. For many paths the positional tolerance may be significant. Often there will be paths within the cylindrical path description which are more efficient than the one that follows the spine along the center of the path. Taylor recognized this, and devised the bounded deviation strategy as an alternative approach to performing Cartesian trajectories. Given a Cartesian straight line path and an allowable deviation, the bounded deviation strategy will determine the number of points to be traversed in joint space which will keep the trajectory within the stated bounds. This strategy would make direct use of the position path deviation specified in the input command for the entire path, not just at via points. The steps of the algorithm for Cartesian straight line motion from P_0 to P_1 with allowable path position deviation e_p and rotation deviation e_r are as follows [55]:

1. Compute the joint configurations θ_0 and θ_1 which correspond to P_0 to P_1.
2. Compute the joint space midpoint, $\theta_m = \theta_1 - \frac{(\theta_1 - \theta_0)}{2}$. Use θ_m to compute X_m and R_m, the Cartesian position and orientation corresponding to θ_m.
3. Compute the Cartesian path midpoint, $X_{cm} = \frac{(X_0 + X_1)}{2}$, $R_{cm} = R_1\,\mathrm{Rot}(n_r, \frac{-\theta_r}{2})$, where $\mathrm{Rot}(n_r, \theta_r) = R_0^{-1} R_1$.
4. Compute the deviation between X_m and X_{cm},

$\Delta_p = | X_m - X_{cm} |$

$\Delta_r = |$ angle part of $R_{cm} R_m |$.

5. If $\Delta_p \leq e_p$ and $\Delta_r \leq e_r$, then the procedure is finished. Otherwise, the joint solution θ_m corresponding to the Cartesian midpoint is computed and steps 2-5 are applied recursively for the path subsegments until a sufficient number of joint space positions are identified.

The joint positions as a function of time are then computed using the interpolation technique described previously for joint space motions.

Taylor's bounded deviation algorithm increases computational efficiency, since it minimizes the number of kinematic inversions required to follow a Cartesian straight line path with the desired accuracy. The resultant path is not strictly guaranteed to stay within the Cartesian bounds, however, since the maximum deviation does not necessarily occur at the midpoint of the Cartesian straight line. Also, the trajectory knot points are required to be on the spine of the tubular path, which may not result in the "best" path through the tube. Other paths may be faster or more energy efficient. Finding a true minimum-time trajectory through a generalized cylinder path, while respecting actuator torque and velocity, and joint position limits is a difficult problem. Minimum-time trajectory planning has been investigated by a number of authors, as discussed in section 5.3; however, in most cases they do not take advantage of the additional position freedom provided by the path tolerance.

The use of path tolerance in minimum-time trajectory planning has been addressed by Suh and Bishop [53], however. The algorithm discussed in [53] consists of two phases. First, an optimal set of quartic polynomials which connect points on the spine which are equally-spaced in time is determined. Then, using this as an initial feasible solution, the gradient method is used to find the optimal placement of knot points when they are not required to lie on the spine. The path description consists of straight line cylindrical segments between reference points, with a specified cross sectional radius for each segment. Although it is theoretically possible to perform the minimum time trajectory planning for both translations and rotations, the problem is more tractable if the rotation part is ignored. Suh and Bishop do this, by assuming a spherical approximation to account for the swept volume of the end-effector during wrist motions. While this may be appropriate in some instances, requiring a free volume to be large enough to allow any orientation of the end-effector may at times be too restictive, especially if a large object is held. Also, this type of algorithm takes a relatively long time (on the order of minutes) to plan, and once planned, such time-optimal trajectories are not easily modified in real time (in response to sensor data, for instance), since any modification would require reevaluation of the actuator and path constraints for the new trajectory. Thus, it may be most appropriate for use in off-line planning of trajectories that will be executed many times.

6.5 Sensory-Interactive Trajectory Generation

There are many situations where it may not be possible or appropriate to define the desired path of the manipulator a priori. In some cases, it may be more appropriate to simply command a goal state which defines *what* is to be achieved, along with an algorithm specification that defines *how* to achieve it. This is true for vision-servoed trajectories, for example

[5,29,56]. These types of algorithms perform what is referred to here as *sensory-interactive trajectory generation.* As mentioned previously, sensory interaction may be accomplished either by frequent replanning of the trajectory as it is executed or by planning a trajectory function which represents a general profile of the desired motion, with the trajectory details produced dynamically as the trajectory is executed. Examples of both approaches will be discussed here.

Consider the task of moving an object held by the manipulator end-effector to a position relative to a moving object, as shown in figure 9. The vector r_m defines the position of the manipulator end-effector relative to the base, or world frame of the manipulator. The location of the goal position relative to the same world frame is given by r_o. The vector r_e represents the position of the goal relative to the current end-effector position; in other words, the error (in Cartesian space) which must be nulled in order to accomplish the motion. The object moves with some velocity v_o. Assume cameras are available which may provide ro or re directly (although this is non-trivial).

For the replanning approach, one possibility is to determine the goal position relative to the world frame, plan a trajectory which takes the end-effector to the goal state, and replan the trajectory periodically with updated information about the location of the goal. In planning the trajectory, the current position and velocity of the object should be used to predict the goal position at the end of the planned motion. This estimation need not be extremely accurate at the beginning of the trajectory, since future replanning will provide better estimates. Andersson [5] uses this approach (although in joint space) to move the paddle of a ping-pong playing robot to predicted hit locations. A command specification for this type of trajectory would look like the following:

Command number	=	n
Prim algorithm	=	joint space quintic polynomial with replanning
Coordinate system	=	(world, block.hole, peg.tip)
Pos. cmnd. descr.	=	null

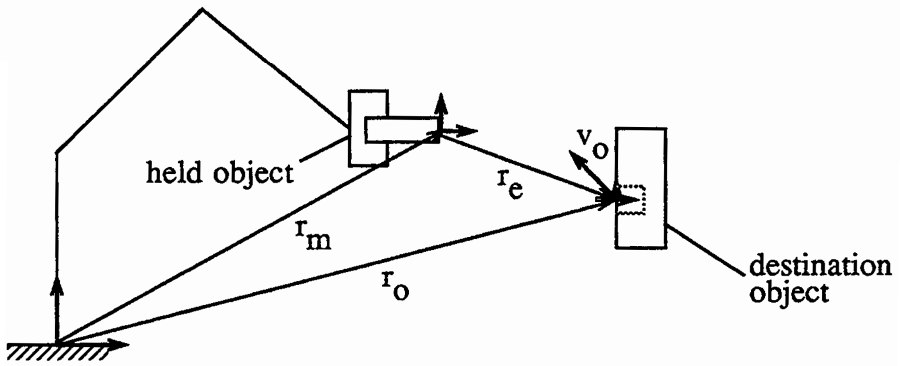

Figure 9. Position relationships for sensory-interactive motion.

