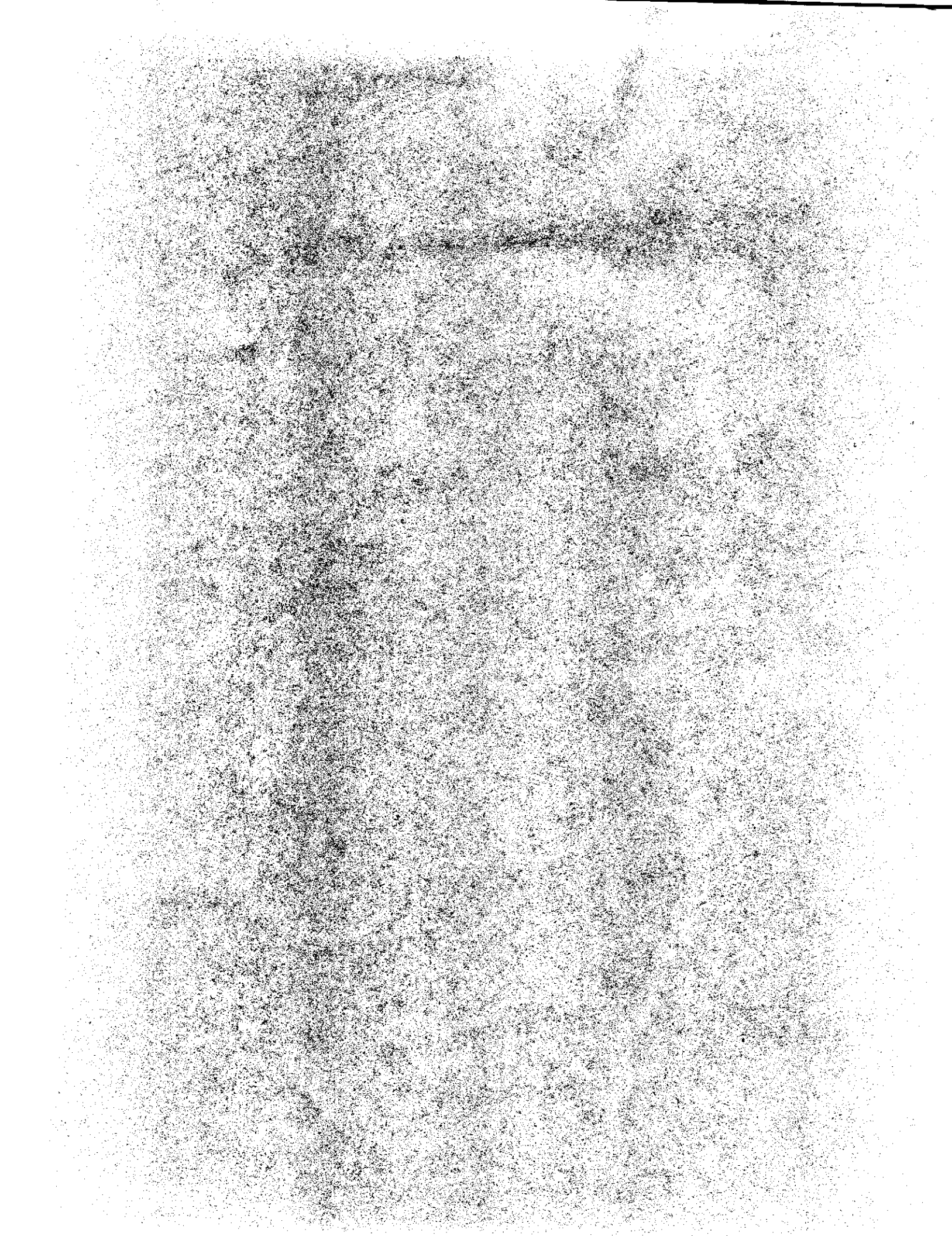
[redacted]



NISTIR 90-4286

IMPLEMENTATION OF A JACOBIAN-TRANSPOSE ALGORITHM

John Fiala
Al Wavering

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
Robot Systems Division
Intelligent Controls Group
Bldg. 220 Rm. B124
Gaithersburg, MD 20899

April 1990

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary
Lee Mercer, Deputy Under Secretary for Technology
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
John W. Lyons, Director

Implementation of a Jacobian-Transpose Algorithm

Intelligent Controls Group
Robot Systems Division
National Institute of Standards and Technology

Principle Authors: John Fiala
Al Wavering

Date: April 17, 1990

Document number: ICG-# 23
Document version: 1.0

Scope of the Document

This document describes an implementation of a Cartesian servo algorithm on a seven degree-of-freedom manipulator. In the basic algorithm the servo error is computed in Cartesian coordinates and transformed to joint-space torque commands by use of the transpose of the Jacobian relating the two coordinate systems. This control algorithm is described mathematically and in terms of the computational processes operating in a multiprocessing system to achieve the control. Two applications of the algorithm are described: compliant motions for contact of the arm with the environment, and ping-pong ball catching using real-time visual feedback.

Keywords: multiprocessing, control system architecture, impedance control, robot visual sensing, robot programming, robot trajectory generation

1. Introduction

The basic architecture for a robot control system is described in [1]. The architecture is hierarchical, consisting of multiple levels numbered from level 1, the lowest level, to level 4, the highest level for an individual robot. The implementation of this type of architecture being constructed in the Intelligent Controls Group (ICG) laboratory is shown in Figure 1. (Operator Interface connections are not shown in this figure.) As of this writing, the implementation involves only level 1 and level 2 system components. A more detailed discussion of the ICG approach to developing an implementation can be found in [7,21].

This document addresses the implementation of specific algorithms in the manipulator branch of the control system. The task decomposition structure of the manipulator Level 1 (Servo Level) is described in [3]. This structure is designed to support a number of different algorithms, however this document only discusses the implementation of one type of Servo Level algorithm. Other algorithms have been implemented in the laboratory as well. World modeling structure for the Servo Level is discussed in detail in [5]. Level 2 for a manipulator is also called the Primitive Level, and [4] describes the task decomposition structure of this level. The Primitive level is involved in using the Servo algorithm to achieve task objectives. This will be the subject of Section 4.

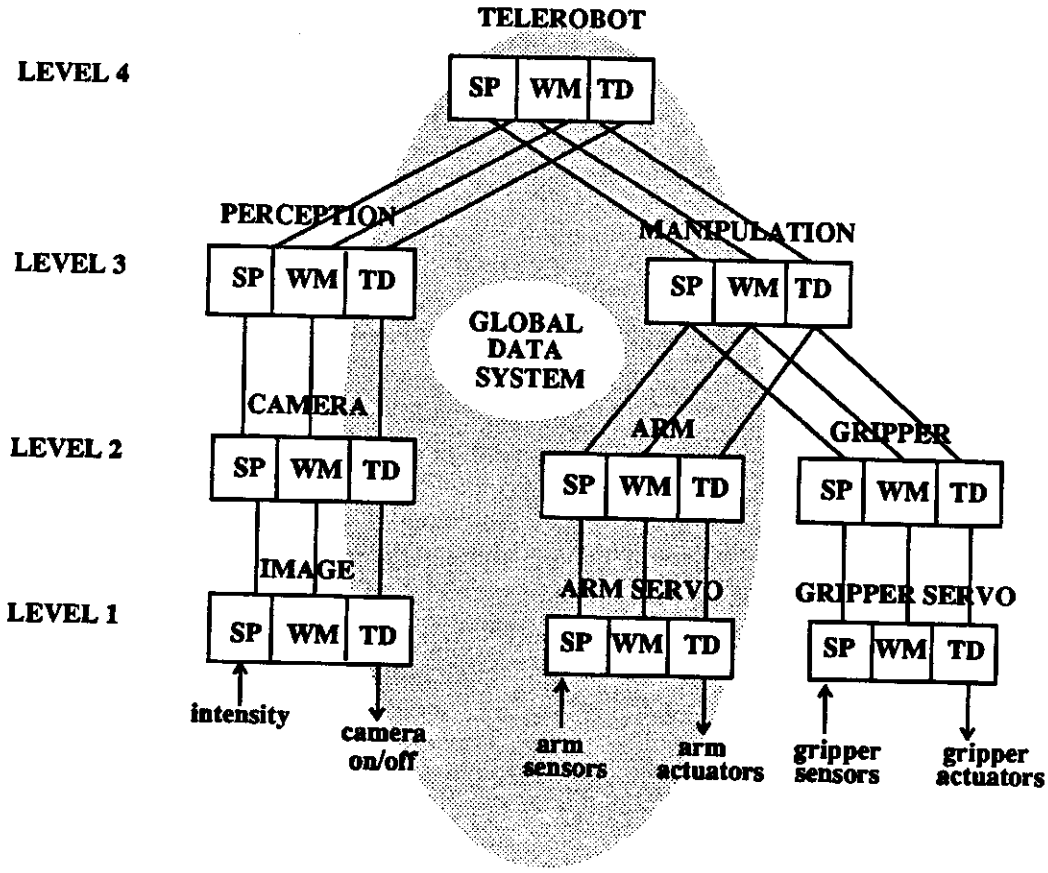


Figure 1. Hierarchical Architecture of Laboratory System

The type of algorithm which computes servo errors in a Cartesian coordinate system and relates those errors to joint torques by a Jacobian-transpose multiplication is the subject of this discussion. There are two principal reasons for wanting to use a control algorithm of this type. First, the fact that the servo is directly in the Cartesian coordinates eliminates the need to do explicit inverse kinematics, which can be expensive in the seven degree-of-freedom case. Second, the gains of the Cartesian servo provide a convenient means for achieving end effector compliance in task coordinates. Section 2 will describe the algorithm in detail and explain these points further. The algorithm does have some disadvantages as will be explained in Sections 2 and 4.

