A Structured Framework for the Control of Industrial Manipulators

KANG G. SHIN, SENIOR MEMBER, IEEE, AND STUART B. MALIN, MEMBER IEEE

Abstract—**A framework for the intelligent control of industrial manipulators is described. The control structure that is described is designed to take full advantage of the improved sensing capabilities and more powerful languages that have become, or are now becoming, available. The framework is a hierarchical structure in which the aspects of motion control are isolated in independent layers of the organization. A scheme for control (which is conceptually replicated at each level of the structure) is introduced that is nontraditional in orientation; this control scheme is specifically intended to operate in the framework presented. The control scheme is not strictly organized, but rather achieves the some pose in response to both the intended goal and the real-world conditions. Two adjacent layers of the hierarchy are discussed in detail. These layers are capable of providing noncompliant control of a multijointed manipulator. It is indicated where other aspects of intelligent machine control are located in this structure, but there are no details presented regarding their operating specifics such as vision, taction, and collision avoidance. Of the two layers, the low-level mechanism is responsible for servoing a single joint independently. The individual low-level controllers are then integrated and coordinated by the higher level controller. Because of the nature of the flexible control scheme, the control is adaptive to manipulator dynamics without explicit modeling.**

I. Introduction

An INTELLIGENT ROBOT should be capable of performing a variety of assigned tasks, be aware of its environment, and be able to effectively respond to unexpected events. Therefore, it will be able to accommodate for misalignment of parts in the workspace, perform tasks described in an abstract manner, and be capable of fine detail work. In order to have the preceding capabilities, an intelligent robot will require visual and/or tactile sensing and interpretation, goal-seeking task executors, collision avoiding path planners, and a versatile manipulator control structure. Though last on the list, the flexible control structure is that which will enable the others to effectively interface with the manipulator.

In this paper we present a structured *framework* for this flexible control system. It is felt that such control techniques will enable robotic systems to take full advantage of the improved sensing capabilities and more powerful task-oriented languages that are now beginning to emerge. These two components (i.e., sensing and languages) are placing an unprecedented demand on the control subsystem, namely, that the controller be able to react quickly to a series of changing directives.

These developments have made it necessary to reexamine the organization of the robot control and computing structures. It has become quite clear that a hierarchical organization is needed, at least in defining the logical structure of the system as shown in [1], [2], [9], [13]. In a hierarchical system the information available to, and used by, a particular level is unique to the function performed on that level. For example, in a high-level task-oriented language, objects are referred to with nouns that represent the object (e.g., bracket, screw, baseplate), are manipulated by action verbs (e.g., attach, procure, insert, tighten) with possible modifiers (e.g., smoothly, quickly), and are subject to constraints (e.g., until finger-tight). This contrasts sharply with the lower level language, where objects are described by mathematical frames and manipulated via transformations, and where constraints must be carefully defined and properly effected. And on yet a lower logical level, motion is carried out in an n-dimensional mathematical space for an n-jointed robot, specified by joint servo rates, and subject to the physical forces/torques of moving components.

The control of a manipulator has long been regarded difficult because of the nonlinearity and joint couplings in its dynamics. To circumnavigate this difficulty, the manipulator control problem is usually divided into *trajectory or path planning* and *trajectory* or *path control*. Trajectory planning is concerned with the calculation of the timing of joint position and velocity from a geometric path supplied by a *task planner*. This is done off-line normally by minimizing total traveling time subject to certain constraints (e.g., constraints on accelerations [5], [10], or those on input torques [3], [15]).

The prime task of path control is to generate the individual joint motions needed to move from a place to the desired destination following a path specified by the trajectory planner. There are three well-known methods of path control that are all kinematically oriented. These are "resolved motion position control" [10], "resolved motion rate control" [17], and "resolved motion acceleration control" [4]. See [7] for an excellent survey of work done thus far in the area of manipulator control.

Note that most conventional works deal with only *specific subproblems* of the manipulator control and planning

Manuscript received October 13, 1983; manuscript revised August 1984. This work was supported in part by the US AFOSR under Contract F49620-82-C-0089 and by the New York State Science and Technology Foundation under Grant SSF 81-3.

K. G. Shin is with the Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109.

S. B. Malin is with the Advanced Manufacturing Systems, IBM Corporation, Boca Raton, FL 33432.

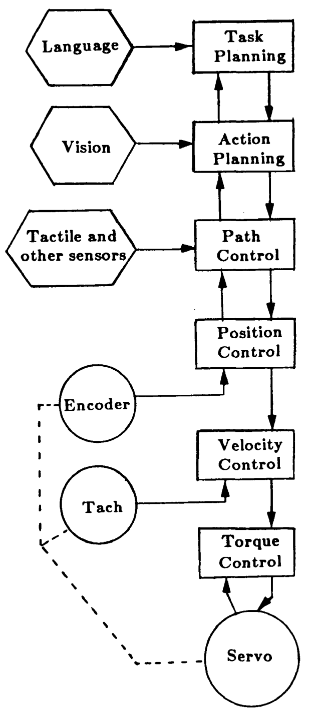

Fig. 1. Hierarchical structure of a robot control system.

without considering their integration. For a few examples, see [11] for robot kinematics, [8] and [12] for compliant control, [16] for planning of Cartesian straight-line trajectories, myriad publications elsewhere for robot vision, etc. Unlike the conventional works, this paper considers a structured framework for assembling these control and planning submodules into a flexible, powerful system. Consequently, complex mathematics are intentionally avoided in the discussion that follows. It should be noted that the main contribution of this paper is to provide a *robot control framework* that is flexible, powerful enough to accommodate current and future solutions to almost all specific robot control and planning problems.[1]

This paper is organized top-down. In Section II the manipulator control system is decomposed into a hierarchical structure. Section III deals with both the concepts of high- and low-level motions. Sections IV and V discuss the noncompliant controls of high- and low-level motions, respectively. The paper concludes with Section VI.

II. Hierarchy in the Manipulator Control System

A flexible manipulator control system is organized as a hierarchical framework. The levels of the hierarchy are cleanly divided: information processed at a particular level is not directly available to other levels of the structure.[2] There are two paths of information flow: *upward* and *downward*. Downward moving data represents the flow of command; a level may issue commands only to the level immediately below. Upward moving data comprises the flow of feedback information; the feedback-based control of a level is closed in the level immediately above. Information is abstracted as it flows upwards through the hierarchy; more physically specific information is processed in the lower levels of the hierarchy. Each level filters and transforms the data it receives producing a more abstract representation for further upward flow. Fig. 1 schematically depicts the levels and information flows of the hierarchy.

The lowest level is a (*joint motor*) *torque controller*, which generates a drive current for the corresponding joint and receives feedback regarding the motor torque. Above this is the *velocity control* level, which is responsible for specifying a desired servo rate and employs tachometric feedback for velocity stabilization. Both the torque (or acceleration) and velocity controllers are implemented in a single hardware feedback system called a *single axis interface*.

Above the single axis interface is a *position controller*. The position controller generates velocity requests that are then issued to its subordinates, the single axis interfaces. It receives position feedback information from a shaft angle encoder.[3] Control of a single axis is performed by a software process called a *single axis control element* (SACE);[4] the SACE must deal with the dynamic forces/torques acting on the axis.

The individual joint position controllers are integrated into an overall structure at the next higher level. It is this level that the concept of a manipulator emerges from the separate individual joints. The major task executing at this level is called the *chasing point sphere of influence motion model* (CPSIMM). This model provides the capabilities for coordinated joint control. CPSIMM is organized to control the position and orientation of the end-effector in Cartesian space. In addition to this control function, visual/tactile sensing can be interfaced at this level.

Above this level, the hierarchy is flexible and may include other high-level systems that are responsible for the intelligent functions of the entire manufacturing systems. Note that CPSIMM supports a single environmentally sensitive manipulator, whereas the next higher levels extend this awareness to the full complement of workstation devices. Trajectory planning occurs at the lowest level of the higher (than CPSIMM) level structure and its upper level is usually a goal-seeking task planner.