Force cmnd. descr.	=	null
Held object	=	peg
Destination object	=	block
Termination cond.	=	sensed position and orientation within tolerance of goal (Cartesian position of peg.tip with respect to block.hole)
Red. resol. spec.	=	default
Priority	=	1
Objective function	=	minimize OF = $(t_f - t_{fd})^2$

Notice that, although the trajectory will be planned and executed in joint space, the goal position is defined in Cartesian space. The end-effector offset frame is indicated by an object name including the name of a particular feature. The offset of the world frame is also indicated symbolically, by giving the name of the destination object and a particular feature on the object. For the trajectory algorithm described presently, these offsets must be evaluated by obtaining the positions of the held and destination objects relative to the end-effector and the robot base, respectively. A trajectory function to perform the motion is planned by obtaining this information from the world model and computing coefficients as before. Since the world frame offset changes with time, however, the position of the destination object will have to be retrieved periodically and used to replan the trajectory as it is being performed. This involves estimating what the manipulator and object states will be at the start of execution of the updated trajectory function, and using this information to determine the new function itself.

The cameras (or other sensors) must be calibrated with extreme care for this approach to be successful, and if the robot trajectory is described in Cartesian space, the kinematic model of the robot must be highly accurate, as well. While this approach may be appropriate in some situations, a more robust approach in terms of sensitivity to modeling inaccuracies is to measure the relative position (error) directly. This error may then be used to generate an incremental motion toward the target. This approach has been used for simulated satellite docking [29]. The algorithm used in [29] computes an appropriate portion of the total Cartesian error to move each cycle, where the magnitude of the increment (and therefore, the end-effector velocity), is a function of the distance of the end-effector from the target.

For this type of algorithm, the Planning module doesn't plan functions of manipulator position, velocity, and acceleration as functions of time, but rather computes only the trajectory parameters which affect the general profile of the motion. Once the trajectory parameters for the desired velocity profile have been planned, the execution takes place without replanning, and the exact manipulator trajectory is determined as it is executed. The Execution module obtains updated object position information from the world model every cycle in this case. The command specification for this type of algorithm is similar to the previous one. However, the offset of the world frame (the position of the object relative to the robot base) in this case need not be explicitly evaluated—the sensory-interactive motion is performed by making incremental steps until the relative position goal has been achieved.

Although much of the work is performed by the Execution module for algorithms of this type, the Planning module does perform some functions. For example, the Planning module may select Servo gains directly, or it may decide to let the Execution module determine configuration-dependent gains as the trajectory is executed. The Servo algorithm must also be selected, if it has not been specified in the input. In addition, the Planning module may determine certain trajectory parameters which control the motion profile (velocity as a function of distance from the target, for example). The execution of sensory-interactive commands is discussed in section 8.2.

6.6 Biologically-Motivated Trajectory Techniques

Related to sensory-interactive trajectory generation is an approach used by Bullock and Grossberg [13] to explain some characteristics of biological movements. In this approach, the trajectory (positions, velocities, and accelerations) is not planned ahead of execution. Rather, the trajectory is determined as it is being performed (as with sensory-interaction), and comes about by multiplying time-varying position and velocity gains by the difference between the current and target joint space positions. The shape and speed of the movement is determined by the gain function, which could be specified by the Planning module. This type of trajectory generation could be commanded by specifying the goal position in the termination condition list as with sensory-interactive trajectories, and indicating the proper algorithm. Although this algorithm does not provide for gravity and dynamics compensation, and further analysis and experiments would have to be performed to determine the usefulness and stability of the technique in manipulator applications, it is interesting in terms of its apparent similarity to some human movements.

6.7 Trajectory Planning in a Non-Stationary Environment

In general, the environment in which a manipulator operates will have in it other objects that move; other manipulator arms, machines, human trespassers, for example. The E-move path planner is concerned with the movement of such obstacles in terms of allocating the workspace resource, but cannot effectively alter manipulator trajectories to avoid rapidly-moving obstacles since it has very limited control over manipulator behavior in terms of time. The responsibility of avoiding moving obstacles in the near term is logically a function of the Prim level, which determines the time history of manipulator movement. There are two cases of object motion that need to be addressed. The first is when the object motion (current and future) is known or can be estimated with a large degree of confidence when the trajectory is being planned. For instance, the planned trajectories of other manipulators in the work volume may be available through the world model. In this case, the motion of the other objects may be included in the planning process, and some work in this area will be discussed in the following paragraph. In the other case, an unanticipated object may be detected as the manipulator is executing a planned trajectory. Any obstacle avoidance maneuvers which are to be performed in this case are determined by the Execution module. Such *reflexive* obstacle avoidance actions are discussed in section 8.5.

Kant and Zucker [25] have proposed that the trajectory planning problem (TPP) be decomposed into a static path planning problem (PPP) and a temporal velocity planning problem (VPP). Assuming that E-move has solved the PPP, it remains for the Prim Planning module to solve the VPP. In [25], velocity functions are determined by searching for free paths in path-time ($s \times t$) space. Obstacles crossing the path of the manipulator are represented as rectangles in this space. A path is then found in $s \times t$ space which avoids the for-

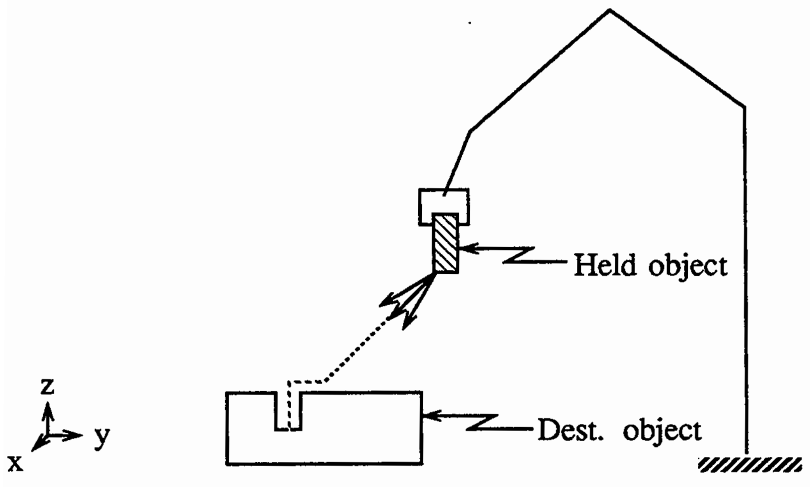

Figure 10. Generalized damper motion.

bidden areas, meets monotonicity and maximum velocity constraints, yields a smooth velocity profile (finite acceleration), and meets some other criterion such as arriving at the goal at a fixed time or in minimum time. This approach works best in environments where obstacles are sparsely distributed and move relatively orthogonally to the manipulator path, since it does not allow the manipulator to leave the planned path. The development presented in [25] deals with point robots; real manipulator end-effector dimensions may be accounted for by "growing" the obstacles, but manipulator links pose a more difficult problem. Also, since the approach assumes complete, accurate knowledge of object trajectories, these trajectories must be estimated frequently if they are not known exactly. Consequently, the manipulator trajectory must be replanned frequently as well, in this case.