An important aspect of the control architecture is that it is designed to be implemented on a multiprocessor computing system. Specifically, the control levels and the control functions within a level are split up into separate software processes that can be distributed to the processors [7]. Section 3 details how the Jacobian-transpose algorithm is distributed among separate software processes and how these processes are distributed to processors. Included in this discussion is the description of the interface to the Robotics Research K-1607 controller.

Section 4 details the implementation of the Primitive level. The discussion focuses on how the Primitive level can make use of a Servo Level capable of executing a Jacobian-transpose algorithm. Two example tasks are described in this context. The first is a demonstration of how the algorithm can be used generate compliant contacts between the robot and stiff environments. The second example is a visual sensing task. The manipulator catches a ping-pong ball rolling down a planar surface using sensory data from the Perception part of the control system.

2. Algorithm

Notation: In the following all quantities will be vector and matrix quantities unless otherwise stated. Generally, Cartesian position vectors are represented by x, Cartesian force vectors by f, (the word force will be used for this vector even though it contains torques in the rotational degrees of freedom). Vectors of joint positions are represented by θ and joint torques by τ. First derivatives of these quantities are represented by a single dot over the character, and second derivatives by two dots. Cartesian vectors are of dimension six and joint vectors of dimension seven. The dimensions of matrices can be inferred from the vector context in which they are used. For example, the Jacobian-transpose J^t multiplies a Cartesian vector producing a joint vector. Thus, the matrix is 7x6. (In reality, the seven vector of 3 elements Cartesian position and 4 elements quaternion is used to represent x, however one should recognize that the underlying space is 6-D so that the Jacobian-transpose is still 7x6. This will be explained in more detail further on.)

The basic form of the Jacobian-transpose algorithm can be written

$$\tau_{act} = J^t(\theta)\,\{K_p(x_d - x) - K_v(\dot{x})\} + \tau_{gravity}(\theta) \qquad (1)$$

where τ_{act} is the command torque to be sent to the joint actuators, K_p and K_v are the position and velocity gains, x_d is the command (or reference) position in Cartesian coordinates, and $\tau_{gravity}(\theta)$ is a joint torque vector providing compensation for the effects of gravity [3,17]. The symbol (θ)

following a quantity indicates dependence of the quantity on manipulator position. The numeric values of such quantities in the control algorithm must be recomputed as the manipulator changes position.

The algorithm (1) can be recognized as basically a Proportional-Derivative (PD) control on Cartesian position with the addition of a gravity compensation term. The Cartesian feedback is obtained by computing the forward kinematics on the sensed joint positions and velocities,

$$x = kin(\theta)$$

$$\dot{x} = J(\theta)\,\dot{\theta} \qquad (2)$$

where x and $\dot{x}$ represent the position and velocity of the manipulator tip with respect to a Cartesian coordinate system fixed at the base of the robot [5].

The velocity representations in (2) should be clear; a 6x7 matrix, J(θ), times a 7-vector, $\dot{\theta}$, yields a 6-vector. The Cartesian 6-vector, $\dot{x}$, gives the rate of change of the Cartesian directions x, y, z, roll, pitch, yaw, in that order. See [16] for a definition of these Cartesian directions. The representation of x however is a bit more complicated. A quaternion is used to represent the rotational part of x as described in [2]. To compute x_d-x in (1), the translational components are determined using vector subtraction, and the rotational components are computed in two steps. First, the quaternion which represents the error between the desired and actual orientations is computed

$$^{w}R_{e} = {}^{w}R_{d} \bullet inv({}^{w}R)$$

where $^{w}R_e$ is the error quaternion with respect to world coordinates, and $^{w}R_d$ and ^{w}R are the the desired and actual orientations with respect to world coordinates. The definition of inverse and multiplication operations for quaternions may be found in [2]. Next, the error quaternion is converted to a 3-dimensional differential rotation vector by multiplying the components of the axis of the quaternion by the magnitude of the angle of the quaternion. The equivalence of these two representations for small rotations is demonstrated in [16]. The rotational difference operation will continue to be represented as a subtraction in the text. Note that all other operations in (1) are the normal operations of vector addition (subtraction) and matrix multiplication.

The proportional gain K_p defines the *stiffness* of the control with respect to deviations from the reference position x_d. The velocity gain K_v determines the amount of *damping* provided by the control. The combination of the gains with the Cartesian position error and velocity can be thought of as a Cartesian force. This force is related to joint coordinates through a Jacobian-transpose matrix multiplication. The result is a joint torque vector which can be commanded to the K-1607 through the robot's torque interface [8].

By selecting the gains appropriately, the manipulator can be made to have a certain stiffness and damping, or *impedance*, at the end effector. Thus, (1) is related to Hogan's work on impedance control [11]. In our control system implementation, the gains are included as part of the command vector to the Servo Level [3], and can therefore be changed every Servo cycle, if desired. Modulation of the control gains play a major role in the applications described in Section 4.

As pointed out by Craig [10], the coupling between degrees of freedom in an articulated mechanism will degrade any control scheme which tries to control the degrees of freedom independently, as in (1). Degrees of freedom can be decoupled by model-based control techniques which compensate for the nonlinear inertia, Coriolis, and centrifugal effects [3,10,13,15]. These techniques rely on exact cancellation of coupling and require that accurate models of the manipulator be available. Many times, as in the current ICG system, such models are not available and these control techniques cannot be used. This document discusses what can be achieved without complete models. The only part of the manipulator model currently used, in addition to the kinematic model [5], is the gravity torque model. This model assumes lumped masses at the centers-of-gravity of the links and necessarily has inaccuracies.

Surprisingly, some analytical stability results can be obtained for (1). As discussed in [17,19], it can be shown that for a robot governed by a dynamic equation of

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

a control scheme (1) is asymptotically stable for any positive definite symmetric matrices K_p, K_v. However, there are a number of caveats to this result. One is that the result is obtained for x_d stationary and does not readily generalize to the case of $x_d(t)$ defining a trajectory for the manipulator. In addition, it may be the case that the stability is not obtained globally. There may be local minima in the potential that prevent the manipulator from reaching the goal. This *stall* behavior is particularly a problem when there are other potential fields operating in the control scheme. Such is the case with joint limit avoidance potentials, discussed below.

The Jacobian-transpose has inherent stall points at the kinematic singularities of the mechanism [19]. In these situations, the manipulator is typically unable to move in a Cartesian direction. This is reflected in the algorithm by a column of the Jacobian-transpose matrix becoming zero, or linearly dependent, such that the algorithm is no longer responsive to Cartesian commands along that direction. That is, joint torques are not produced which provide motion along that Cartesian degree-of-freedom. Note that the control algorithm does not "blow-up" and produce infinite joint rates as in many inverse control schemes. This is a distinct advantage of this algorithm. Also, as stated in [3,4,19], it may be possible to avoid kinematic singularities and other stall points by appropriate planning at higher levels.

Another caveat to the stability result obtained for (1) is limitations on the gains. There is an upper-bound on the magnitude of the gains that can be used with such a control scheme on a real manipulator. This upper-bound limits the performance in terms of the control bandwidth. For most manipulators the upper-bounds on control gains must be determined by experiment [20]. The biggest limiting factor for these upper-bounds is the discrete sampling or around-the-loop rate of the control scheme. For (1), this rate is determined by the length of time is takes to read the joint feedback, compute forward kinematics, compute the control, and output the analog torque command (the around-the-loop rates obtained so far in the ICG laboratory are discussed in the next section). Note that the computation of the Jacobian does not enter into the around-the-loop control rate. This is because it is slowly varying and may be computed at a slower rate than the feedback control itself [12,13].

The basic algorithm (1) can be modified to improve the control bandwidth for a given around-

the-loop rate. One possibility is to add lag-lead compensation to the basic PD algorithm [22]. Lag compensation in the velocity feedback path may remove some high-frequency excitation from the loop, for example. Another modification would be to perform the feedback control in joint space while maintaining a required Cartesian impedance. This can be done by computing new joint gain matrices from the desired Cartesian matrices.

$$K_1 = J^t K_p J \quad \text{and} \quad K_2 = J^t K_v J$$

Then the feedback control loop can be done without the computational cost of forward kinematics.

$$\tau_{act} = K_1(\theta_d - \theta) - K_2(\dot{\theta}) + \tau_{gravity}(\theta)$$

This obviously requires that the reference position θ_d be specified in joint space rather than in Cartesian space. In addition, although the matrix multiplication required by the Jacobian-transpose has been removed, general matrix multiplications are still required for K_1 and K_2, which are 7x7 matrices.