III. Motion Control Concepts

The motion control strategy developed for our framework is a distributed control mechanism, that is, the control function is not centralized to a particular location but exists simultaneously in different forms in different locations. Control is not only distributed over the levels of the

[1] Solving such specific problems is not the intent of this paper and may even lead to several volumes of books. Moreover, some of these specific problems are as of now unsolved research problems, e.g., dynamic detection of obstacles and avoidance of collision.

[2] A concept widely upheld in structured programming.

[3] The position encoder is considered to reside (logically) at the position controller level, although physically it is a component of the lowest level—the manipulator itself.

[4] Typically the SACE resides within a microprocessor-based joint controller.

hierarchy, but across multiple processors existing on any particular level.

On any level of the hierarchy, the processors operate independently of their colleagues. Each processor receives commands from a coordinating processor on the level above itself, and in turn issues commands to those processors below and directly attached to itself. Organizationally, the system is a tree structure where a given node may have multiple children, but will always have only one parent.

To effect motion control, we have allocated two levels of the structure: a low level that performs joint motion on a *per-joint* basis, and a high level that performs Cartesian motion on a *per-manipulator* basis. We posit the existence of a high-level *trajectory generator* that would generate motion requests for the manipulator; this level would in turn receive input from a *task planner*.

The higher level performs motion in a *context*. It is the responsibility of the trajectory generator to select the appropriate context of motion coordination. The lower level performs motion in a *mode*. It is the responsibility of the higher level to select a motion mode for each joint controller. Each of the joint controllers contains a number of functional units called *fixed function modules*. These modules may be connected together in different ways, each form yielding different control characteristics. The low-level motion modes are achieved by different fixed function forms. The form, or mode, changes in response to sensed real-world events (and thus we refer to the system as "flexible" or "auto-reorganizing") and to commands from the higher level.

In general this is true of every level in the hierarchy: that any processor contains a number of fixed function modules that are auto-reorganizing. The particular form determines the control characteristics as the level attempts to control its child processor(s). The form changes either by command from above, or because some situation in its children changes.

A. High-Level Concepts

The *chasing point sphere of influence motion model* is the process that allows for all of the high-level control functions. CPSIMM may be regarded as a bridge between a higher level path generator and the low-level joint motion controllers. It is responsible for coordinating multiple axes to provide uniform end-effector motion and maps the multiple joint space domains into a single Cartesian reference frame. Tactile sensing and processing capabilities may be available to CPSIMM.

The capabilities CPSIMM must provide are indicated by the needs of higher level task systems; the higher level structure generates *macroprimitives* that are the base units or steps of an assembly procedure. As such they are *primitives* with respect to the higher level, but with respect to effecting such primitives they entail much coordinated maneuvering of the lower level motion controllers, and are therefore *macro* in scope. CPSIMM must accept these macroprimitives and produce a series of *microprimitives* that are local in scope (i.e. joint specific). The lower levels execute the microprimitives.

Since CPSIMM is a high-level function, it performs motion in a context. The context determines how the motion is to be interpreted and executed. CPSIMM supports three motion contexts: *preplanned path context*, *dynamic chasing point context*, and *dynamic point injection context* (which will be discussed later in this section).

1) Chasing Point: The motion contexts all share in common the concept of a *chasing point*. The chasing point is a point in the n-dimensional joint space; it also describes a desired end-effector position (location and orientation).

All of the motion contexts operate by specifying the chasing point. The low-level controllers will cause the end-effector to align with the chasing point. High-level control of the end effector is achieved by judicious movement of the chasing point; as the end-effector approaches alignment with the chasing point, the point is moved according to the rules of the current context. As the point is moved through space, the end-effector is always seeking the most current target or chasing point. The result of this behavior is that the end-effector is tracing out a path in space—a path that is determined by the movement of the chasing point.

2) Sphere of Influence: The *sphere of influence* is a sphere whose origin is the chasing point, and is of a radius equal to the length of the end-effector. As the motion context moves the chasing point through space, it must insure that no known object in the workspace intersects the volume of the sphere. This technique is employed as the lowest level of the collision avoidance mechanism.

If the curvilinear lines are drawn between chasing point positions according to the anticipated end-effector path, the sphere may be moved along this path to provide the locus of all points passed through. The solid volume so described will be a curvilinear (i.e., sinuous) cylinder. A second order collision avoidance is obtained by insuring that the curvilinear cylinder does not contain any known objects, in whole or in part.[5]

3) Preplanned Path Context: This context is used when the end-effector must be moved through the workspace according to an *a priori* determined path at a specified speed. Traditional teach-based playback systems operate in a context similar to this.

Paths in this context are composed of a finite set of distinguishable individual path segments. These segments are contiguous and connected, although the derivative is, more often than not, discontinuous. The motion starts and stops at *end points*. The intersection of two path segments is an *intermediate point*. The velocity along each path segment is independently specifiable. The ability to provide constant velocity paths in the Cartesian domain is made possible by the independence of segment velocities.

A path is formed by a finite sequence of points in Cartesian space, which are then transformed into points in

[5]For clarity of the intent of this paper, we have not considered sophisticated obstacles such as "pseudo-obstacles" in [6].

joint space by solving inverse kinematic equations [11]. The path segment between two points in joint space is determined by the particular physical construction of the manipulator.[6] However, even this path cannot be assumed to be the actual path that will be followed. At each intermediate point the joint velocities must change to properly follow the next segment, thus requiring acceleration. If the motion is to be smooth, the position, velocity, and acceleration must all be continuous functions of time. Thus, a smooth acceleration must be performed at each intermediate point.

We define a region of space around each intermediate point, called an *ϵ-neighborhood*. All path segment transitions will occur inside of this neighborhood. When the end-effector enters the neighborhood, the joints are accelerated to the velocities needed for the next path segment. The size of the neighborhood, ϵ, is determined by the maximum amount of time that may be needed to smoothly accelerate from one velocity to the next.

The complete motion path is accomplished by targeting the next point in the sequence. The current target is used to define the chasing point. When the end-effector enters the chasing point centered ϵ-neighborhood, the next intermediate point is used to describe the new position of the chasing point. The joint controllers are instructed to accelerate to the velocity required for the new path segment.

4) Dynamic Chasing Point Context: This context allows a path to be determined in real-time. The paths are usually determined with the aid of environmental feedback such as vision, sonar, proximity or touch sensing. This context can also be used for teaching/creating a preplanned path; a joystick control can be used to move the chasing point. The end-effector will always move towards the chasing point—when the end-effector reaches the point, its motion is stopped.

When the speed of motion is high, the motion in this context is not as predictable as the motion produced by the preplanned path context because the manipulator may possibly be undergoing continually changing acceleration. When the chasing point is moving slowly, and the manipulator is moving at a low speed, fine control of the path is possible. The chasing point is always kept "just ahead" of the actual end-effector position.

5) Dynamic Point Injection Context: If a preplanned path is being traversed and an unexpected object is detected within the projected sphere of influence, then an additional set of intermediate points may be injected into the existing path stream.[7] This is accomplished by temporarily suspending the preplanned path context and entering the dynamic point injection context. Injected points are calculated one at a time. This context must circumnavigate the object and bring the end-effector back to the preplanned path. The mechanism associated with this context then instructs the preplanned path context to advance its index into the path segment table so that it may resume path traversal with correct information about the next segment to execute.

B. Low-Level Concepts

The low-level control of motion is performed with respect to the individual axes of the manipulator. At this low level the concept of a manipulator does *not* exist; low-level control is responsible for servoing a single axis in its joint space. The axes are treated individually and are independent at this level. Dynamical effects impacting an axis are compensated for by an adaptive feedback control algorithm that will be briefly discussed later in Section V (see [14] for detail). A joint (position) controller employs the algorithm to adaptively servo the axis according to a variety of contexts; these contexts are called *modes*.

The particular mode in effect is chosen by either CPSIMM or the current mode. CPSIMM selects modes in an effort to coordinate the multiple axes. An active mode may cause another mode to take control when either unexpected or specific anticipated events occur.