6.8 Generalized Damper Motion

Much of the work to date toward the development of automatic fine motion planners [17,32] has assumed that the manipulator is capable of executing generalized damper motions [57]. Generalized damper motions are executed by commanding a nominal velocity in the desired direction and specifying the relationship between the velocity error and contact forces. This type of control is intended to enable the manipulator to slide objects along constrained directions in order to achieve assembly goals, as shown in figure 10. An example command specification for the motion shown in figure 10 is given below.

Command number	=	n
Prim algorithm	=	Generalized damper
Coordinate system	=	(world, block.hole, peg)
Pos. cmnd. descr.	=	(0 i - .707 j - .707 k, e_f)
Force cmnd. descr.	=	null
Held object	=	peg
Destination object	=	block.hole

Termination cond.	=	velocity = 0 and peg in hole to within tolerance
Red. resol. spec.	=	default
Priority	=	1
Objective function	=	null

In this example, world coordinates are chosen (although end-effector coordinates would serve just as well), and a constant motion direction field is used as the position command description. This means that E-move has determined that the insertion operation can be performed by commanding the manipulator to move in a constant direction from the current position. The commanded direction should be within the given tolerance, which defines the allowable variation of the direction of the commanded velocity at each point. In order to plan this motion, Prim needs to determine an appropriate velocity and reasonable servo gains. Generalized damper movements for assembly tasks will typically not require moving large distances, which means that Prim can command low velocities to minimize the energy of impact without incurring a serious time penalty. The velocity error gain must be set fairly low (a low damping coefficient) to prevent excessive forces when the manipulator is displaced from the commanded direction due to sliding along a constrained surface. Prim might reason the gains from object characteristics, such as stiffness [57]; or, more likely, specific values of velocity and damping coefficient might be stored under the object names.

This example has not dealt with rotations, but a rotational field can be commanded similarly. One possibility is to consider the rotational subspace as a sphere of radius 2π, with the direction of a vector from the origin to a point within the sphere giving the orientation of the gripper axis and the length giving the roll of the gripper. A vector field may be described within this subspace to specify the direction in which to rotate, for each orientation. For instance, a constant rotational field about an arbitrary axis $x_0\, i + y_0\, j + z_0\, k$ may be given as:

$$(y_0 z - z_0 y)\, i + (z_0 x - x_0 z)\, j + (x_0 y - y_0 x)\, k.$$

The output from the Planning to the Execution module for field commands such as this generalized damper motion consists primarily of a velocity field (the direction field scaled by the Prim-determined velocity), and the servo loop gains. The Execution module must evaluate the field equations for the current manipulator position on every cycle in order to calculate the next command to Servo.

Generalized damper motions are a subset of what is known as *impedance* control [22,48,56]. Impedance control may be used for free space and constrained motions alike, by commanding a nominal position path (or direction) and controlling the manipulator impedance to prevent excessive reaction forces when contact is made.

6.9 Hybrid Force/Position Motion

A different approach to performing constrained motion tasks is to explicitly command desired force and position paths (or directions). This type of control has become known as hybrid force/position control [47]. Using this approach, forces and positions are both commanded explicitly in the desired reference coordinate system, as depicted in figure 11. If the de-

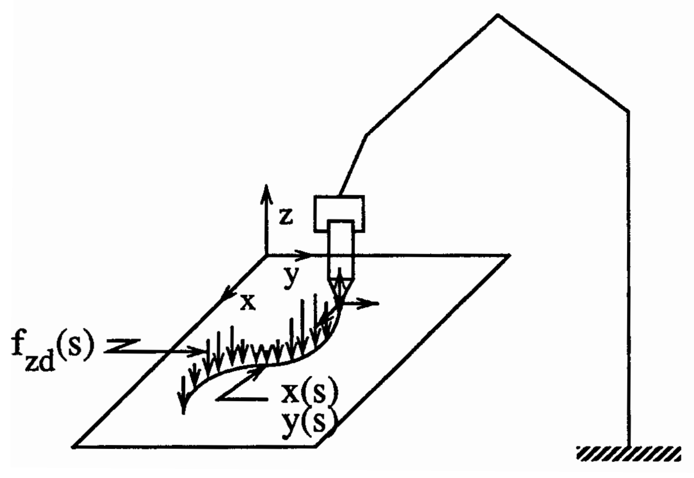

Figure 11. Hybrid force/position motion.

sired motion/force is commanded with respect to the end-effector, then forces and motion directions are specified by E-move. In this case a position path cannot be commanded since the coordinate system moves with the end-effector. If a world coordinate system is specified, the commanded motion consists of the desired forces and either a position path or a motion direction. At any given instant, certain freedoms of the manipulator will be free for position control, and the remaining orthogonal freedoms will be constrained, requiring force control [36]. *Either* position *or* force will be controlled along any given degree of freedom. This is enforced by the selection matrix that is commanded to the Servo level. The selection matrix is based on the *constraint frame*, which describes the constraint situation of the manipulator. It is the task of the world model to determine the constraint frame of the manipulator at any given time.

Three pieces of information are needed to perform a hybrid force/position motion: the desired nominal force/position trajectory, the trajectory of the constraint frame, and which freedoms of the constraint frame are constrained. The first of these is provided by the command from E-move. The second two should be available from the world model. There are several approaches to determining the constrained directions of manipulator motion. One possibility is to rely on predefined position and geometry information. This approach has the disadvantages of requiring geometric reasoning capabilities, and of not being very robust to modeling inaccuracies. Recently, several researchers have begun to investigate the possibility of *sensing* the manipulator constraints. For instance, Asada and Izumi [6] use force and position measurements, obtained while a human operator manually leads the manipulator through the task, to generate a sequence of force and position (without changing orientation) commands in the proper directions to perform the task automatically.

The process of determining the constraints begins with decomposing the sensed reaction force into tangent and normal components, with the tangent direction determined by differentiating the sensed position data. For these two directions, position control can be used for motion along the tangent, while force control should be used in the normal direction. For the remaining degree of freedom perpendicular to both the tangent and the normal directions, some assumptions about the task geometry and the order of reducing the positional freedoms are used to disambiguate the proper control mode. Although the interpretation of sensory information takes place off line in [6], some of the techniques described could be used with instantaneous data in real-time; for instance to determine the proper control modes for following an unknown surface. A somewhat similar approach is used by Merlet [38] to determine the constraint frame. Merlet, however, identifies the constraint situation by evaluating the measured positions and forces encountered when small successive motions are commanded along each degree of freedom. Further information on using force measurements to determine contact geometries and reduce positional uncertainties may be found in [50] and [9].

6.10 Kinematic Redundancy

Kinematic redundancy can add another dimension of complexity to the Prim Planning process. As mentioned previously, if the technique used to resolve the redundancy performs the function of transforming a static motion description (path or direction) into a dynamic trajectory, then it should be executed by the Prim Planning module. If the redundancy resolution is a static inverse kinematic transformation, it is also performed by Prim, but at the Execution level. If the redundancy resolution technique is an integral part of a servo algorithm that outputs joint space torques given a Cartesian input command, then the redundancy resolution takes place at the Servo level and is specified by the algorithm.