A reference velocity may be added to (1) to improve tracking of the desired trajectory. This is, in fact, done in our laboratory control system.

$$\tau_{act} = J^t \{K_p(x_d - x) + K_v(\dot{x}_d - \dot{x})\} + \tau_{gravity} \qquad (3)$$

(The θ-dependency notation on the Jacobian and gravity terms is dropped for convenience.)

Additional terms have been added to (3) to enhance the control systems performance and capabilities. The simplest term is a friction compensation term which just adds a constant torque to overcome Coulombic friction disturbances.

$$\tau_{act} = J^t \{K_p(x_d - x) + K_v(\dot{x}_d - \dot{x})\} + \tau_{gravity} + \tau_{fric} \qquad (4)$$

where, for each joint i,

$$\tau_{fric}[i] = \begin{cases} c_1[i], & \dot{\theta}[i] > v_1[i]; \\ 0, & v_2[i] \le \dot{\theta}[i] \le v_1[i]; \\ c_2[i], & \dot{\theta}[i] < v_2[i] \end{cases}$$

The friction compensation torque is zero for a joint whose velocity falls within some deadband, given by constants v_1 and v_2. The compensation is either a positive (c_1) or negative (c_2) constant outside this deadband. The friction compensation constants are determined empirically for each joint.

To help prevent accidental collisions with joint hardstops and other violations of joint limits, a joint torque for avoidance of joint limits is computed and added to (4).

$$\tau_{act} = J^t \{ K_p (x_d - x) + K_v (\dot{x}_d - \dot{x}) \} + \tau_{gravity} + \tau_{fric} + \tau_{jl} \qquad (5)$$

where

$$\tau_{jl}[i] = \begin{cases} k[i]\ (l_1[i] - \theta[i]), & \theta[i] > l_1[i]; \\ 0, & l_2[i] \le \theta[i] \le l_1[i]; \\ k[i]\ (l_2[i] - \theta[i]), & \theta[i] < l_2[i] \end{cases}$$

Here, l_1 gives the upper limit for the joint position, and l_2 gives the lower limit. These limits specify the position at which the joint limit spring force begins to exert a torque resisting further movement toward the physical joint limits. The amount of force generated for joint limit avoidance is determined by the spring constant k.

Just as for joint limits, torques which act to avoid obstacles [12] and other things may be computed and added to the control. Likewise, attractive torques may be added for desirable things such as regions of maximum manipulability [18] and desired points of contact or other activity. The presence of attractive and avoidance points in the robot work volume can create local minima which can stall the manipulator. A simple example can be seen with (5), when the shortest straight-line distance to the goal x_d would take one or more joints through a joint limit. The joint limit avoidance torque would keep the manipulator away from the joint limit(s) and the manipulator motion would come to a stop at a point where the joint limit avoidance force balanced the PD force trying to reach the Cartesian goal. In this case, the stall behavior is beneficial since it prevents a joint limit violation. But, since there is another path that would lead to the Cartesian goal, it might be preferable to have the manipulator move along this path rather than just stall. Determining this non-stalling path and planning trajectories that move the manipulator along it would appear to be tasks of higher levels in the control system.

Another undesirable behavior associated with (5) is *jumping* behavior. This typically occurs when the manipulator is stalled due to certain forces, such as *stiction*, and then, as the goal x_d is moved, other forces in the control scheme build-up until the manipulator jumps out of the stall point. Joint stiction is the worst culprit for producing jumping effects. If significant stiction is present and a low velocity movement is made, the manipulator may even bounce as it alternately stalls and jumps along the trajectory. Obviously, this effect is most prominent when using small gains K_p and K_v. Higher gains will tend to reject stiction disturbances.

Since the manipulator has seven degrees of freedom, it is redundant with respect to the Cartesian PD task of (5). This means that the manipulator can execute *self-motions* while still satisfying the Cartesian control task. Note that (5) provides no control for this self-motion whatsoever, except to support it with respect to gravity. For a manipulator controlled by (5) one could push the elbow around to any angle and it would stay there. Additional control of the self-motion can be provided by computing a null-space operator [5,14]

$$(I - J^{+}J), \text{ where } J^{+} = J^{t}(JJ^{t})^{-1}$$

Through this operator a joint torque vector can be computed which acts only to produce manipulator self-motion.

$$\tau_{ns} = (I - J^{+}J)\tau \qquad (6)$$

The vector τ is selected to provide the desired control for self-motion. For example, τ could be some type of avoidance torque as described above. Through this torque the self-motion could be made to avoid obstacle contact or other undesirable circumstances which may arise during the task. In the ICG control system, due to joint 1 stiction problems, the following self motion control vector has been used

$$\tau[i] = \begin{cases} -k\theta[i], & i=1 \\ 0, & \text{otherwise} \end{cases}$$

This torque tends to move the self-motion such that joint 1 remains in the center of its travel. Combining this with (5) yields the control equation which is currently implemented at the Servo Level.

$$\tau_{act} = J^{t}\{K_p(x_d - x) + K_v(\dot{x}_d - \dot{x})\} + \tau_{gravity} + \tau_{fric} + \tau_{jl} + \tau_{ns} \qquad (7)$$

Another approach to handling the redundancy would be to include it explicitly in (5). This requires that the Cartesian vectors be modified to include a self-motion parameter which specifies the angle of the elbow. This parameter can be a scalar variable which corresponds to the angle between the manipulator elbow plane and the vertical plane [23]. The Jacobian must be augmented to relate changes in this angle to joint space. This *augmented Jacobian* can then be used in a control scheme analogous to (5).

$$\tau_{act} = J_a^{t}\{K_p(z_d - z) + K_v(\dot{z}_d - \dot{z})\} + \tau_{gravity} + \tau_{fric} + \tau_{jl} \qquad (8)$$

In this equation, J_a^t is the transpose of the 7x7 augmented Jacobian matrix, and the z vectors are the 7-D control vectors which include the 6-D Cartesian position of the end effector and the elbow angle parameter. Obviously, for this approach the forward kinematics are modified to include the computation of the elbow angle parameter. The approach has been implemented in the ICG system, and its development at JPL for the Robotics Research arm has been well-documented in the literature [23,24].

Given that (7) is the basic control algorithm, it is desirable to allow the coordinate frame in which the reference x_d is specified to be selected as appropriate for the application. For example, if the task involves motion with respect to some object fixed in the environment, (as in assembly tasks,) it may be more convenient to specifiy the reference x_d with respect to a coordinate system fixed in the object rather than with respect to a coordinate system fixed at the robot base. This type

of modification is straight-forward with (7) as described in [3]. It involves computing the 6x6 differential change matrix [16] corresponding to the change of coordinates and multiplying this matrix by the Jacobian to get a new Jacobian for use in (7). The forward kinematics are modified with this additional coordinate transformation as well. Parts of this technique have been tested in the ICG system, but it has not been implemented entirely due to timing limitations of the current hardware configuration.

One other variation on (7) which has been implemented in the ICG control system is to compute the Cartesian control which respect to end effector coordinates instead of world coordinates. This requires that a new Jacobian be computed which relates joint rates to the Cartesian velocities of the end effector represented in the coordinate frame defined at the end effector. In this scheme, x_d is still specified in world coordinates, but the operation x_d-x is modified to give the error relative to the end effector frame.

$$^{endeff}R_e = inv(^wR) \bullet {}^wR_d$$

$$^{endeff}P_e = inv(^wR) \bullet (^wP_d - {}^wP)$$

The symbols involving R represent the rotational part of the Cartesian vectors, that is, R is the rotational part of x and R_d is the rotational part of x_d. Likewise, the symbols involving P represent the positional part of the Cartesian vectors. Compare with description of the world coordinate difference operation on page 4.

The end effector coordinate version of (7) is easily implemented by substituting the appropriate x_d-x and Jacobian computations. The change in the Cartesian difference operation is achieved by selecting a different *servo algorithm* for the Servo Execution process [3], whereas the end effector Jacobian might be implemented by a completely new and separate process in the system. The configuration of functions in terms of processes is discussed in the next section.

3. Servo Processes

Implementation of (7) in the ICG laboratory is achieved by a set of eight software processes coded in Ada using the process model described in [7]. These processes implement the descriptions given in [3,5] for the interfaces and functions of Servo Level processes. These processes are distributed to 4 processors in a multiprocessor backplane. This section describes the Servo processes and their allocation to processors. Figure 2 shows the Servo Level processes used to implement (7).

The RRC Communications process communicates with the K-1607 controller using the serial link as described in [8]. A 68010 processor board is dedicated to running this process to provide 5 msec updates to the K-1607 controller. The RRC Communications process reads the command torque from common memory, transmits these values to the K-1607 controller, and then receives the feedback data and writes the values to common memory.

The Joint Feedback process gets the feedback data written by the RRC Communications process and converts it to SI units. The result, joint position and velocity feedback, is written to common memory.