Briefly, there are four major modes of motion. Mode 1 (M1) attempts to bring the axis to a specified joint space coordinate in a specified amount of time. Mode 2 (M2) will accelerate the axis from its current velocity to a specified velocity. This change will be performed subject to the constraint that the acceleration will not exceed the rated joint capacity. Mode 3 (M3) will decelerate the axis to a stop. This may be accomplished by two methods. In the first method, the axis stops as quickly as possible (e.g., an emergency stop). In the second method, the axis is slowed and stopped at a specified joint coordinate. Mode 4 (M4) will maintain the axis at a joint coordinate. This is an active, dynamic process due to the effects arising from, for example, gravitational forces.

For its implementation on digital computers, joint control is performed in discrete time intervals. A real-time clock periodically interrupts the joint controller to invoke execution of the low level motion control software. When the software is invoked it schedules the current motion mode controller for execution. The mode controllers perform their particular function with the aid of several fixed function modules, some of which are discussed in Section V.

1) Mode 1 Motion: Mode 1 motion will servo the joint from its current position in joint space to a *desired target coordinate p_d*, subject to the constraint that the target coordinate will be achieved in a specified amount of time T_c, called the *time to converge*. When the axis achieves the target coordinate, it may have a nonzero velocity and/or a nonzero acceleration. Mode 1 motion is further characterized by an ϵ multiplier (EPS). In this mode a neighborhood is defined about p_d.[8] The minimum size of the

[6]For cylindrical manipulators, the segments are arcs of Archimedean spirals.

[7]The necessity of this context is obvious, but we are not implying any solution to the problem of dynamic detection of obstacles and avoidance of collision.

[8]This neighborhood is related to the ϵ-neighborhood of an intermediate point in the preplanned path context.

neighborhood ϵ_0, is the maximal distance required to stop the axis (i.e., when moving at maximum speed). The size of the neighborhood is actually determined by EPS.

$$\epsilon = \epsilon_0 * 2^{\mathrm{EPS}}. \tag{1}$$

This technique for the expansion of ϵ_0 was chosen because it allows a limited precision EPS to select a large range of ϵ-neighborhood sizes.

A mode 1 motion is completely specified by supplying it with the values of p_d, T_C, and EPS. Only EPS need not be explicitly defined; it has a default value of zero.

When mode 1 is invoked the joint controller must be informed of the high-level intent: stop when the target coordinate is achieved (i.e., CPSIMM is targeting a terminus), or motion is to be continued (i.e., CPSIMM is passing through an intermediate point). When the axis enters its ϵ-neighborhood, SACE signals CPSIMM via an interrupt, causing mode 3 to take control. Mode 3 is divided into two submodes, namely, mode 3a and mode 3b. If this is a terminus, mode 3b is started, and if this is an intermediate point, mode 3a is initiated instead.

2) Mode 2 Motion: Mode 2 is used to link one M1 motion to another. When mode 2 is invoked the joint is driven from its current velocity to a requested velocity. The time required to do this can be of two forms: a) the acceleration is performed in minimum time, that is, the maximally allowed joint acceleration will be used, or b) the joint is accelerated in a fixed amount of time. The fixed value must be greater than or equal to the smallest time that *any* acceleration may require.

The desired acceleration is determined by generating a polynomial description of velocity curve (see Section V-B for detail). This curve, and its derivative, will not have any discontinuities to prevent any jerky motion, and the derivative will never exceed the rated capacity of the joint.

The mode 2 controller configures the fixed function modules such that the velocity requests (generated by a polynomial function) are fed directly to the servo driver. Mode 2 does not monitor the actual behavior of the servo; it does not compensate for discrepancies between requested velocities and actual joint velocities. When the time for the transition expires, the mode 2 controller will issue an interrupt to signify completion, and will continue to request the velocity at the desired target velocity.

SACE executes mode 2 motion by comparing the generated velocity profile with actual joint behavior; this process is called a *behavior matching*. For the generation of the velocity profile we use the current velocity estimated from measurements of joint position. SACE also computes a time expansion index on the basis of the current and desired velocities. See Section V for more on this.

3) Mode 3 Motion: The mode 3 controller will servo the joint to a halt from its current velocity. The time required for the deceleration (to a stop) is determined by the upper limit of the integral of the velocity profile, that is, M3 will attempt to bring the axis to a stop at the target coordinate. Mode 3 has two submodes designated by M3a and M3b. The former is used when passing near an intermediate point, whereas the latter is used when targeting terminal points. That is, M3a does not require the end-effector to stop exactly at the target, whereas M3b does.

In M3a the controller will bring the axis to a stopped state in the vicinity of the target point. When the axis is stopped, an interrupt is generated to indicate this, and a modified variant of the mode 4 controller (called M4M) is invoked to maintain the axis at its current resting position.

In M3b the controller will terminate the mode when a certain small velocity v_s is achieved. When this occurs mode 4 is initiated to maintain position at the final target point. In the event that the current axis position is not the target point, the smallness of v_s allows M4 to bring the axis to the target point without overshoot. The completion interrupt is *not* generated.

Both M3a and M3b share in common the characteristic that they will bring the axis motion to a halt as close to the target point as is possible. To effect such motion SACE uses a polynomial-based velocity profile to calculate a trajectory. M3b differs from M3a in that the controller monitors the actual velocity. When the velocity is less than some preset limit v_s, control is transferred to mode 4; M4 is instructed to bring the axis to the target point.

4) Mode 4 Motion: The mode 4 controller will maintain the axis at a particular position. The mode operates by comparing the current position p_a to the desired position p_d, generating an error signal, and maps this error signal into a velocity request. The velocity request is used for subsequent actualization.

Mode 4 may be invoked in two ways. In the first, the standard operating procedure, the controller will bring the axis to the target point from wherever it currently is. Because M4 makes no attempt to insure the smoothness of the acceleration, it should not be relied upon for gross servoing of an axis. The modified procedure, M4M, will set the target point equal to the value of the current coordinate at the time M4M is invoked. This procedure, used to keep the axis at its current position, should only be used when the velocity is small; otherwise, the axis may be driven at an exceedingly excessive acceleration.

The error, $p_d - p_a$, will be used by the mode 4 controller to generate a velocity request that will bring the axis towards the target coordinate. The controller can be instructed to generate an interrupt when the target coordinate is first achieved. For M4M the first instance of this (when first invoked) is not considered an achievement of the target coordinate—it will generate an interrupt at the first occurrence of $p_a = p_d$ *after* the mode is initiated.

A special variant of mode 4 called M4D (dynamic) is available for use when the high-level processes are dynamically moving the end-effector through space. In M4D the convergence of the approach to the terminus can be specified. M4D will generate an interrupt when the axis first enters the ϵ-neighborhood, but it will not interrupt when $p_a = p_d$.

IV. High-Level Motion Control

The motion control concepts as developed in the previous section are designed to support each other in such a way that the versatility and capabilities of the manipulator control system are maximized. The details that are specific to the high-level motion controller (CPSIMM) are those that relate to the appearance of end-effector paths. The details that concern the low-level (SACE) solely are those that involve a single axis only.

The constituent components of CPSIMM are those that support the motion control contexts. These contexts schedule the individual axis controllers to perform various low-level control modes. The arrangement of low-level modes in a time-sequenced pattern allows a high-level motion pattern to appear.

Similarly, SACE organizes the available modules into a structure that will actualize the desired modal motion. The SACE system is self-organizing, that is, it is capable of structuring the data flow paths between its constituent modules.

The high-level concept of motion is divided into two styles: planned and dynamic paths. Although these two are quite different in temperament, they are very similar in that they achieve motion control with the same repertoire of available low-level motion modes.

Planned paths are defined by a set of points which roughly describe the path. These paths may be examined from two viewpoints: *description* and *actualization*. The description will require the development of the notion of a *simultaneously convergent* path in joint space. The actualization viewpoint centers on the development of *descriptor nodes*; these nodes are information packets describing the path segments.

Dynamic paths are constructed according to environmental and other external dictates. Dynamic paths are far more subtle than planned paths; effective execution of a dynamic path is predicated on the availability of *side information*, information that is deducible from the nature of the task. This side information will imply an approach to configuring the low-level modes.