An example of redundancy resolution performed by the Prim Planning module is when global optimization techniques are used to determine the joint positions or torques for an entire trajectory [53,39]. Nakamura's method makes use of Pontryagin's Maximum Principle to determine how the extra degree(s) of freedom may best be used over an entire trajectory to globally optimize a specified performance index. Hollerbach uses a similar method which is theoretically equivalent, but less formidable to compute.

7. Prim Execution Module Interfaces

The Prim Execution module shares an interface with the Planning module, the world model, and the Servo Level Task Decomposition module. This section describes the information passed across each of these interfaces, which are shown in figure 12.

7.1 Prim Planning to Execution Interface

Having planned the manipulator position and force trajectory and appropriate gains, the Planning module sends the following information to the Execution module:

Command number

A command number is passed from the Planning module to the Execution module to indicate when a new information is being sent. This command number is independent of the numbers associated with Job Assignment and Planner module commands.

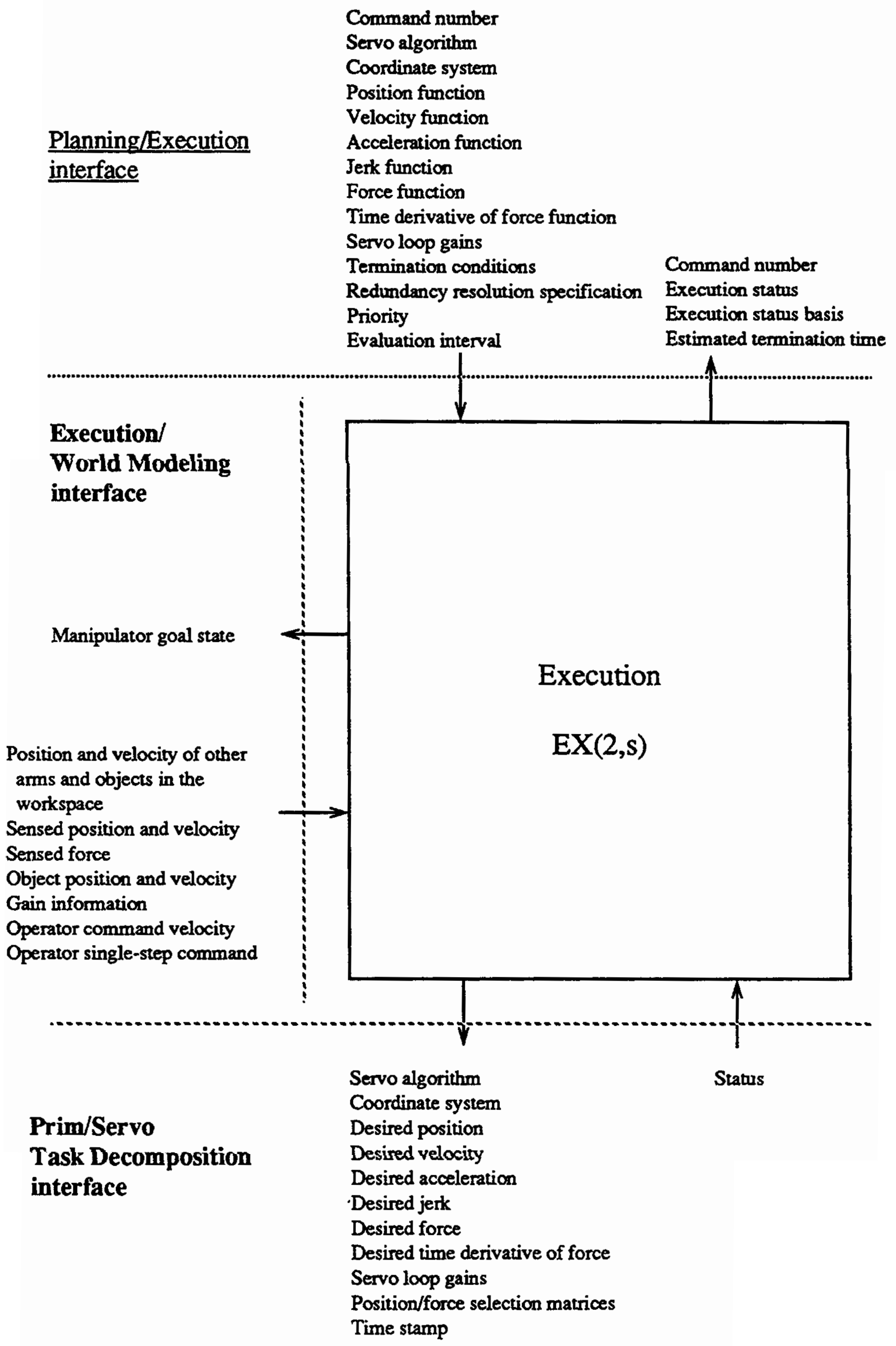

Figure 12. Prim execution module interfaces.

Servo algorithm

The Servo algorithm specifies the Servo level behavior to be used in executing the command.

Coordinate system, C_z

The coordinate system again determines the reference frame in which the command is to be interpreted.

Position function, $z_d(t)$

The position function specifies the desired position as a function of time. It represents the plan for the desired trajectory.

Velocity function, $\dot{z}_d(t)$

The velocity function gives the desired velocity as a function of time. It must be compatible with the position function, if one is specified.

Acceleration function, $\ddot{z}_d(t)$

The acceleration function indicates the desired acceleration as a function of time. Again, it must be compatible with the other trajectory functions.

Jerk function, $\dddot{z}_d(t)$

The jerk function specifies the desired rate of acceleration as a function of time. Control of the jerk may result in less drivetrain wear and decreased excitation of structural resonances [44]. The jerk function must also be compatible with the other trajectory functions.

Force function, $f_d(t)$

The force function provides the desired force as a function of time.

Time derivative of force function, $\dot{f}_d(t)$

Sometimes it may be desirable to command the rate of change of force as a function of time. This function allows such a command.

Servo loop gains, K_p, K_v, K_i, $K_{pf,}$ K_{vf}, K_{if}, K_τ

The gains which are to be used in the control laws of the Servo level may be determined by the Prim Planning module and commanded to Servo through the Execution module. The gains may be functions of time to be evaluated by the Execution module.

Termination conditions, TC

The termination conditions are passed through the Planning module to the Execution module, which checks for their attainment.

Redundancy resolution specification, R

When a trajectory plan is specified in Cartesian space and it is desired to servo the arm in joint space, the Execution module will be required to perform inverse kinematics. In order to do so, the method to be used to resolve redundancy (for manipulators with more than 6 dof) must be given. The redundancy resolution specification will either have been determined by E-move and passed down through the Planning module, or the Planning module itself will have determined which method to use.

Evaluation interval

The evaluation interval specifies how frequently the position and force trajectory functions are to be evaluated. For predetemined trajectories (where desired position and force have been specified as a function of time), the evaluation interval indicates the particular values of time that the functions are to be evaluated for. Note that in this case, the actual evaluation need not occur at the times specified by the evaluation interval. For sensory-interactive trajectories, however, the evaluation interval gives the desired cycle time for executing the trajectory generation algorithm. Thus, execution of a sensory-interactive trajectory will result in commands to Servo every evaluation interval.