The joint feedback is used by the four world modeling processes to compute elements of the

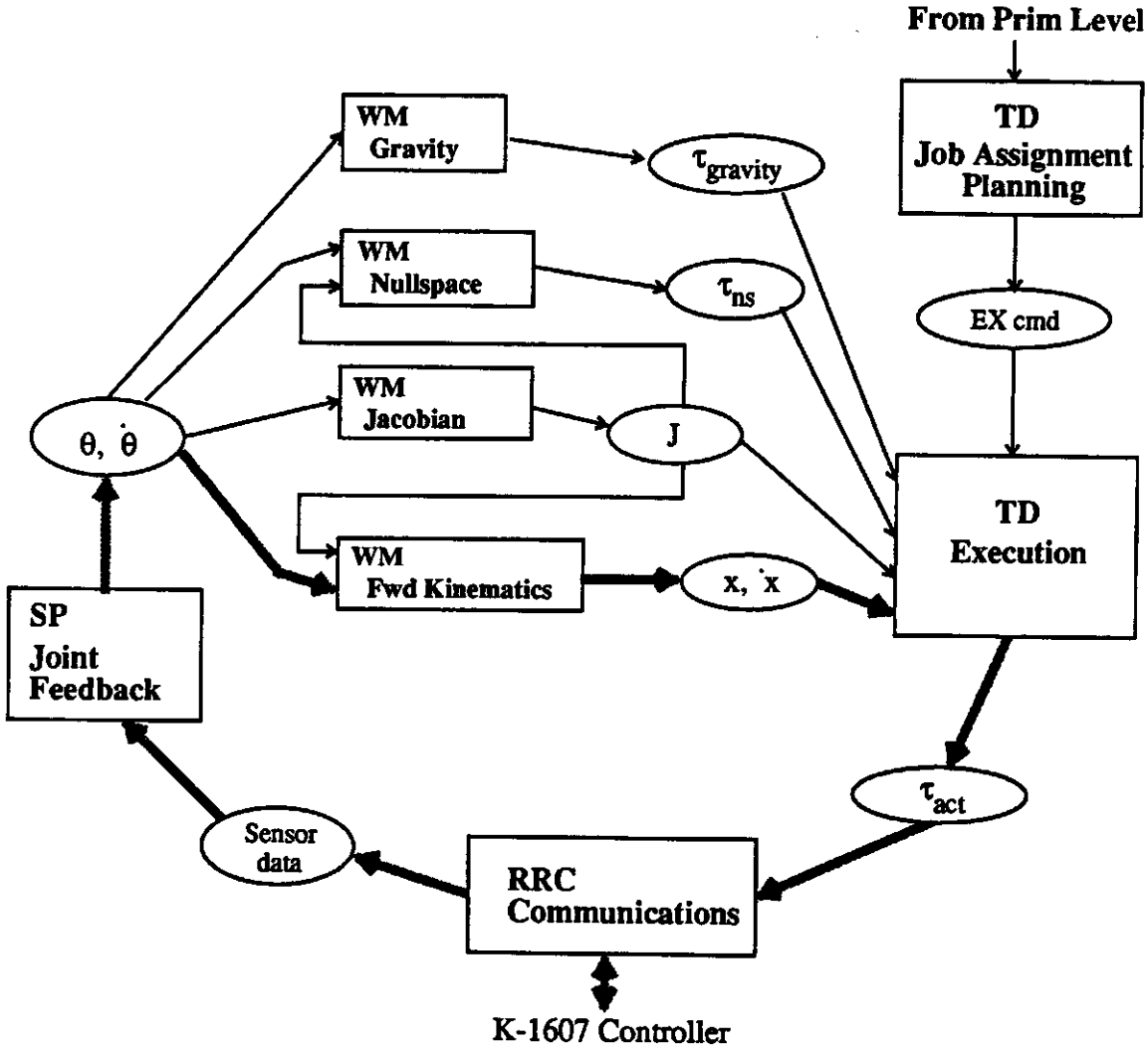


Figure 2. Servo Level Processes.

control algorithm. The Gravity process computes the gravity compensation torque. The Nullspace process computes the nullspace torque according to (6). The Jacobian process computes the Jacobian which relates changes in a world coordinate system to joint space. As described earlier, the Jacobian is used by the both the Nullspace process and the Forward Kinematics process. The Forward Kinematics process converts the joint feedback into Cartesian feedback using (2).

The Execution process computes (7). The process reads the elements of the equation from common memory locations written by the world modeling processes and the Job Assignment/Planning process. The friction compensation torque τ_{fric} and the joint limit avoidance torque τ_{jl} are computed within the Execution process itself. These computations could probably be separated as additional world modeling processes, however, the calculations are currently so simple that there is no advantage to such a separation.

The Job Assignment/Planning process is a single process which performs the functions of both

Table 1. Servo Process Execution Times.

Process	Execution time
Job Assignment/Planning	1.4 msec
Execution	3.5 msec
Joint Feedback	0.5 msec
Gravity	2.1 msec
Nullspace	17.6 msec
Jacobian	4.7 msec
Forward Kinematics	3.2 msec

the Job Assignment and Planning processes described in [3]. These were originally separate processes, but were combined to fit on a processor with Forward Kinematics. These processes will be separated when an additional processor is added to the system. The Job Assignment/Planning process receives the Servo commands from the Primitive Execution process. The Cartesian reference signals x_d and $\dot{x}_d$, and the gains K_p and K_v, are output as the *Ex cmd* to the Execution process.

Table 1 shows the execution times for the processes of Figure 1. The time shown is the execution time for one complete cycle of the process computing the appropriate component of (7). This includes common memory reads and writes. These times were obtained from runs of the actual code on the target hardware.

The RRC Communications process, as stated above, runs on a 5 msec cycle synchronized with K-1607 controller. The 5 msec cycle time is determined by the update rate of the torque interface. The torque interface accepts new torque commands every 2.5 msec. However, the current serial link to the K-1607 controller cannot run fast enough to provide new updates every time, so updates are provided every other cycle instead[8].

The Execution process should provide new torques to the RRC Communications process every 5 msec so that the torque interface will receive new data at the fastest possible rate. Thus, Execution should run every 5 msec. From Table 1, the process will take 3.5 msec of a 5 msec cycle when computing (7). This leaves about 1.5 msec for a processor to be servicing other processes. When the Execution process is running in joint-space, (and not running the Jacobian-transpose algorithm,) the control loop time can be minimized by obtaining the joint feedback in the same 5 msec cycle as Execution process. This fact, plus the fact that no other process really fits, lead to grouping the Joint Feedback process with the Execution process on a single processor. These processes execute in about 4 msec, and repeat execution every time there is new data available from the RRC Communications process, i.e. every 5 msec.

As stated in Section 2, it is important that the forward kinematics be computed as rapidly as possible to improve the around-the-loop rate of the algorithm. Therefore, everytime there is new joint feedback data, the Forward Kinematics process should run. This implies a 5 msec cycle for this process. Except for Job Assignment/Planning, all other processes can run at rates slower than

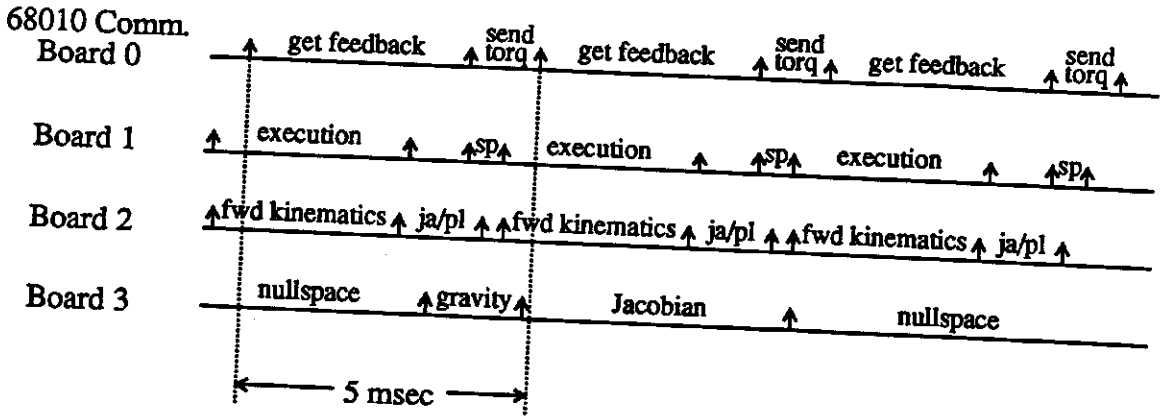


Figure 3. Servo Timing.

the control loop rate. Thus, the Job Assignment/Planning process and the Forward Kinematics process are grouped on one processor, which consumes 4.6 msec out of a 5 msec cycle.

The remaining processes, Gravity, Nullspace, and Jacobian, are lumped together on another processor and allowed to run one-after-the-other at the maximum rate. Since the total execution time of these processes is 24.4 msec, each of these processes will update its output every 25 msec, approximately. This rate is perfectly acceptable for these processes because of the role they play in the algorithm.

Figure 3 shows the Servo timing for the system for the four processor boards. One can trace the data through the system, from the start of feedback to the end of torque command, (the bold arrows of Figure 2,) and see that the around-the-loop time for the Jacobian-transpose algorithm is 15 msec.