A. Preplanned Paths

1) Path Description: A path begins at a *starting point*, moves through a series of *intermediate points*, and concludes at a *terminus*. The motion is initiated by targeting the first intermediate point. The data concerning a path segment is contained in a descriptor node associated with the intermediate point. When the segment is completed, the next segment's node is accessed to provide the data needed to continue the motion. Motion does not stop at each intermediate point; motion halts only at the terminus.

A node contains several sets of data: a set is required for each axis involved with the motion. Each set contains a desired target coordinate p_d, and ϵ multiplier (EPS), and a time to converge, T_C.

These parameters are derived from two vectors associated with each point. The first vector, P, identifies the position of the robot's wrist in Cartesian (robot) space. The second vector, O, describes the orientation of the end-effector at that point. In coordinating motion the two sets of degrees-of-freedom comprising P and O are treated independently. Orientation may be controlled along each path segment (as is required when the orientation must be held constant in Cartesian space), it may be controlled independently of, but simultaneously with, the position control of the path so that the desired orientation is achieved when the terminus is reached, or it may be brought into alignment after the final position is achieved.

In transit orientation is not performed in the following treatise, its development is a logical extension of the model described.

Assume that the current position of the hand in robot space is (X_i, Y_i, Z_i) corresponding to some joint coordinate $(\xi_i, \theta_i, \zeta_i)$ and the position of the current target point is (X_f, Y_f, Z_f) corresponding to some joint coordinate $(\xi_f, \theta_f, \zeta_f)$. Each axis must change its joint space position by an amount that is the difference between the target coordinate value and the current value, namely, $\Delta\xi \equiv \xi_f - \xi_i$, $\Delta\theta \equiv \theta_f - \theta_i$, and $\Delta\zeta \equiv \zeta_f - \zeta_i$. Since the maximum allowed joint velocity for each axis is constrained by the hardware, and thus known (i.e., $v_\xi^{\max}, v_\theta^{\max}, v_\zeta^{\max}$),[9] then the minimum possible time for each axis is to reach its destination is in degenerate form, for we are neglecting acceleration as a first-order approximation: $t_\xi = \Delta\xi / v_\xi^{\max}$, $t_\theta = \Delta\theta / v_\theta^{\max}$, and $t_\zeta = \Delta\zeta / v_\zeta^{\max}$. Because the use of these constants is in determining a minimum time, if the servo cannot realize this velocity in actual operation (due to payload and other dynamical effects), the actual time will be greater than the calculated minimum time, which is as it should be. Only in the no payload situation, the servo would actually be able to perform in close to calculated minimum time.

In order to account for acceleration, another approximation will be made. Because CPSIMM does not know the current axis velocities, a worst case approximation will be made to insure that the second-order time approximations will be large enough to always allow the segment motion to accommodate any required velocity changes. The maximum allowed accelerations are also predefined: $a_\xi^{\max}, a_\theta^{\max}, a_\zeta^{\max}$. Assume that the current velocity is the largest negative velocity possible, and that the next path segment will require the largest positive velocity allowed. The total velocity change is twice the allowed maximum velocity.[10] The acceleration time required to produce such a velocity change is

$$\begin{aligned} t_{a_\xi} &= 2v_\xi^{\max}/a_\xi^{\max} \\ t_{a_\theta} &= 2v_\theta^{\max}/a_\theta^{\max} \\ t_{a_\zeta} &= 2v_\zeta^{\max}/a_\zeta^{\max}. \end{aligned} \tag{2}$$

[9]By "maximum allowed joint velocity" we refer to that velocity which can be achieved by the joint in a no load situation as determined by the torque and velocity limits of the servomotor.

[10]We are here assuming that the maximal positive and negative velocities are of the same magnitude.

The second-order minimal time approximation is found by adding this acceleration time to the first-order transit time

$$t_\xi^{\min} = t_\xi + t_{a_\xi}$$
$$t_\theta^{\min} = t_\theta + t_{a_\theta}$$
$$t_\zeta^{\min} = t_\zeta + t_{a_\zeta}. \tag{3}$$

The second-order approximation is required when the segment transit times are small compared to the minimum acceleration time of (2). When either a) intermediate points are close together or b) the velocity changes between segments is not large, then a third-order approximation can be used.

The third-order approximation determines the acceleration time required between path segments. The velocity of the path segments is approximated by using the first-order time approximations. Let $t_j(i-1)$ be the first-order time approximation for the current path segment $i-1$ of axis j, and $t_j(i)$ be the first-order approximation for the upcoming path segment i. The average velocity along path segment i for axis j is found by dividing the joint path length, $\Delta j(i)$, by the first-order approximation

$$v_j(i) = \Delta j(i)/t_j(i). \tag{4}$$

The velocity change required between two adjacent path segments $i-1$, and i is

$$\Delta v_{j:i-1,i} = v_j(i) - v_j(i-1) \tag{5}$$

and the minimum time required for acceleration is

$$t_{a'_j} = \Delta v_{j:i-1,i}/a_j^{\max}. \tag{6}$$

Thus the minimum time to be allotted for traversing segment i for axis j is

$$t_j^{\min}(i) = t_j + t_{a'_j}. \tag{7}$$

Whether first-, second-, or third-order transit times are computed depends upon the nature of the motion in the context. The techniques converge at this point for further calculation. Let $t_j^{\min}$ be the minimum time value calculated for the axis j. Then, select the largest of the axis times

$$t_{\text{move}} = \max_j t_j^{\min}. \tag{8}$$

This is the minimum time required for all axes to reach their respective target coordinates; it represents the time required for a simultaneously convergent motion, and is therefore used as the time parameter of the motion, i.e., $T_C \equiv t_{\text{move}}$. This T_C parameter must be common to all axes involved in a motion segment. The total time of the motion is the sum of the segment time-to-converge values.

Lastly, the ϵ multiplier (EPS) must be specified. This multiplier controls the tolerance or accuracy to which the path actually passes through the intermediate point. An EPS value of zero requires the manipulator to pass close to the intermediate point. Larger values of EPS lessen the required "closeness."

2) Actualization: To execute the path, the preplanned path context (PPP-C) controller first employs mode 2 to bring the axis up to speed. Mode 2 interrupts to indicate that the axis is close to the desired speed. When all the axes have so interrupted, PPP-C invokes mode 1. The p_d, T_C, and EPS parameters of the next target node are sent to the axis controllers, and mode 1 is initiated. PPP-C then prepares the next parameter packet and transmits these to the SACE communication buffer area. One of the advantages, if multiple processors are used, is that several independent but related sets of calculations are being performed simultaneously. The communications between these processes are woven between the control functions.

Each low-level controller brings its axis towards its target coordinate. When an axis enters its ϵ-neighborhood, it slows down as it completes its targeting of the terminus. SACE informs CPSIMM that the ϵ-neighborhood has been entered. PPP-C waits for all of the axes involved in the path segment to enter their ϵ-neighborhood; when this condition is met CPSIMM is assured of two facts: 1) that the end-effector is sufficiently close to the intermediate point, and 2) that all axes are resynchronized in space for continued coordinated control. CPSIMM then instructs SACE to accelerate the axis to the velocity required for the next path segment using mode 2 (the data is already in the SACE communication buffer area). When up to speed, mode 1 is reinvoked using this data.

The cycle is repeated until the data for the terminus is loaded into the SACE buffer area. After this motion is initiated, PPP-C, waits for the set of interrupts to indicate that all ϵ-neighborhoods have been entered. Normally the axes are slowing down to target the terminus and the controller is bringing the velocity to zero using M3b. However, when the terminus is the current node, the SACE controllers are instructed to use M3a instead. This will automatically invoke M4 when the controller slows the axis sufficiently. When M4 brings the axis to a stop on the terminus, it interrupts CPSIMM. When all the interrupts are received, PPP-C has completed the preplanned path.

B. Dynamic Path Modification

The details of the techniques that properly execute the dynamic point injection context (DPI-C) are not contained within the scope of this paper;[11] but insight into the need for this capability has led to its inclusion here, so that we may illustrate how such function can be incorporated into the structure.

Two techniques predominate for the dynamic inclusion of points extraneous to a preplanned path. In the first technique, the axis is servoed as quickly as is possible towards a target point; the path is not determined. In the second technique, the injected point determines a path.