The Execution module returns the following items back to the Planning module:

Command number

Execution status

Execution status basis

Estimated termination time

These items are the same as those discussed in section 2.1, except for the command number, which simply echos the input command number.

7.2 Prim Execution to World Modeling Interface

The Execution module needs several different types of information from the world model. It requires the sensed position and velocity of the manipulator to evaluate termination conditions. Other sensed values available from the world model may be needed for the evaluation of termination conditions, such as sensed forces or tactile information. For sensory-interactive trajectories, the Execution module needs to know the position and velocity of objects which are being tracked, and the values of sensors (as interpreted by the world model) which may be used to control the trajectory manually, such as a joystick or "sensorball" [22]. The exact information required from the world model will depend on the nature of the sensory-interactive algorithm, so it is not possible to delineate all possible information which may be needed from the world model, but these are some examples. In addition, the Execution module may obtain manipulator state-dependent gains from the world model to use in the command to Servo. The output of the sensor used to manually-control the manipulator velocity must also be available, as well as the control signals required to perform single-stepping of Execution module operation.

7.3 Prim Execution to Servo Task Decomposition Interface

The output of the Prim Execution module consists of commands which contain the following information:

Servo algorithm

The Servo algorithm specifies the control algorithm to be used by the Servo level in trying to achieve the commanded goal state. It determines the general behavior of the manipulator in response to errors and disturbances.

Coordinate system, C_z

The coordinate system indicates the reference frame of the commanded goal state. Again, the format of C_z is (coordinate system, T_b, T_e). For the Prim-to-Servo interface, the coordinate system can have the following values:

(motor, -, -)	=> path description in motor space
(joint, -, -)	=> path description in joint space
(end-effector, -, T_e)	=> motion direction or force in Cartesian system fixed with respect to the end-effector
(world, T_b, T_e)	=> path description of Cartesian system fixed with respect to the end-effector in terms of Cartesian system related to the manipulator base by T_b

The coordinate systems indicate the same relationships as those discussed in section 2.1. In Prim output commands to Servo however, the transformations T_b and T_e are not allowed to be defined symbolically (i.e. by names of objects). If a transformational relationship is time-varying, it must be updated in the commanded C_z, not by the world model. The reason for this restriction is to be able to meet strict timing requirements at the Servo level.

Desired position, velocity, acceleration, and jerk; z_d, $\dot{z}_d$, $\ddot{z}_d$, $\dddot{z}_d$

The desired position, velocity, acceleration, and jerk indicate the position-related portion of the commanded attractor set, in the specified coordinate system.

Desired force and time derivative of force; f_d, $\dot{f}_d$

The desired force and time derivative of force form the force-related portion of the commanded attractor set, also in the coordinate system specified by C_z.

Servo loop gains, K_p, K_v, K_i, K_{pf}, K_{vf}, K_{if}, K_τ

The Servo loop gains, along with the Servo algorithm, are the predominant factors in determining the manipulator behavior.

Position/force selection matrices, S, S′

The position/force selection matrices specify which degrees of freedom are to be position-controlled and which are to be force-controlled. They may also be used as generalized task specification matrices [26].

Time stamp

The time stamp parameter contains two pieces of information, the synchronization flag and the time variable, t_p. The time stamp is used to direct the timing of Servo command execution, as described in section 8.1.

A single parameter is returned to Prim from Servo:

Status

The status parameter from Servo contains the status of execution of the most recent command. As a minimum, the Servo level must be able to inform Prim of the following conditions:

executing	=> Servo is processing the most recent command
ready-next-point	=> Servo Job Assignment is ready for additional data
command error	=> Servo could not process the last command
Servo error	=> Servo encountered a fatal error during execution

8. Prim Execution Module Operation

The Prim Execution module has two primary functions. First, it must evaluate the position, velocity, acceleration, jerk, force, and time derivative of force functions of time for the intervals specified by the Planning module. This results in the point attractor vectors which are sent as commands to the Servo level. In addition, the Execution module is responsible for monitoring position, velocity, force, and other sensor states or world model conditions for achievement of the termination conditions.

For a limited number of trajectory algorithms (those which plan a trajectory in Cartesian space, but then send joint space commands to Servo) the Execution module also performs static inverse kinematics. The Cartesian to joint space transformation is performed by Prim in this case so that the commands to Servo will be in the coordinate system used to compute the servo error [20]. However, for servo algorithms that compute a Cartesian error, the Servo module performs this transformation (typically Jacobian-based in this case).

The Prim Execution module must check for closeness to kinematic singularities as the trajectory is being evaluated and perform any trajectory modifications which may be necessary to negotiate the singularity in a controlled manner. Also, the resolution of redundancy, if it is performed by kinematic criteria, is performed by the Execution module. Furthermore, the Execution module incorporates the operator velocity control and single-step interactions. The Execution module also should calculate the estimated termination time, since it may alter the commanded trajectory as a result of these interactions. The following sections discuss various aspects of the operation of the Execution module.

8.1 Executing Completely-Planned Trajectories

The execution of trajectories which have been completely planned by the Planning module is fairly straightforward. The position and force trajectory functions are simply evaluated for the indicated time intervals, and the results are sent to the Servo level as commands. The commands to Servo are associated with the time they are to be executed through the use of the time stamp parameter. To synchronize with the Servo clock, the Prim Execution module sets the synchronization flag, and zeros the time parameter, t_p. Thereafter, the synchronization flag is reset and t_p is updated for each command to Servo. For hybrid position/force trajectories, there is the added complication of determining the constraint frame of the manipulator.

Most conventional robot controllers use joint coordinates for servoing, even if the desired trajectory is described in task (Cartesian) space. If Cartesian trajectories are to be executed in this manner, then the Prim Execution module must perform the following functions:

1. Evaluate the Cartesian trajectory functions for the desired time intervals.
2. Transform the resulting Cartesian goal position into joint space.
3. Plan a smooth joint space trajectory to traverse the joint space goals.
4. Execute the joint space trajectory and send the results to Servo, while checking the commanded joint velocities and modifying the output command as necessary to negotiate singular configurations.

It is possible, however, to use a servo algorithm that computes errors in the task space instead of joint space [15,26]. If a Cartesian space path is commanded from E-move, this is a much more appropriate means of servoing to the desired trajectory. In this case, the Prim Execution module does not need to perform any kinematic transformations or do joint space trajectory planning. All that is required is to evaluate the trajectory functions and send the resulting Cartesian goals to Servo. The Servo level Cartesian control algorithm must incorporate an effective and appropriate method for handling motions near/through singular positions. Prim should still check its output commands and scale or otherwise modify them to make sure they will be dynamically small in joint space as well as in Cartesian space, however. Negotiation of singular configurations is discussed further in section 8.3.

8.2 Executing Sensory-Interactive Trajectories

In the case of sensory-interactive trajectory algorithms which use replanning to respond to world modeling updates, the execution of replanned trajectories is performed in a manner similar to that for completely planned motions, in the previous section.