Note also that the process-to-processor mapping described above considered only the execution of one algorithm. Since there is no automatic method for distributing processes to processors dynamically in the ICG system, the design must be able to handle all operating modes of the level. For example, if the end effector Jacobian-transpose algorithm described at the end of Section 2 is to be used in the same system with the world Jacobian-transpose algorithm, the process-to-processor mapping must be compatible with both schemes. This means that the processor running the Execution process must have enough processing margin to handle the additional cost of computing the relative Cartesian position error. Likewise, there must be sufficient processing power to compute the end effector Jacobian. (See [25,26] for more on distributing processes to processors.)

Since both the world and end effector Jacobians are not generally required at the same time, a simple scheme has been established for activating and inactivating processes as described in [7]. Thus, a process to compute the end effector Jacobian could be *active*, and a process to compute the world Jacobian *inactive*, for the operation of the end effector Jacobian-transpose algorithm. In the current implementation the activation of processes is handled by the Job Assigner. When switching between algorithms, the execution of the new control mode is delayed until the level is fully reconfigured for the new mode.

4. Primitive & Applications

The Jacobian-transpose servo algorithm has been used to perform two demonstration tasks in our lab. The first is stable, nondestructive contact and surface-following with a stiff environment. The second is catching a ping-pong ball which rolls down a planar surface populated with obstacles to produce a non-deterministic ball trajectory. This section describes the Primitive level trajectory generation algorithms which are used with the Jacobian-transpose servo algorithm to perform these tasks. See [4] for additional information on the structure and function of the Primitive task decomposition processes. Also, a videotape is available from the Intelligent Controls Group at NIST which demonstrates the described tasks.

4.1. General-Purpose Trajectories

The system currently provides two trajectory generation algorithms for general-purpose Cartesian motions, including those for which contact will occur between the robot end effector and fixed, stiff objects in the environment. The first one plans and executes a trajectory with a trapezoidal velocity profile. The second is based on a quintic polynomial function for the time sequence of Cartesian positions. Both types of trajectories are discussed in detail in [10]. For both types of trajectories, the user enters the desired Cartesian goal position, execution time, and the name of the object, if any, which will be contacted during the motion. The trajectory algorithms are currently implemented only for cases where the initial and final velocities and accelerations are zero. The planning for these trajectories is currently performed only once; there is no trajectory replanning while the motion is being performed. Gains remain constant throughout the motion, and remain effective until the next trajectory is performed.

Trapezoidal-profile motions consist of a constant acceleration portion followed by constant velocity, followed by constant deceleration to the goal. The trapezoidal motions generated by our implementation are symmetric; that is, the magnitude of acceleration is the same as the magnitude of deceleration. For trapezoidal trajectories, the Primitive Planning process determines the acceleration/deceleration and constant velocity for each Cartesian degree-of-freedom, and the time which will be spent accelerating. The percentage of time to be spent in acceleration is read from a planning file. An acceleration period which is 5% of the total motion time works well for most situations. This results in a velocity for the constant speed portion which is about 5% greater than the average velocity for the motion. Using this approach, both acceleration and velocity will be larger for fast motions than for slower ones. The Planning process also determines the gains to be used during the motion. Currently, gains are obtained from a file associated with the name of the contact object.

Once the Planning process sends down the parameters for a new trapezoidal trajectory, the Primitive Execution process starts executing the motion. On each execution cycle, the desired Cartesian position and velocity are computed based on the time which has elapsed since the start of the motion.

For quintic polynomial trajectories with zero initial and final velocities and accelerations, the Planning process need only determine the gains and pass along the goal position and desired traversal time. The parameters for the quintic polynomial do not have to be precomputed in this case, since the equation may be rewritten to depend only on the initial and goal positions as
where x_1 = goal position, x_0 = initial position, and h(t) = (traversal time - t) / t. Similarly, the fol-

$$x(t) = x_1 - (x_1 - x_0)\,(10h^3(t) - 15h^4(t) + 6h^5(t))$$

lowing equation may be used for rotations

$$R(t) = R_1 \bullet R(n, -\phi\,(10h^3(t) - 15h^4(t) + 6h^5(t))$$

where R_1 = goal orientation, $R(n,\phi)$ = rotation of angle ϕ about axis n required to reorient R_0 into R_1. These equations are evaluated by the Execution process for increasing values of t until the end of the motion is reached. The initial position is read from the Cartesian position feedback buffer on the first cycle of execution. The difference between the initial position and the goal position is also computed during this first cycle for use throughout the motion. No velocity or acceleration checking is currently performed for Cartesian quintic trajectories. Also, only the desired position is commanded to Servo at this time, although adding the desired velocity would not be difficult. The Primitive Execution process evaluates a new Servo goal every 5 ms for both trapezoidal and quintic trajectories.

Either of these trajectory algorithms may be used with the Jacobian-transpose servo algorithm to perform Cartesian straight-line motions. Relatively low gains may be used in directions of stiff contact with environmental objects to keep reaction forces small, while high gains may be used for degrees of freedom which will not experience contact. The use of higher gains for non-contacting degrees of freedom allows more accurate performance of the desired trajectory when reaction forces are not an issue. The largest gains which have been used with (7) (for free space motions) are

$$K_p = \begin{bmatrix} 4000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 700 & 0 & 0 \\ 0 & 0 & 0 & 0 & 700 & 0 \\ 0 & 0 & 0 & 0 & 0 & 700 \end{bmatrix}$$

$$K_v = \begin{bmatrix} 200 & 0 & 0 & 0 & 0 & 0 \\ 0 & 200 & 0 & 0 & 0 & 0 \\ 0 & 0 & 200 & 0 & 0 & 0 \\ 0 & 0 & 0 & 30 & 0 & 0 \\ 0 & 0 & 0 & 0 & 30 & 0 \\ 0 & 0 & 0 & 0 & 0 & 30 \end{bmatrix}$$

where the units are N/m for the translational elements of K_p, N-m/rad for the rotational elements of K_p, N-s/m for the translational elements of K_v, and N-m-s/rad for the rotational elements of K_v. The largest stable gains will, of course, be robot configuration-dependent. Larger gains (especially for larger rotational velocity gains) result in instability. No special effort has been made to optimize the system to be able to use the highest gains possible, however. It is expected that considerably higher gains could be used if the around-the-loop time were decreased and lag-lead compensation were implemented as mentioned in Section 2.

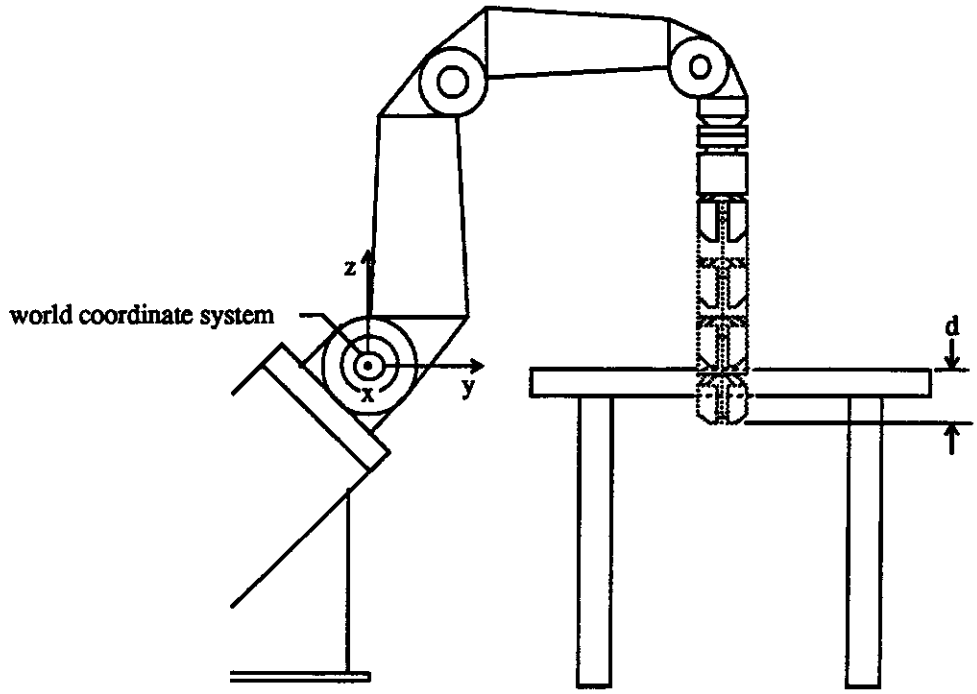


Figure 4. Example Robot Contact Motion.