The first technique is used to avoid a collision with an obstacle: an intermediate target point in space is determined that is both far from the interfering object and that does not significantly deviate from the current, pre-

[11]Dynamic path calculation is a subject related to dynamic collision avoidance; it is not the intent of the present paper to solve this problem, nor implies that we have solved that problem.

planned path. In this way the controller need not fight the axis inertia, it must only redirect the path. Such a point can be targeted by low-level M4 motion control. Although the path is not anticipatable, the end-effector can be expected to move toward the injected target point with a speed that is inversely proportional to the distance, i.e., the farther away the point, the faster the axis moves towards it, slowing down as the point is approached.

The second technique is used once the end-effector is near the dynamically injected point. A high-level dynamic path planner generates a new preplanned path that merges with the old path, avoiding the obstacle. This dynamic path can be calculated in transit, and is controlled by the tenets of the PPP-C context.

C. *Dynamically Created Paths*

The dynamic chasing point context (DCP-C) allows the manipulators to be servoed in real-time according to environmentally determined information. In this context, the axes are servoed using a variant of the low-level M4D controller.

Although dynamically determined path control is extremely difficult and is also beyond the scope of this work, the general concept is as follows. In M4D, the controller brings the axis to the target point with a moderate velocity. The diminution of employed velocity is specified to M4D by the setting of a parameter.[12] Unlike the normal M4 operation, an ϵ-neighborhood is employed. When the end-effector enters the ϵ-neighborhood about the current chasing point, the point is moved. The low-level controllers, always executing M4D, will cause the end-effector to always *chase after* the point.

V. Low-Level Control

Low-level control entails servoing the axis in the joint domain. From this perspective there is no concept of a manipulator, only a single highly nonlinear servo positioning system. The dynamical effects manifest at this level, and must be handled appropriately.

The axis control is accomplished via feedback and is closed at this level. SACE, a software process executing in this level microprocessor, is responsible for the control function. SACE performs motion control by scheduling a variety of modular components into configuration suitable for effecting the style of motion called for.

The configured set of modules is conceptually described as an adaptive feedback control algorithm. The algorithm is a primitive learning system that attempts to dynamically ascertain the axis' motion behavior. It adapts to the perceived axis forces; it compensates not only for the ever-changing dynamics, but allows motions to be defined without requiring specification of operating speed and payload (see [14] for detail).

SACE can configure the adaptive algorithm for the execution of four motion modes. By appropriately selecting sequences of motion modes, the high-level CPSIMM achieves coordinated end-effector motion.

A discrete-time motion model is used to guide the servoing of an axis as a linear function of time. The model computes the velocity needed to bring the axis from its current joint position to the target point in exactly the amount of time T_C. This velocity is not constant because the changing dynamical effects impact actual axis motion.

We have to estimate the axis velocity on the basis of a known position history, because 1) axis velocity information is required by the other modules, and 2) the hierarchical organization presents the joint controller with position information only.

For the adaptive control algorithm, we use estimated axis response to condition future control. That is, it adjusts the velocity requests generated by the motion mode controllers into a form that will cause the axis to actually perform as requested.

The velocity profile described by a third-order polynomial is employed to guide a motion that smoothly changes the joint servo rate to achieve some requested axis velocity. The profile imposes boundary conditions to create a smooth blending function. It then expands this function in the time domain so the axis will be able to perform the desired motion; however, this method does not generate time-constrained velocity profiles. A modified version of the above will bring the axis from an initial velocity to rest while attempting to bring the axis as close to the target point as is possible.

The difference between the axis' real position and the desired position will be used to servo the axis in such a way that the axis is brought to the target point *without* overshoot.

In the following discussions, the modular components of SACE are discussed in detail.

A. *Discrete-Time Motion Model (DTMM)*

The DTMM is an interrupt driven software process that generates velocity requests for a single joint.[13] The DTMM goal is to bring the joint to the desired target coordinate in a specified amount of time i.e., the time to converge T_C.

Fig. 2 shows a trajectory for an axis at some joint position x_0 at time zero, terminating at the target coordinate p_d at time T_C. The horizontal time axis has been subdivided into n elements of width δt, where δt is the iteration period of the interrupting real-time clock:

$$n = \frac{T_C}{\delta t}. \tag{9}$$

The average amount of motion required per iteration is

$$\Delta x = \frac{(p_d - x_0)}{n}. \tag{10}$$

[12]This parameter k affects the damping of the control algorithm and will be discussed in Section V.

[13]A nonmaskable interrupt from the real-time clock causes SACE to schedule the DTMM when a motion mode using it is active.

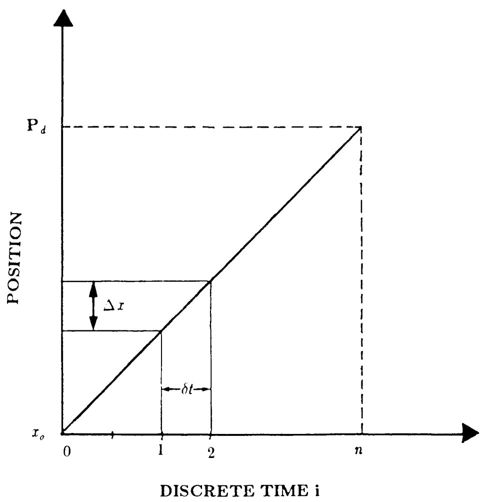

Fig. 2. An idealized discrete-time motion model.

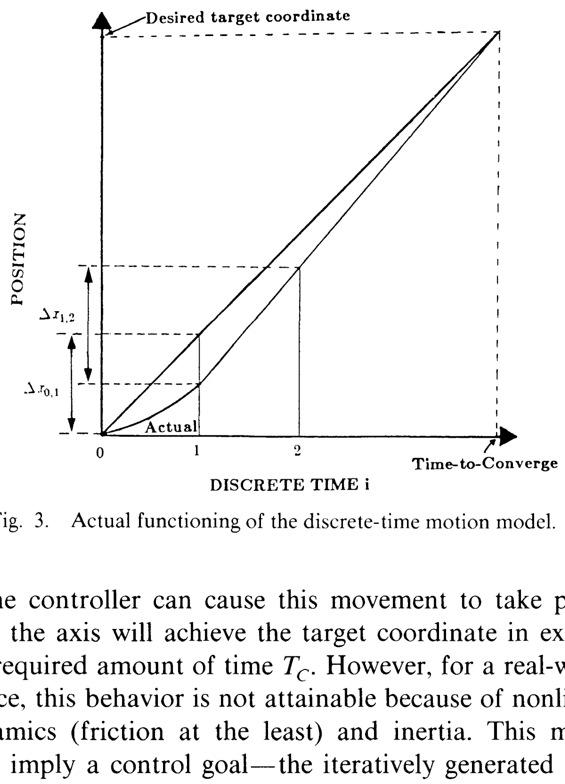

Fig. 3. Actual functioning of the discrete-time motion model.

If the controller can cause this movement to take place, then the axis will achieve the target coordinate in exactly the required amount of time T_C. However, for a real-world device, this behavior is not attainable because of nonlinear dynamics (friction at the least) and inertia. This model does imply a control goal—the iteratively generated solution for a time constrained velocity profile. We now modify this idealized model so that it compensates for the nonideal performance of actual moving systems.

If we let the DTMM calculate anew the trajectory of Fig. 2 at each iteration period, it will generate a sequence of velocity requests $\Delta x_{i,i+1}$, where i ranges from zero to $n-1$. That is, at each interrupt a new velocity request will be calculated based upon the remaining distance and time to travel.

In Fig. 3 we illustrate a plausible response to the initial velocity request calculated by (10). At the next interrupt a new linear path is constructed, the slope of which is the desired velocity. Let p_a denote the actual joint position at time i, then the velocity request for this iteration is

$$\Delta x_{i,i+1} = (p_d - p_a)/(n-i). \tag{11}$$

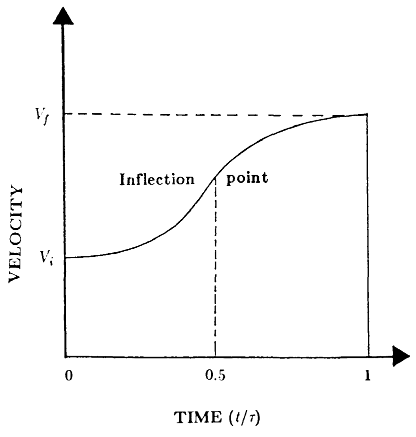

Fig. 4. A generic velocity profile curve.