For sensory-interactive trajectories which use only a planned motion profile, the manipulator responds to world model information according to the commanded algorithm. For example, execution of the vision servo without replanning discussed in [29] consists of the following steps, which are executed according to the time interval specified by the Planning module:

1. Obtain the sensor pose and velocity from the world model, given the algorithm (the world model selects a suitable sensor).

2. Compute the translation and rotation increments of the sensor to go to the new goal, according to an exponential velocity law (as the goal is approached the velocity is decreased).

4. Obtain the desired manipulator motions from the world model given the desired sensor motions.

5. Determine the status of the command by comparing the current and goal positions.

6. Transform the Cartesian goal points into joint space commands, scaling for singularities and checking for joint limits.

7. For the first output of the command, set the synch flag to 1, and set t_p to 0. Thereafter, set the synch flag to 0 and increment t_p each execution cycle.

A typical output to the Servo level then, would look like:

Servo algorithm	=	joint PID
Coordinate system	=	(joint)
Desired position	=	θ_d
Desired velocity	=	null
Desired acceleration	=	null
Desired jerk	=	null
Desired force	=	null
Desired deriv. force	=	null
Servo loop gains	=	K_p, K_v, K_i
Pos./force sel. mat.	=	null
Time stamp	=	$(0, t_p)$

8.3 Negotiating Singular Configurations

An important aspect of trajectory generation in Cartesian space is the negotiation of the manipulator around or through singularities, or degenerate positions of the manipulator. Singular configurations present a problem when a Cartesian space trajectory is transformed into a joint space trajectory. A singularity causes a loss of a degree of manipulator freedom, and a direct result of passing near or through a singular configuration is that goal points which are closely spaced in Cartesian space may in fact require extremely large joint motions. Thus, in order for Prim to make sure that the commands sent to Servo are "small in a dynamic sense", the Prim Execution module should monitor the nearness of computed goals to singularities, and modify them as necessary before sending them to Servo. To the extent possible, this should be done even when Prim sends Cartesian goals to Servo, so that only dynamically realizable goals are commanded.

If Prim is sending joint space commands to Servo, there are several techniques available to modify goals in the Execution module. One approach is to simply check the calculated goal velocity of each joint, and when a joint reaches its maximum allowable velocity, scale back the other joint commands before sending them down to Servo. This results in the manipulator following the commanded trajectory, although at a diminished speed, and may be used with several different trajectory generation algorithms. Another possibility is to allow a small amount of orientation error near singularities [55]. In this case, the manipulator position and speed are maintained, but the orientation accuracy is compromised. A modification to inverse Jacobian rate control has been proposed in [1] for dealing with singularities in which wrist joint velocities are monitored. When a joint rate becomes excessive, it is limited to a maximum value and a modified inverse Jacobian is used to calculate the rates for the remaining joints. This control also results in (only) an orientation error from the desired motion. In other cases, maintaining orientation may be more important than achieving the desired position.

An effective means of handling singularities is to provide the manipulator with one or more redundant degrees of freedom. This will be discussed in the following section. Redundancy allows a variety of joint configurations for a given end-effector position. As mentioned previously, if the technique used to resolve the redundancy performs the function of transforming a static motion description (path or direction) into a dynamic trajectory, then it should be executed by the Prim level. Otherwise, the Servo level will perform redundancy resolution as indicated by the parameter R passed from E-move through Prim.

8.4 Kinematic Redundancy

Singularity avoidance is a prime motivation for using kinematically redundant manipulator mechanisms. Redundancy resolution is performed by the Execution module if a purely kinematic technique is used [7,8,28]. In addition to singularity avoidance, kinematic redundancy may be exploited to achieve other goals, as well. One approach to redundancy utilization is to view the various goals as prioritized subtasks [35,39,60]. For instance, following the desired end-effector path might be the highest priority goal. Given that this can be accomplished in an infinite number of ways with a redundant arm, other goals, such as singularity and obstacle avoidance, may be used to fully determine the joint trajectories.

8.5 Reflexive Obstacle Avoidance

The Prim Execution module should have some facilities for attempting to avoid unanticipated obstacles. This type of "reflexive" obstacle avoidance should be performed by the Execution module because the sensors used to detect such obstacles must be monitored at a higher rate than that used for planning (and replanning), and appropriate action must be taken immediately. At the Prim level, such reflexive behavior consists of modifying the ideal trajectory to prevent unwanted collisions, although no avoidance algorithm can guarantee collision-free motion in a dynamic environment [40]. A major problem in trying to incorporate reflexive behavior of any sort lies in defining what the "appropriate action" is, provided that some reflex situation has been identified (another difficult problem). A third problem is discriminating between desired and undesired collisions. For obstacle avoidance, one can imagine a multitude of proximity sensors which provide a sensing field of some useful range about the manipulator links, or at least those most likely to encounter unforeseen or misrepresented obstacles. The use of such a set of sensors for obstacle avoidance is discussed by Espiau and Boulic [18]. They interpret the sensor outputs as repulsive displacements which are applied to the nominal trajectory to prevent collision both for the end-effector and links of a redundant manipulator.

8.6 Operator Velocity Control

A very desirable characteristic of a control system architecture for telerobots is to provide a means by which a human operator can control the speed of manipulator movements. While it is fairly straightforward to provide a means for modifying the speed of an entire trajectory by applying a uniform time scaling [24], it is considerably more difficult to determine the effects of velocity changes commanded by the operator during execution of a motion, which are expected to be responded to in real time.

8.7 Operator Single-Step Control

Another desirable capability for telerobot control is to allow the operator to step through a trajectory by executing knot points one-at-a-time. This capability is often useful for testing purposes, and can be provided by having the Execution module monitor the state of single-step control switches. Based on the state of these switches, the Execution module can determine if the next command to Servo is to be issued immediately, or if it should wait for stepping of execution by the operator. The Execution module would reset the synchronization flag for every Servo command in this mode of operation. Of course, single-stepping capability may not be appropriate for some Prim algorithms (a Cartesian vision servo, for instance). Manual control of the manipulator will be discussed further in a subsequent document.

9. Implementation Issues

This section discusses two important implementational aspects of the Prim task decomposition module described in the preceeding sections; the size of the module interfaces and the computational requirements of the module.

9.1 Interface Size and Information Flow Rates

The input command interface, described in section 3.1, contains a considerable amount of information. The size of this interface will of course depend upon the details of the implementation, but a rough estimate may be made nonetheless:

Command number	=> 1 word
Prim Algorithm	=> 1 word
Coordinate System	=> 33 words
Position command description	=> 100 words
Force command description	=> 100 words

Held object	=> 3 words
Destination object	=> 3 words
Termination conditions	=> 13 words
Redundancy resolution specification	=> 1 word
Priority	=> 1 word
Objective function	=> 13 words
	269 words

If 32-bit words are used, then a complete command consists of slightly over 1 K bytes, which will be sent on the order of every 100–1000 ms. This implies a required communications bandwidth of about 80 K bits/s for the input command interface. The output status of the Prim task decomposition module is much smaller and adds little to the bandwidth requirement. Of course, it is likely that some of the parameters above would be implemented as ASCII strings, which would probably result in a somewhat larger interface.