A specific example will now be discussed to show how this type of motion works when contact with a stiff object is encountered. Consider the motion of the robot gripper straight down onto a rigid table, as shown in Figure 4. The desired robot motion is to start above the table, maintain contact for a while, then move back to the start position. For this motion, trapezoidal trajectories are used along with the Jacobian-transpose servo algorithm operating in world coordinates, and the robot is commanded to move to a position below the table surface. A relatively small gain is used in the world Z direction, while the gains in the other five Cartesian directions are fairly large. The actual Cartesian gains used for this motion are

$$K_p = \begin{bmatrix} 2500 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2500 & 0 & 0 & 0 & 0 \\ 0 & 0 & 300 & 0 & 0 & 0 \\ 0 & 0 & 0 & 500 & 0 & 0 \\ 0 & 0 & 0 & 0 & 500 & 0 \\ 0 & 0 & 0 & 0 & 0 & 500 \end{bmatrix}$$

The robot will follow the desired trajectory until contact is made with the table. As the desired servo goal positions go further below the surface of the table, the reaction force between the gripper and the table increases due to the increase in position error. Figure 5 shows a plot of the expected and actual forces in the Z direction during contact with the table. The actual force was measured by placing a six-axis force/moment sensor between the gripper and the table during the motion. The expected force includes the effect of the additional position error introduced by the thickness

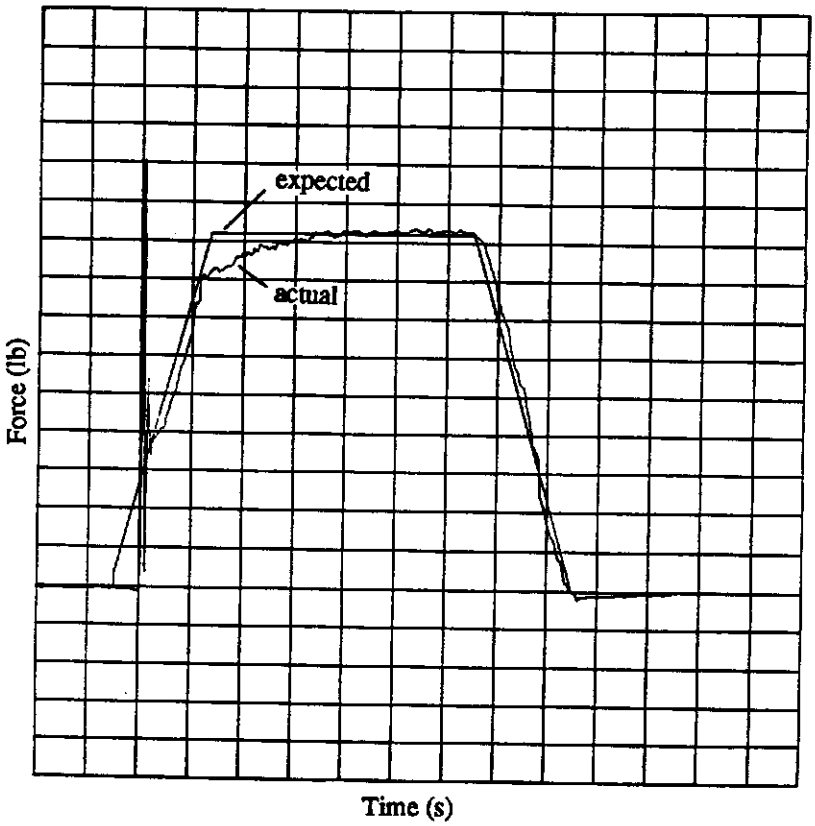


Figure 5. Expected and Actual Z Forces After Table Contact.

$$K_v = \begin{bmatrix} 55 & 0 & 0 & 0 & 0 & 0 \\ 0 & 55 & 0 & 0 & 0 & 0 \\ 0 & 0 & 20 & 0 & 0 & 0 \\ 0 & 0 & 0 & 30 & 0 & 0 \\ 0 & 0 & 0 & 0 & 30 & 0 \\ 0 & 0 & 0 & 0 & 0 & 30 \end{bmatrix}$$

of the force sensor, but it does not include an estimate of the impulsive force experienced at impact. Note that there is a delay between the time when contact is expected to occur and the time of the actual impact. This is due to the trajectory following error, which is quite large as a result of the low gain in the Z direction. The end effector is moving at about 0.07 m/s at impact, and even at this slow speed it is clear from Figure 5 that large forces are experienced as the kinetic energy of the manipulator is dissipated.

After the impulsive forces from the initial impact have diminished and the trajectory has completed, the remaining force experienced by the gripper is very close to the theoretical value predicted by multiplying the Cartesian position gain in the Z direction with the final position error, in that

direction (the dimension d in Figure 4, 0.137 m in our example). After a dwell leaving the goal position unchanged, the force decreases as expected when the robot starts moving back up toward the start position.

The Z gain used in this example is small enough that a styrofoam coffee cup placed upside-down on the table below the gripper will not be crushed during the contact portion, even though the resulting position error is increased further. Similar trajectories, using the same gains, have been used to slide the gripper across the table surface in a stable fashion. Again, the Z gain is small enough that a light bulb can be placed below the gripper during this motion and slid along the table without breaking .

Although the low Z gain in this example is useful for maintaining contact without generating excessive interaction forces, it results in degraded trajectory tracking when moving toward the table, as mentioned above. Because the gain is low, disturbances such as those caused by residual joint friction and inertial effects are poorly rejected. Joint friction causes the free-space motion to be unsteady; slowing down because of friction, then speeding up as the error becomes large enough to pull the manipulator along.

Several steps have been taken to reduce the frictional effects. First, the robot manufacturer has made modifications to reduce joint friction where possible. Second, some friction compensation is provided by the τ_{fric} terms in the servo algorithm (7). An attempt could be made to implement a more sophisticated friction model, but it is difficult to compensate for friction exactly because the parameters are typically time-varying and position- and direction-dependent. Third, the motion is improved somewhat by increasing the velocity gain and commanding a desired velocity. For faster motions with low gains, inertial effects will also cause substantial errors. These may be reduced by providing compensation in the servo algorithm, if sufficient computational resources are available.

In summary, the experiments performed indicate that the Jacobian-transpose servo algorithm (7) can be used to perform useful compliant motions. Compensation for gravity (and friction, to an extent,) frees the position control algorithm from having to reject these disturbances, which allows gains to be selected based on the task. Although contact forces are not controlled explicitly, gains may be selected which will result in ballpark forces if the amount of deflection is known or can be estimated. In most cases the exact forces experienced are not critical, as long as the robot complies in the desired manner and destructively large forces are avoided. Further experiments will be performed to evaluate the performance of this algorithm in actual space station-related assembly operations.

4.2. Sensory-Interactive (Ball-Catching) Trajectories

The trajectory generation algorithm used in the ping-pong ball catching task is quite different from those described above. Before discussing details of the algorithm, it will be useful to describe the experimental setup for this task. As shown in Figure 6, a board (2'x2') is mounted on a tripod so that it may be positioned easily in the robot workspace, and the board orientation may also be easily adjusted. The board has a number of pins placed to produce a non-deterministic ball trajectory. An example of an actual pin layout we used is shown in Figure 7. When a ball is rolled down the board, its motion is observed by a stationary camera. An image differencing algorithm is used to locate the centroid of the moving ball image. Image space centroid values are filtered and trans-

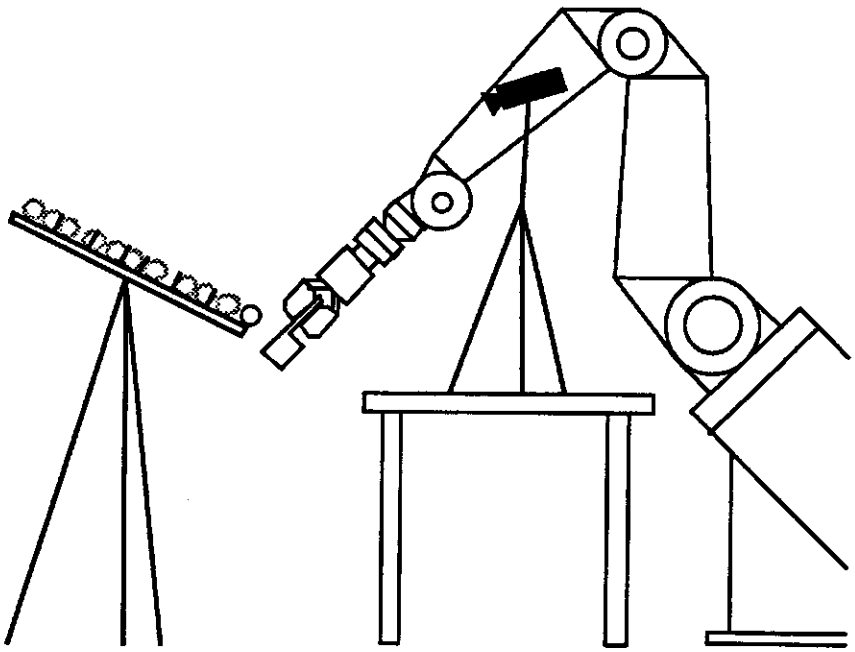


Figure 6. Ball Catching Setup.

formed to robot world coordinates by a World Modeling process. The function of the Sensory Processing and World Modeling algorithms for the ball-catching demonstration task are described in [9, 28]. The objective is for the robot to use an 8 oz measuring cup (3 in. diameter) to catch the ball as it rolls off the bottom edge of the board.