Equation (11) can be used at each interrupt to generate the velocity request. One effect of this technique is a velocity creep that accrues until the axis attains its average operating velocity. The terminal velocity that is reached is

$$v_t = \lim_{i \to n-1} \Delta x_{i,i+1}. \tag{12}$$

Because of this velocity creep, an axis should never be requested to move at its maximum allowed velocity (doing so will invalidate the guarantee of a simultaneously convergent motion path).

B. Generation of Velocity Profile

The velocity profile used by the mode 2 controller is determined by a polynomial function describing smooth transitions from a given initial velocity to a specified desired velocity. Since the polynomial function represents a velocity profile in the time domain, its derivative represents acceleration. In general, a third-order polynomial is sufficient for smooth transitions. The polynomial can then be specified by four boundary conditions: the initial and final values of both the polynomial and its derivative.

Since we are concerned with one complete move, the initial and final acceleration are assumed to be zero. A generic velocity profile generated is illustrated in Fig. 4. The general form of an equation describing such a curve is

$$V = C_3\eta^3 + C_2\eta^2 + C_1\eta + C_0 \tag{13}$$

where the C_i are constants to be determined. The derivative of (13) is the acceleration curve, and is represented by

$$A = 3C_3\eta^2 + 2C_2\eta + C_1. \tag{14}$$

At the start of a blend ($\eta = 0$), the velocity is V_i and the acceleration is assumed to be zero. This provides two boundary conditions $V|_{\eta=0} = V_i$, $A|_{\eta=0} = 0$. At the end of the blend ($\eta = 1$), the velocity should be the specified velocity V_f and the acceleration should be zero. This yields two additional boundary conditions: $V|_{\eta=1} = V_f$, $A|_{\eta=1} = 0$. From these boundary conditions one can obtain coefficients $C_0 = V_i$, $C_1 = 0$, $C_2 = 3(V_f - V_i)$, and $C_3 = -2(V_f - V_i)$.

The polynomial must be mapped to real time. To do this, (13) can be written as

$$V = C_3\xi^3 + C_2\xi^2 + C_1\xi + C_0 \tag{15}$$

where $\xi = t/\tau$, τ represents the total time used in performing the velocity blending, and $t \in [0, \tau]$ is real time. The value of τ is found by insuring that the maximum acceleration is not required to exceed the rated maximal acceleration for the axis (i.e., $A_j^{\max}$ for the jth joint). The acceleration is

$$A = \left[3C_3\xi^2 + 2C_2\xi + C_1\right]/\tau. \tag{16}$$

The maximum acceleration occurs when the derivative of this is zero. This occurs when $\xi = 0.5$, i.e., $t = \tau/2$. The maximum acceleration is

$$A_{\max} = A|_{\xi=0.5} = [3C_3/4 + C_2]/\tau = 3(V_f - V_i)/2\tau. \tag{17}$$

To find the minimum time $\tau_{\min}$ for joint j such that the profile will always require the maximally allowed joint acceleration,[14] $A_j^{\max}$, without exceeding it, solve for τ when $A_{\max} = A_j^{\max}$

$$\tau_{\min} = 3(V_f - V_i)/2A_j^{\max}. \tag{18}$$

That the acceleration will never be required to be exceeded is guaranteed by the fact that the time computed here is for minimum time. If the servo cannot realize this acceleration in actual operation (due to payload for example), the actual time will be greater than the calculated minimum time.

The largest such minimum time for some joint j occurs when the initial and final velocities are opposite in sign, and are the largest permissible velocities magnitudewise, that is, a complete velocity turnaround:

$$\tau'_{\min\max} = 3V_j^{\max}/A_j^{\max}. \tag{19}$$

For the mode 3b controller we need also smooth transitions from an initial velocity to zero velocity such that a specific distance is traversed. This can be achieved similarly to the above using a polynomial of the same form as (13). Because the desired velocity V_f is zero, the coefficients of (13) are $C_0 = V_i$, $C_1 = 0$, $C_2 = -3V_i$, $C_3 = 2V_i$. Substituting these into (13) yields the polynomial

$$V = 2V_i\eta^3 - 3V_i\eta^2 + V_i \tag{20}$$

and is mapped to real time as

$$V = V_i\left[2(t/\tau)^3 - 3(t/\tau)^2 + 1\right]. \tag{21}$$

The polynomial generates a velocity curve of the form as shown in Fig. 5. Using the curve, we can calculate the distance traversed s to be $V_i\tau/2$. It is desired that this distance be the size of the ϵ-neighborhood, thus $\epsilon = V_i\tau/2$ from which the time of motion is found $\tau = 2\epsilon/V_i$. The acceleration is given by the derivative of (21):

[14] By "maximum allowed joint acceleration" we refer to that acceleration which can be achieved by the joint in a no load situation as determined by the physical joint torque motor limits.

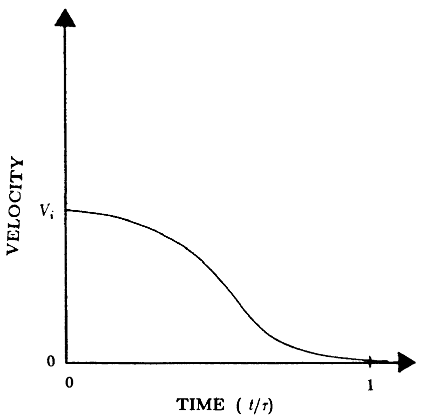

Fig. 5. A generic velocity curve used for mode 3 control.

$$A = V_i\left[6(t/\tau)^2 - 6(t/\tau)\right]/\tau. \tag{22}$$

The maximum acceleration occurs at $t = \tau/2$, and is

$$A_{\max} = -3V_i/2\tau. \tag{23}$$

The time required for the transition can be found

$$\tau = 3V_i/2A_{\max}. \tag{24}$$

Minimum time occurs when maximally allowed acceleration is used, i.e., $\tau_{\min} = 3V_i/2A_j^{\max}$, and the upper bound is found when the axis is at the maximum velocity: $\tau_{\min\max} = 3V_j^{\max}/2A_j^{\max}$. The minimum size ϵ-neighborhood is found from this by $\epsilon_0 = 0.5V_i\ \tau_{\min\max} = 3\{V_j^{\max}\}^2/4A_j^{\max}$. Also one can obtain the distance required in stopping the axis:

$$s = 3V_i^2/2A_{\max}. \tag{25}$$

This implies that for $V_i < V_j^{\max}$, the distance traversed in stopping the axis s is always less than ϵ_0.

C. Velocity Estimation

We need to estimate the current axis velocity from a position history. The position of the axis is determined by SACE at each real-time clock tick interrupt. The current and last l values are saved. The current velocity is determined as a function of these $l + 1$ values. Because the axis velocity is always changing, it is not necessary for l to be large, in fact, a large l would infuse the velocity estimate with inappropriate data; the velocity estimator is a short-term memory function.

Fig. 6 illustrates a position history with $l = 2$. Let x denote the joint coordinate and i be the current time (clock tick number). The change in position between two consecutive known positions is

$$\Delta x_{i-1,i} = x_i - x_{i-1}. \tag{26}$$

The change in position is also given by the average velocity for the segment

$$\Delta x_{i-1,i} = (V_i + V_{i-1})/2. \tag{27}$$

Solving (26) and (27) for the current velocity yields

$$V_i = 2\Delta x_{i-1,i} - V_{i-1}. \tag{28}$$

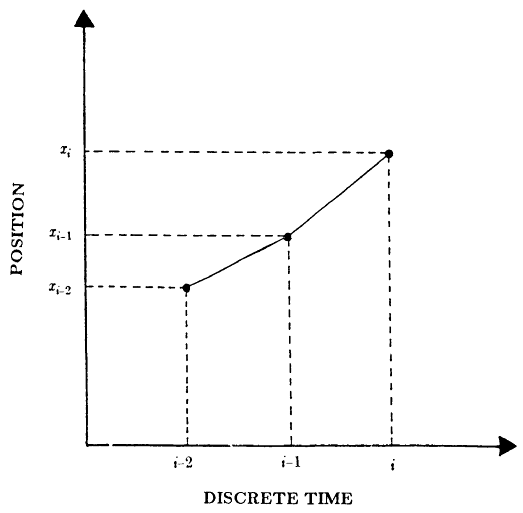

Fig. 6. Position history used for velocity estimation.