The size of the output command interface to Servo is about 512 bytes [20]. For a Servo command update rate of 10 ms, the required bandwidth to update the entire command every cycle would be about 400 K bits/s.

9.2 Processing Requirements

The interfaces described above allow for a variety of trajectory generation techniques. The computational requirements for a few of the simpler algorithms will be discussed below. Most of the Prim task decomposition processing occurs in the Planning and Execution modules. Although the Planning process executes cyclically, the time between consecutive planning operations will vary (typically 40–500 ms), depending on trajectory generation algorithm, the length of the motion segment and the degree to which sensor feedback indicates that the motion is proceeding as planned. The Prim Execution process, on the other hand, must update commands to Servo every 10–20ms. The estimated computational complexity involved in the Planning and Execution processes for several simple algorithms follow. The numbers provided are for a single manipulator.

1. Joint interpolated trajectory composed of quintic polynomial segments; output joint position, velocity, and acceleration:

   Planning requirement: 98 n operations/planning cycle
   Executing requirement: 102 n operations/execution cycle,
   where n = number of manipulator joints

2. Cartesian straight line trajectory generator with 7th order polynomial transition between trajectory segments; output Cartesian position, velocity, and acceleration:

   Planning requirement: 240 operations/planning cycle

Executing requirment: 614 operations/execution cycle

3. Same as (2), but output *joint* space position (and not velocity or acceleration).

Planning requirement: 240 operations/planning cycle
Executing requirement: 656 operations for a 6–dof manipulator

Trajectory generators (1) and (2) provide a closely-spaced sequence of position, velocity, and acceleration goals, and allow for smooth transitions between path segments. Of course, it is possible to command only positions, resulting in some computational savings. This would be desirable for the third trajectory generator in particular, since the inverse kinematics are involved. A factor of two has been included in these estimates to allow for overhead, and the estimates are derived primarily from [61]. Note that the computational cost for the second trajectory planner is independent of the number of manipulator joints, since both the input and the output of the algorithm are in Cartesian space.

For the above simple cases, the Planning module calculates the parameters which define the manipulator position (and velocity and acceleration, in some cases) as a function of time which will perform the next desired motion. The Planning module thus looks ahead and determines the future manipulator positions for the entire duration of the planned motion segment. The planned trajectory functions are then evaluated by the Execution process for incrementally larger values of time at each cycle to produce the string of closely spaced goals for Servo. The planning and execution of the algorithms has been discussed in previous sections. The trajectory generation process for the above algorithms is very simplistic, with no replanning or explicit incorporation of manipulator or payload dynamics being performed in the Planning module; only the most basic functions required to compute trajectories.

Although these algorithms provide *much* less than the full desired capabilities, they nonetheless provide a baseline for the minimal amount of computation performed by Primitive. For these simple cases, the computational requirement (<75 K real operations per second) is well within the capabilities of current computing hardware. Depending on the actual capabilities built into the level, however, the processing requirements can be much higher. For example, the inclusion of procedures to calculate gains and full manipulator dynamics, check planned trajectories to remain within hardware constraints, and use the freedoms allowed by the position tolerance and extra manipulator degrees of freedom to optimize an objective function will all add to the number of planning operations required. Many trajectory planning algorithms which have been developed (particularly for motion that is optimal in some sense) require considerable computation, and are actually intended to be performed off line. For example, the MCTP algorithm discussed in section 6.3 may take from 1–110 seconds to compute (on a VAX 11/780 for a 3 degree of freedom arm), depending primarily on the size of the grid used in the dynamic programming optimization [51]. In the Execution process, the monitoring of the manipulator state to determine the execution status and the modification of command outputs to successfully negotiate singular configurations are examples of necessary processing which has not been specifically included in the above estimates.

10. Conclusion

The function and interfaces of the Prim level of a hierarchical manipulator control system have been described. Examples of several types of commands and different trajectory planning algorithms have been presented to show how they are accommodated by the proposed interfaces. The interfaces to the world model are difficult to specify precisely, and depend heavily on what is implemented in the Task Decomposition module. However, many desirable types of information which may be required have been identified.

11. References

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

[2] Albus, J. S., McCain, H. G., Lumia, R., NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM), NBS Technical Note 1235, July 1987.

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

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

[5] Andersson, R. L., "Agressive Trajectory Generator for a Robot Ping-Pong Player," Proc. IEEE Conf. Robotics and Automation, Philadelphia, PA, April 1988.

[6] Asada, H., Izumi, H., "Direct Teaching and Automatic Program Generation for the Hybrid Control of Robot Manipulators," Proc. IEEE Conf. Robotics and Automation, St. Louis, MO, March 1985.

[7] Baillieul. J., et al., "Kinematically Redundant Manipulators," Workshop on Space Telerobotics, Pasedena, CA, January 1987.

[8] Baker, D. R., Wampler, C. W., "Inverse Kinematics for Redundant Manipulators," GM Research Publication GMR-5478, July 1986.

[9] Blauer, M., Belanger, P. R., "State and Parameter Estimation for Robotic Manipulators Using Force Measurements," IEEE Trans. on Automatic Control, Vol. AC-32, No. 12, December 1987.

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

[11] Brady, M., et al., eds., Robot Motion: Planning and Control, MIT Press, Cambridge, MA, 1982.

[12] Brooks, R. A., "Solving the Find-Path Problem by Good Representation of Free Space," IEEE Trans. Sys., Man, and Cybernetics, Vol. SMC-13, No. 3, 1983.

[13] Bullock, D., Grossberg, S., "Neural Dynamics of Planned Arm Movements: Emergent Invariants and Speed-Accuracy Properties During Trajectory Formation," Psychological Review, Vol. 95, No. 1, 1988.

[14] Chand, S., Doty, K. L., "On-line Polynomial Trajectories for Robot Manipulators," Inter. Journal of Robotics Research, Vol. 4, No. 2, 1985.

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

[16] Dubowsky, S., Norris, M. A., Shiller, Z., "Time Optimal Trajectory Planning for Robotic Manipulators with Obstacle Avoidance: A CAD Approach," Proc. IEEE Conf. Robotics and Automation, San Francisco, CA, April 1986.

[17] Erdmann, M., "Using Backprojections for Fine Motion Planning with Uncertainty," Inter. Journal of Robotics Research, Vol. 5, No. 1, 1986.

[18] Espiau, B., Boulic, R., "Collision Avoidance for Redundant Robots with Proximity Sensors," Proc. Third Int. Symp. on Robotics Research, Gouvieux, France, October 1986.

[19] Faux, I. D., Pratt, M. J., Computational Geometry for Design and Manufacture, Halstead Press, 1981.

[20] Fiala, J., "Manipulator Servo Level Task Decomposition," NIST Technical Note 1255, NIST, Gaithersburg, MD, October 1988.

[21] Freund, E., Hoyer, H., "Collision Avoidance in Multi-Robot Systems," Second Int. Symp. on Robotics Research, Kyoto, Japan, August 1984.