The position of the board in world coordinates is available to the Primitive Planning process. Currently, the board position is determined by teaching three points on the board surface; the board position and orientation is then stored in a data file. Teaching is performed by controlling the manipulator with gravity compensation torques only, allowing it to "float". The gripper is then manually moved to the desired position, and the manipulator joint or Cartesian position is recorded by reading the common memory buffers described in Section 3. The actual board position we have used is somewhat different from that shown in Figure 6.

The approach taken to perform this task is one in which the Planning process plays a minor role, and most of the trajectory generation is performed by the Execution process. The general idea is this:

1) There is a line just away from the bottom of the board along which the cup will be moved to catch the ball. The goal position is the projection of the estimated ball position down onto that line, as shown in Figure 7. The estimate of the ball position includes a small amount of prediction, as will be discussed later.

2) Instead of planning a trajectory to take the robot from the current position to the goal position, the goal position is always sent directly to Servo. However, when the ball is near the top of the board, a relatively low gain in the direction of the goal pose line is sent to Servo (along with the position goal), and as the ball rolls down the board, this gain is increased

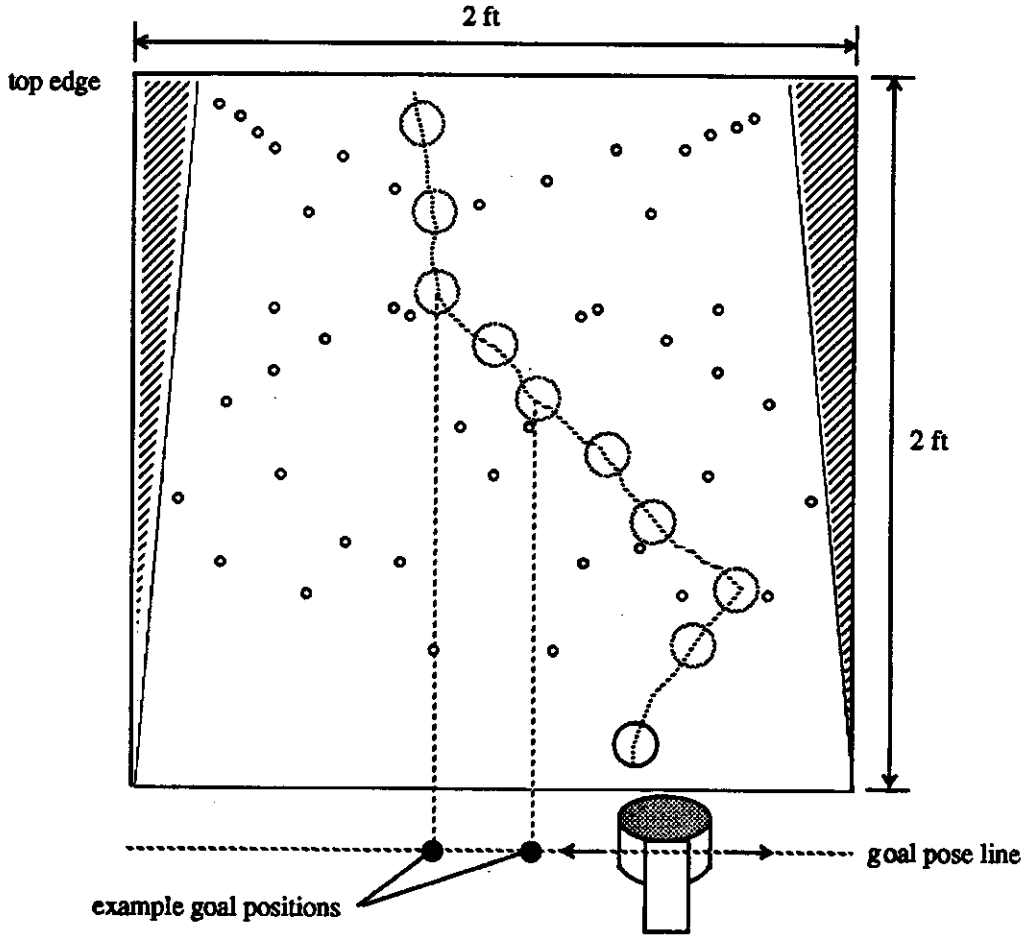


Figure 7. Example Board Layout.

until it reaches a maximum value as the ball rolls off the board.

Subjectively, this means that when the ball is near the top of the board, the robot will not try very hard to get to the goal position. As the ball rolls down the board, the robot increases the effort made to track the ball. The reasoning behind this approach is that initially, the error between the robot position and the goal position may be quite large, and there is much uncertainty about where the ball will actually end up, so it is undesirable to have the robot try to go there as fast as possible. The robot will move *toward* the goal, however, gradually reducing the tracking error, which in turn allows a higher gain to be used.

Projecting the expected ball position onto the goal pose line gives only a 3-dimensional position goal. To determine the desired orientation, a nominal goal orientation is defined with respect to the board. To this orientation is added a pivoting rotation which acts about the direction parallel to the board normal. The magnitude of the pivoting rotation is proportional (up to a limit) to the error between the goal position and the current robot position. This acts to point the cup in the direction of robot motion, and tends to reduce the amount of motion required of the larger robot

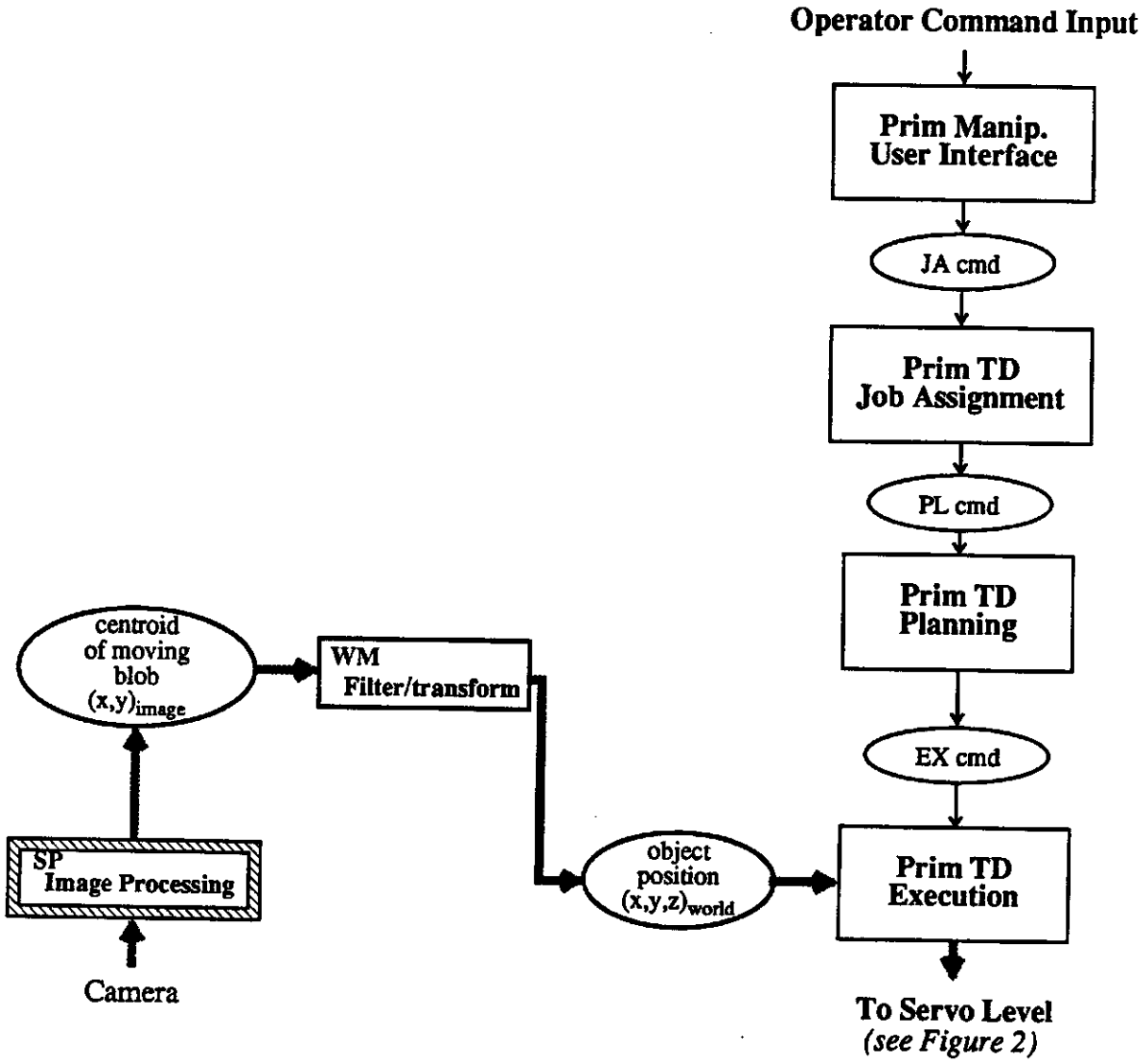


Figure 8. Primitive Level Processes.

joints.