The velocity at time $i - 1$ is approximated by

$$\langle V_{i-1} \rangle = (x_i - x_{i-2})/2 \tag{29}$$

where the angle brackets indicate an estimate. Substituting (29) into (28) and then simplifying the result, we get

$$\langle V_i \rangle = 3x_i/2 - 2x_{i-1} - x_{i-2}/2. \tag{30}$$

D. Discrete Error Corrector

The discrete error corrector moves the axis with a velocity that is proportional to the distance remaining (i.e., the error). The distance remaining at some interrupt-invoked iteration is $x_r = p_d - p_a$. In order to reduce the propensity for overshoot, and to provide a mechanism for controlling the rate of convergence, the selection of velocity request $\Delta x_{i,i+1}$ is quantized. An integer valued parameter k specifies the quantization.[15] All velocity requests in the range of 1 to k are mapped to 1, and those in the range $k + 1$ to $2k$ are forced to be 2, etc. The ranging is performed by

$$\Delta x_{i,i+1} = \lfloor (x_r + (k - 1)\operatorname{sgn}(x_r))/k \rfloor \tag{31}$$

where $\Delta x_{i,i+1}$ is the desired velocity for the upcoming iteration period, and sgn is the sign function.[16] This function produces the ranging

$$\Delta x_{i,i+1} = \begin{cases} 0, & \text{if } x_r = 0 \\ j, & \text{if } x_r \neq 0 \end{cases} \tag{32}$$

where j is the integer that satisfies

$$k * (j - 1) + 1 < x_r \leqslant k * j. \tag{33}$$

The k parameter affects speed performance: an increase in k slows the rate of convergence. This effect is illustrated in Fig. 7.[17] It is important to note that the introduction of such k eliminates undesirable effects, e.g., overshoots, of a pure proportional control. This is an interesting departure from the conventional controls such as PID control.

E. Behavior Matching

This module attempts to ascertain the effects imposed on the axis by the dynamical forces. It estimates the disparity between the drive signal S_i at iteration i and the resultant motion. The behavior matching assumes this disparity is linear[18]

$$S_i = \alpha\Phi_i + \beta \tag{34}$$

where Φ_i is the velocity requested at iteration i, β an offset, and α a constant of proportionality.

A first-order approximation for β is obtained by assuming the offset is the difference between the last request Φ_{i-1} and the achieved velocity V_i^a

$$\beta = \Phi_{i-1} - V_i^a. \tag{35}$$

The first-order approximation for α is gotten by generating a performance index κ, which compares the change in requested velocity to the change in actual velocity

$$\kappa = (\Phi_{i-1} - \Phi_{i-2})/(V_i^a - V_{i-1}^a) \tag{36}$$

and expanding $\alpha\Phi_i$ to $\Phi_i + \kappa\Delta\Phi_{i-1,i}$ where $\Delta\Phi_{i-1,i} = \Phi_i - \Phi_{i-1}$. This procedure maps the change $\Delta\Phi_{i-1,i}$ according to the measured change arising from the previous response κ, as illustrated in Fig. 8.

To improve this, a history of performance indices κ_i are kept. A second-order polynomial is fitted to the three most recent values: κ_{i-2}, κ_{i-1}, and κ_i. An estimate of the future value $\langle \kappa_{i+1} \rangle$ is calculated.

A generic quadratic polynomial $q = C_2t^2 + C_1t + C_0$ passes through points A, B, C at times $t = 0, 0.5, 1$, respectively. This leads to coefficient values of $C_2 = 2A - 4B + 2C$, $C_1 = 4B - 3B - C$, $C_0 = A$. The next point D occurs at $t = 3/2$, and is given by $D = A - 3B + 3C$. Using this information, the estimated performance index at iteration i is

$$\langle \kappa_{i+1} \rangle = \kappa_{i-2} - 3\kappa_{i-1} + 3\kappa_i. \tag{37}$$

Employing this in place of κ leads to second-order behavior matching

$$[\Phi | V^a]_i = \Phi_i + \langle \kappa_{i+1} \rangle \Delta\Phi_{i-1,i} \tag{38}$$

where $[\cdot]_i$ indicates that this is a velocity request conditioned on measured velocity. The drive signal S_i is produced by

$$S_i = [\Phi | V^a]_i + \beta. \tag{39}$$

VI. Experimental Results and Discussion

In this paper a robot control framework, incorporating CPSIMM and SACE, has been developed in a structured fashion. The individual components of this framework has been clearly identified and their details specified.

A first-order control process, consisting of M1 and M4 only, has been implemented and its behavior examined. By

[15] k is related to the damping factor of the algorithm.

[16] It returns a value of +1, 0, or −1.

[17] During performing system test or calibration, the k parameter was selected *experimentally*. For completeness, follow-up activity can be performed to generate a heuristic that selects k based on manipulator kinematics and dynamics.

[18] Although it is not, it is suitable as an approximation.

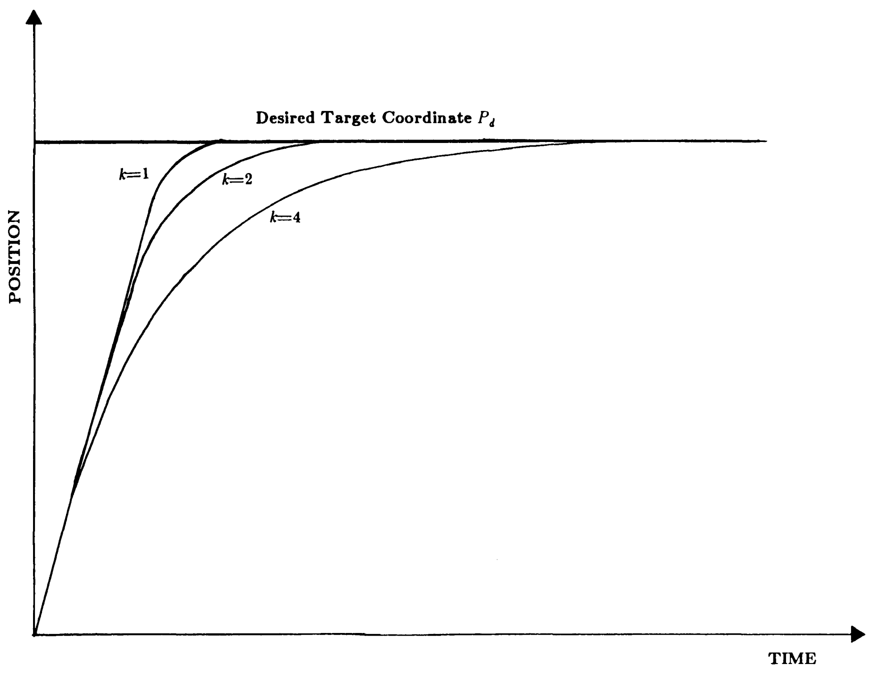

Fig. 7. Convergence of discrete error correction for mode 2 (with τ = 100 ms).