[22] Hirzinger, G., "Robot Learning and Teach-in Based on Sensory Feedback," Proc. Third Int. Symp. on Robotics Research, 1986.

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

[24] Hollerbach, J. M., "Dynamic Scaling of Manipulator Trajectories," Journal of Dyn. Sys., Meas., and Control, Vol. 106, March 1984.

[25] Kant, K., Zucker, S. W., "Toward Efficient Trajectory Planning: The Path-Velocity Decomposition," Inter. Journal of Robotics Research, Vol. 5, No. 3, Fall 1986.

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

[27] Kim, B. K., Shin, K. G., "Minium-Time Path Planning for Robot Arms and Their Dynamics," IEEE Trans. Sys., Man, and Cybernetics, Vol. SMC-15, No. 2, March/April 1985.

[28] Klein, C. A., Huang, C., "Review of Pseudoinverse Control for Use with Kinematically Redundant Manipulators," IEEE Trans. on Sys., Man, and Cybernetics, Vol. SMC-13, No. 3, March/April 1983.

[29] Leake, S., "Control System Architecture for Telemanipulator Operation," in Recent Trends in Robotics: Modeling, Control, and Education, M. Jamshidi, J. Y. S. Luh, M. Shahinpoor, eds., North-Holland, New York, NY, 1986.

[30] Lin, C. S., Chang, P. R., "Joint Trajectories of Mechanical Manipulators for Cartesian

Path Approximation," IEEE Trans. Sys., Man., and Cybernetics, Vol. SMC-13, No. 6, 1983.

[31] Lin, C. S., Chang, P. R., Luh, J. Y. S., "Formulation and Optimization of Cubic Joint Trajectories for Industrial Robots," IEEE Trans. Automatic Control, Vol. AC-28, No. 12, 1983.

[32] Lozano-Perez, T., Mason, M. T., Taylor, R. H., "Automatic Synthesis of Fine-Motion Strategies for Robots," Int. Symp. on Robotics Research, 1983.

[33] Lozano-Perez, T., "A Simple Motion-Planning Algorithm for General Robot Manipulators," IEEE Journal of Robotics and Automation, Vol. RA-3, No. 3, June 1987.

[34] Luh, J. Y. S., Lin, C. S., "Approximate Joint Trajectories for Control of Industrial Robots Along Cartesian Paths," IEEE Trans. Sys., Man, and Cybernetics, Vol. SMC-14, No. 3, May/June 1984.

[35] Maciejewski, A. A., Klein, C. A., "Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments," Inter. Journal of Robotics Research, Vol. 4, No. 3, Fall 1985.

[36] Mason, M. T., "Compliance and Force Control for Computer Controlled Manipulators," IEEE Trans. Sys., Man, and Cybernetics, Vol. SMC-11, No. 6, June 1981.

[37] Mason, M. T., "Automatic Planning of Fine Motions: Correctness and Completeness," Proc. IEEE Conf. Robotics and Automation, Atlanta, GA, March 1984.

[38] Merlet, J-P., "C-surface applied to the design of an Hybrid Force-Position Robot Controller," Proc. IEEE Int. Conf. on Robotics and Automation, Raleigh, NC, March 1987.

[39] Nakamura, Y., Hanafusa, H., Yoshikawa, T., "Task Priority Based Redundancy Control of Robot Manipulators," Inter. Journal of Robotics Research, Vol. 6, No. 2, Summer 1987.

[40] Newman, W. S., Hogan, N., "High Speed Robot Control and Obstacle Avoidance Using Dynamic Potential Functions," Proc. IEEE Int. Conf. on Robotics and Automation, Raleigh, NC, March 1987.

[41] Norcross, R. J., Wang, J. C., McInnis, B. C., Shieh, L. S., "Pole Placement Methods for Multivariable Control of Robotic Manipulators," Journal of Dynamic Systems, Measurement, and Control, December 1986.

[42] Paul, R. P., "Manipulator Path Control," Proc. Inter. Conf. on Cybernetics and Society, San Francisco, CA, September 1975.

[43] Paul, R. P. C., "Manipulator Cartesian Path Control," IEEE Trans. Sys., Man, and Cybernetics, Vol. SMC-9, No. , 1979.

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

[45] Paul, R. P., Zhang, H., "Robot Motion Trajectory Specification and Generation," Second Inter. Symp. on Robotics Research, 1984.

[46] Pfeiffer, F., Johanni, R. "A Concept for Manipulator Trajectory Planning," IEEE Journal of Robotics and Automation, Vol. RA-3, No. 2, April 1987.

[47] Raibert, M. H., Craig, J. J., "Hybrid Position/Force Control of Manipulators," Transactions of the ASME, Vol. 102, June 1981.

[48] Sahar, G., Hollerbach, J. M., "Planning of Minimum-Time Trajectories for Robot Arms," Inter. Journal of Robotics Research, Vol. 5, No. 3, 1986.

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

[50] Salisbury, J. K., "Interpretation of Contact Geometries from Force Measurements," Int. Symp. on Robotics Research, 1984.

[51] Shin, K. G., McKay, N. D., "A Dynamic Programming Approach to Trajectory Planning of Robotic Manipulators," IEEE Trans. on Automatic Control, Vol AC-30, No. 6, June 1986.

[52] Shin, K. G., McKay, N. D., "Selection of Near-Minimum Time Geometric Paths for Robotic Manipulators," IEEE Trans. on Automatic Control, Vol. AC-30, No. 6, June 1986.

[53] Suh, K. C., Hollerbach, J. M., "Local versus Global Torque Optimization of Redundant Manipulators," Proc. IEEE Int. Conf. on Robotics and Automation, Raleigh, NC, March 1987.

[54] Suh, S. H., Bishop, A. B., "The Tube Concept and its Application to the Obstacle-Avoiding Minimum-Time Trajectory Planning Problem," Proc. IEEE Int. Conf. on Sys., Man, and Cybernetics, Alexandria, VA, October 1987.

[55] Taylor, R. H., "Planning and Execution of Straight Line Manipulator Trajectories," IBM J. Res. Develop., Vol. 23, No. 4, 1979.

[56] Weiss, L. E, Sanderson, A. C., Neuman, C. P., "Dynamic Sensor-Based Control of Robots with Visual Feedback," IEEE Journal of Robotics and Automation, Vol RA-3, No. 5, October 1987.

[57] Whitney, D. E., "Force Feedback of Manipulator Fine Motions," Journal of Dynamic Sys., Meas., and Control, December 1982.

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

[59] Will, P. M., Grossman, D. D., "An Experimental System for Computer Controlled Mechanical Assembly," IEEE Trans. on Computers, Vol. C-24, No. 9, September 1975.

[60] Yoshikawa, T., "Analysis and Control of Robot Manipulators with Redundancy," First Int. Symp. on Robotics Research, 1983.

[61] Zhang, Y., Paul, R., "Robot Manipulator Control and Computational Cost," report produced for the National Bureau of Standards under Department of Commerce Grant No. 60NANB7D0749, 1988.