For this trajectory algorithm, the Primitive Planning process sends parameters such as the minimum and maximum gains, maximum Cartesian force and torque, nominal goal orientation, and maximum pivot rotation to the Execution process. Appropriate values for these parameters were determined through experimentation. The plan for this task describes not the desired manipulator position as a function of time, but rather the gains to be used as a function of ball-travel down the board. The position and velocity gains for both translation and rotation are effectively increased as a linear function of ball-travel down the board. The Execution process reads the current ball position in world coordinates from a common memory buffer, and uses this information to compute the goal position and orientation, and the gains to send to Servo.

The Primitive Level processes used for ball-catching are shown in Figure 8. The image processing box shown in this diagram is a simplified representation of the actual Sensory Processing processes which compute the image space centroid of the moving ball. See [28] for a detailed description of the implementation of these processes. All other boxes in Figure 8 represent "atomic" processes [7]. The position of the ball in world coordinates is updated every 66.7 ms, although there is a latency of about 103 ms between the start of image acquisition and the availability of the new ball position based on that image. The latency time includes the time it takes to acquire two images of the same field (even or odd), take the difference of the two images, threshold and take cumulative x and y histograms of the result, and transform the image space centroid to world coordinates. This latency is problematic for high-performance visually-directed motions; particularly when, as in this case, the trajectory of the object of interest cannot be predicted accurately from previous position samples. The entire latency period is not compensated for, since this would cause large errors when the ball changes direction as a result of hitting a pin. If no correction is made, however, it is difficult to get the robot to the goal in time to catch the ball. The estimate of the ball position therefore includes a limited prediction of where the ball is going based on a simple linear extrapolation of the two most recent position samples. The equation used to estimate the ball position is

$$P_{est} = P_n + w\,(P_n - P_{n-1})$$

where P_{est} = estimate of current ball position, P_n = most recent position sample, P_{n-1} = previous position sample, and w = prediction factor which determines the amount of prediction to apply. Although other values have been tried, a prediction factor of w = 1 seems to work fairly well. This corresponds to using a constant velocity to reduce the latency by about 67 ms (one update period).

A chute has been attached to the board (not shown in the figures), which allows the robot to put the ball back in play after a catch. After a catch is made, the system switches to a joint space quintic polynomial trajectory algorithm and high-gain joint PD servo control. These algorithms are used to dump the ball in the chute, and then to make a high-speed motion back to the start position before the ball reaches the end of the chute. The start position is near the middle of the bottom edge of the board. There is some freedom in selecting the start position, since the robot will move toward the goal pose line regardless of where it starts from. If the start position is behind the line, the robot will move towards the board as the ball rolls nearer, giving the appearance of aggressively going after the ball. If the start position is too far back, however, the robot frequently overshoots the goal pose line and misses the ball, because of suboptimal gains and the lack of inertia compensation. The approach which was found to work best was to use a start position and a goal pose line that were close to the board. Then, just as the ball rolls off the edge, the robot pulls back by an amount related to how fast the ball is moving. This provides a crude "velocity matching" of the cup to the ball as it is caught.

In practice, the robot is able to catch the ball approximately 80% of the time using this technique. The actual performance of the robot depends on a number of factors, including the accuracy of the board and camera calibrations. In cases where the ball had a high horizontal velocity, or radically changed direction near the bottom of the board, the robot was simply not fast enough to get to the ball in time.

Of course, the gain scheduling approach is not the only way to control a robot for this task. In fact, it was not the first approach tried. The initial approach was to scale the position error to stay

within maximum acceleration and velocity constraints, and use this scaled value to determine the goal position. Large gains would be used for the entire motion, and the ball would be tracked as closely as the limits on acceleration and velocity would allow. This approach did not work very well, however. The acceleration constraint scaled the error down to such an extent that the robot would not move to the new goal position, because it was too close (the torques resulting from the error times the gain were not large enough to overcome joint stiction). Since the velocity remained zero, the error always remained small due to acceleration limiting, and the robot never moved.

Although further experiments could have been performed with the acceleration-limiting approach, trying things like making sure the new goal position was at least some minimum distance from the current position, it was decided instead to look at the gain scheduling approach. One reason for this decision was that the gain scheduling would allow the gains to be gradually increased until stiction was overcome and the robot would start moving. The robot does in fact start moving quite smoothly using gain scheduling. Of course, near the bottom of the board the motion becomes jerky if the ball changes direction rapidly, since the ball must be tracked closely as it nears the edge of the board.

Another possible approach is to continually replan a conventional trajectory (such as a quintic polynomial) that will take the robot from its current state to the estimated goal state (catch position), as in [27]. When the ball is at the top of the board, the estimated goal state and time-to-catch would not be correct, but only a few cycles of this plan would be executed. Subsequent plans would provide increasingly accurate estimates of the final catch position and time. Gains would remain constant throughout the motion. This approach has the advantage of providing direct control over the smoothness of the trajectory, and will be investigated in future experiments.

The Jacobian-transpose servo algorithm works well in this application. It allows the robot to be commanded directly in a Cartesian reference frame relevant to the task, and eliminates the need to perform inverse kinematics. It enables one to think about the product of servo errors and gains in terms of Cartesian forces acting on the end effector. Perhaps most importantly for this application, it provides the capability to manipulate the gains directly to reflect the time-varying accuracy requirements of the task. The performance is not quite as good as it could be, due to frictional and inertial effects, but this situation is expected to improve as more complete compensation is provided.

5. Conclusions

The details of an implementation of a Jacobian-transpose servo algorithm in a multiprocessing robot control system have been presented. This algorithm has been useful in laboratory applications. The use of the algorithm for two of these applications, stable contact motions with a stiff environment and catching a rolling ping-pong ball, has also been discussed.

The Jacobian-transpose algorithm provides a means of Cartesian control without requiring explicit inverse solutions to Jacobians or kinematics. The accuracy and speed of the Cartesian motions are determined by the magnitude of the control gains. The nonlinear nature of robot manipulator dynamics, along with disturbances such as stiction arising from actuator seals and bearings, degrade the performance of the Cartesian PD control when moderate gains are used. Although there will always be a limit on the size of the gains, a number of approaches to improving the overall performance of the control have been indicated. The pursuit of several of these approaches is planned in the future, including improving the rate at which torque commands and feedback data

are updated.

Still, the Jacobian-transpose algorithm can be used to successfully generate compliant motions useful for contact tasks involved in assembly operations. The algorithm can be used for sensory interactive or other tasks where the goal is determined with respect to a Cartesian frame of reference. This would include teleoperation via a Cartesian hand controller.

6. References

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

[2] Brady, M., et al., eds., Robot Motion: Planning and Control, MIT Press, Cambridge, MA, 1982, pp. 242-243.

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

[4] Wavering, A. "Manipulator Primitive Level Task Decomposition," NIST Technical Note 1256, NIST, Gaithersburg, MD, October, 1988.

[5] Kelmar, L. "Manipulator Servo Level World Modeling," NIST Technical Note 1258, NIST, Gaithersburg, MD, March, 1989.

[6] Chaconas, K., Nashman, N., "Visual Perception Processing in a Hierarchical Control System: Level 1," NIST Technical Note 1260, NIST, Gaithersburg, MD, June, 1989.

[7] Fiala, J., "Note on NASREM Implementation," NIST Internal Report 89-4215, NIST, Gaithersburg, MD, December, 1989.

[8] Fiala, J., "A High-Speed Serial Interface to the Robotics Research Corporation Servo Controller," Intelligent Controls Group Internal Document, November, 1989.

[9] Kelmar, L., Lumia, R., "World Modeling for Sensory Interactive Trajectory Generation," Third Int. Symp. on Robotics in Manufacturing, Vancouver, July, 1990.

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

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

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

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

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

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

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

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

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

[19] Koditschek, D. E., "Automatic Planning and Control of Robot Natural Motion Via Feedback," in Adaptive and Learning Systems: Theory and Applications, Narendra, K. S., ed., Plenum Press, N.Y., 1986.

[20] Khosla, P. K., "Choosing Sampling Rates for Robot Control," Technical Report, CMU-RI-TR-87-5, Dept. of Electrical and Computer Engineering, CMU, March, 1987.

[21] Lumia, R., Fiala, J., Wavering, A., "The NASREM Robot Control System and Testbed," Second Int. Symp. on Robotics & Automated Manufacturing, Albuquerque, N.M., November, 1988.

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

[23] Kreutz, K., Long, M., Seraji, H., "Kinematic Functions for the 7 DOF Robotics Research Arm," NASA Conf. on Space Telerobotics, Pasadena, January, 1989.

[24] Seraji, H., "Configuration Control of Redundant Manipulators: Theory and Implementation," IEEE Trans Robotics & Automation, Vol. 4, No. 4, August, 1989.

[25] Wheatley, T. E., "Mapping Processes to Processors for Space Based Robot Systems," IEEE Int. Conf. on Systems Engineering, Dayton, August, 1989.

[26] Michaloski, J. L., Wheatley, T. E., Lumia, R., "Analysis of Computational Parallelism with a Concurrent Hierarchical Robot Control System," Intelligent Controls Group Internal Document, January, 1990.

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

[28] Chaconas, K., Kelmar, L., and Nashman, M., "A NASREM Implementation of Position Determination from Motion, to be published.