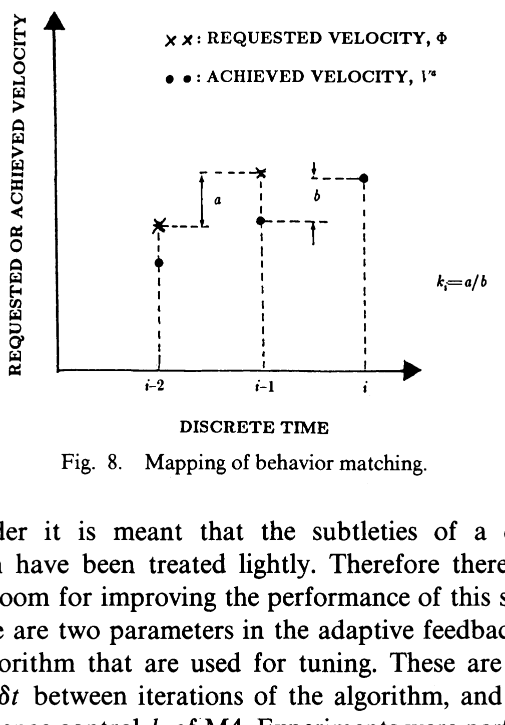
x x: REQUESTED VELOCITY, Φ
• •: ACHIEVED VELOCITY, V^a
a
b
$k_i = a/b$
REQUESTED OR ACHIEVED VELOCITY
i-2
i-1
i
DISCRETE TIME

Fig. 8. Mapping of behavior matching.

first-order it is meant that the subtleties of a quality solution have been treated lightly. Therefore there exists ample room for improving the performance of this system.

There are two parameters in the adaptive feedback control algorithm that are used for tuning. These are 1) the period δt between iterations of the algorithm, and 2) the convergence control k of M4. Experiments were performed in which these parameters were changed; the effect upon system behavior was as expected. Tuning was found to be a simple and straightforward process. For experimentation with a six-joint, cylindrical manipulator called the PACS arm (manufactured by Bendix Corporation), we selected a revolute joint with a rest position perpendicular to the plane of the base (i.e., the link hung vertically). This type of axis experiences nonlinear gravitational effects as it rotates. In addition, a stiff spring was attached from the end-effector to the base to exacerbate the nonlinearities. Tests were made with a variety of loads held by the grippers. With these different loads the joint was moved with various speeds so that the Coriolis effect would have impact on the test system as well (at faster motion rates).

In the tests the control algorithm brought the axis to the desired position in the requested amount of time. System variables were logged on a display device so that analysis could be performed. As Fig. 9 shows, the path quickly converged to linear form. Note that the low-order bits in the velocity requests become significant as $t \rightarrow T_C$ because they are inversely proportional to $t_r = T_C - t$. However, this is about the area of the ϵ-neighborhood. We would then enter another mode before these "nasty behaviors" manifested.

Accidentally dropped loads, which might severely impact a traditional controller, were quickly recovered in the adaptive environment. This experiment was done by snatching the load away from the robot during the motion. Also, a defined motion path was traversed nearly identically under different load conditions.

Although the tests we performed are simple, the results, as indicated above, are quite favorable. Furthermore, the general and flexible nature of our system structure should form a foundation for the intelligent control of the growing number of various types of industrial manipulators.

References

[1] A. J. Barbera, J. S. Albus, and M. L. Fitzgerald, "Hierarchical control of robots using minicomputers," in *Proc. 9th Int. Symp. on Industrial Robots*, Washington, DC, Mar. 1979, pp. 449-461.

[2] D. Graupe and G. N. Saridis, "Principles of intelligent controls for robotics, prosthetics and orthotics," in *Proc. NSF Workshop on Research Needed to Advance the State of Knowledge in Robotics*, Newport, RI, Apr. 1980.

[3] B. K. Kim and K. G. Shin, "An efficient minimum-time robot path planning under realistic constraints," in *Proc. 1984 Amer. Control Conf.*, San Diego, CA, June 1984, pp. 296-303.

[4] J. Y. S. Luh, M. W. Walker, and R. Paul, "Resolved-acceleration control of mechanical manipulators," *IEEE Trans. Automat. Contr.*, vol. AC-25, no. 3, pp. 468-474, June 1980.

[5] J. Y. S. Luh and C. S. Lin, "Optimum path planning for mechanical manipulators," *Trans. ASME J. Dynamic Syst., Meas., Contr.*, vol. 102, pp. 142-151, June 1981.

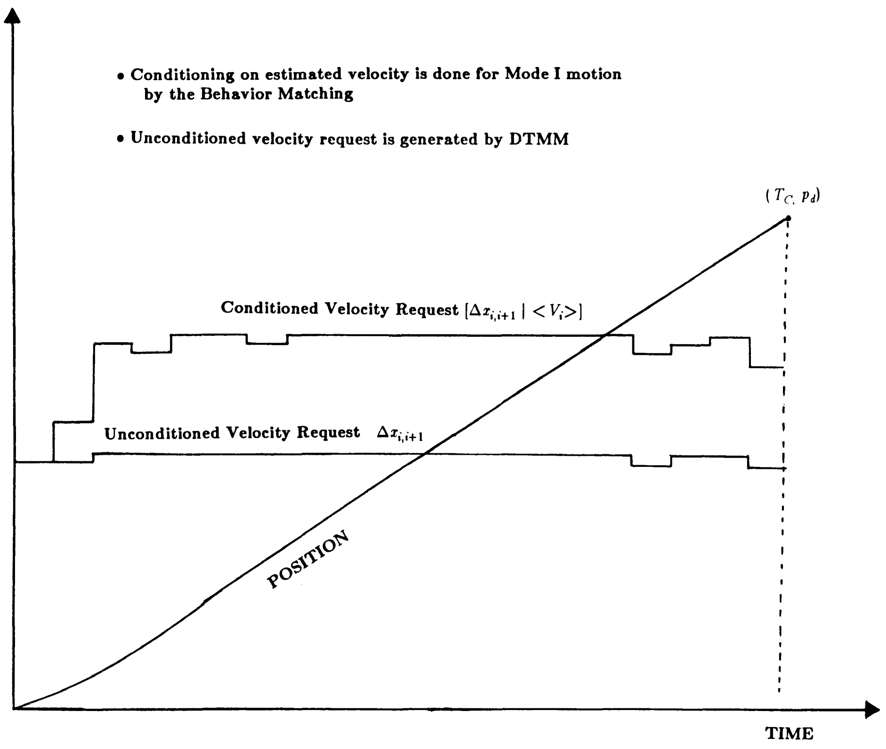

Fig. 9. Convergence of adaptive feedback control algorithm.

[6] J. Y. S. Luh and C. E. Campbell, "Collision-free path planning for industrial robots," in *Proc. 21 CDC*, Dec. 1982, pp. 84–88.

[7] J. Y. S. Luh, "An anatomy of industrial robots and their controls," *IEEE Trans. Automat. Contr.*, vol. AC-28, no. 2, pp. 133–153, Feb. 1983.

[8] M. T. Mason, "Compliance and force control for computer controlled manipulators," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-11, no. 6, pp. 418–432, Dec. 1981.

[9] N. D. McKay and K. G. Shin, "A microprocessor-based robot control system with a two-level hierarchy," in *Proc. 20th Int'l Symp. on Mini and Micro Computers and Their Applications*, Cambridge, MA, July 1982.

[10] R. P. C. Paul, "The mathematics of computer controlled manipulator," in *Proc. Joint Automatic Control Conf.*, vol. 1, 1977, pp. 124–131.

[11] _____, *Robot Manipulators: Mathematics, Programming and Control.* Cambridge, MA: MIT Press, 1981.

[12] M. H. Raibert and J. J. Craig, "Hybrid position/force control of manipulators," *Trans. ASME J. Dynamic Syst., Meas., Contr.*, vol. 102, pp. 126–133, 1981.

[13] K. G. Shin and S. B. Malin, "A hierarchically distributed robot control system," in *Proc. COMPSAC'80*, Oct. 1980, pp. 814–820.

[14] _____, "Dynamic adaptation of robot control to its actual behavior," in *Proc. 1981 IEEE Conf. on Cybernetics and Society*, Atlanta, GA, Oct. 1981.

[15] K. G. Shin and N. D. McKay, "An efficient robot arm control under geometric path constraints," in *Proc. 22nd CDC*, Dec. 1983, pp. 1449–1457.

[16] R. H. Taylor, "Planning and execution of straight-line manipulator trajectories," *IBM J. of Research and Development*, vol. 23, pp. 424–436, 1979.

[17] D. E. Whitney, "Resolved motion rate control of manipulators and human prosthesis," *IEEE Trans. Man-Machine Syst.*, vol. MMS-10, no. 2, pp. 47–53, June 1969.