Massachusetts Institute of Technology
Artificial Intelligence Laboratory

AIM-549 October 1979

MECHANICAL ARM CONTROL

by

Richard C. Waters

Abstract

This paper discusses three main problems associated with the control of the motion of a mechanical arm.

1) Transformation between different coordinate systems associated with the arm.

2) Calculation of detailed trajectories for the arm to follow.

3) Calculation of the forces which must be applied to the joints of the arm in order to make it move along a specified path.

Each of the above problems is amenable to exact solution. However, the resulting equations are, in general, quite complex and difficult to compute. This paper investigates several methods for speeding up this calculation, and for getting approximate solutions to the equations.

This memo is an updated version of an internal MIT working paper which originally appeared in March, 1973 as MIT Vision Flash 42.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Office of Naval Research under Office of Naval Research contract N00014-77-C-0389 and in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643.

I. Introduction

This paper is concerned with the motion control portion of a mechanical arm control system, and the kinds of computations it needs to perform. Figure 1 is a partial schematic of the flow of information in a robot system containing an arm. The solid arrows in the figure indicate the principle kinds of information used by the motion controller.

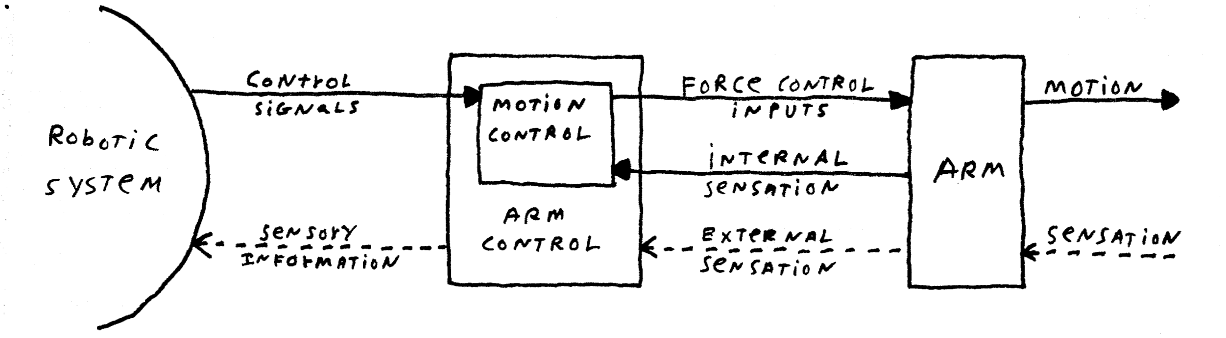

Figure 1: The flow of information in an arm control system.

In this discussion it is assumed that the mechanical arm is made up of a series of links connected by one degree of freedom joints. Further, it is assumed that the motion of the arm is caused by forces applied to these joints. The <u>force control inputs</u> to the arm control the magnitude of these forces.

Typically an arm will have internal sensors which measure the position (and perhaps the velocity) of each joint. The outputs of these sensors make up the <u>internal sensation</u> information coming from the arm to the motion controller. A generalized coordinate system based on these values is a "natural" way for the motion controller to represent the state of the arm.

The <u>control signals</u> from the rest of the robot system to the motion control system are not constrained by the way in which the arm is constructed, but rather by the fact that they need to be concise and convenient from the point of view of the higher levels of the robotic system. This means that in general these signals will be neither complete specifications of the path of an arm motion, nor expressed in the coordinate system natural to the arm. Rather, they will contain only essential information about the motion, expressed in a coordinate system which is easily usable by the rest of the robot system.

In order to control the motion of the arm, the motion controller must take these control signals along with feedback from the arm's internal sensors and produce signals to the force transducers in the arm. This paper discusses three main aspects of this process:

A) <u>Coordinate Transformation</u>: Section II develops methods which can be used to transform between joint oriented generalized coordinates for arm position, and other coordinate systems.

B) <u>Trajectory Calculation</u>: The motion controller must develop a completely specified detailed trajectory from the partial information in the external control signals. This problem is discussed in Section III.

C) Force Calculation: Once a path has been selected, the arm must be moved along this path. To do this the arm controller needs to calculate what forces must be applied to the joints in order to obtain the required accelerations. Section IV develops a set of equations for doing this.

Each of the above problems is amenable to exact solution. However, the resulting equations are, in general, quite complex and difficult to compute. This paper explores three ways for dealing with this problem. First, it is sometimes possible to express the equations in a form which makes it possible to calculate them rapidly. Second, there are a variety of methods available for obtaining approximate solutions to the equations. Third, it is sometimes possible to design the arm in such a way that the equations are simplified.

II. Describing the State of a Mechanical Arm

In order to facilitate analytical treatment of the arm, the first part of this section develops a mathematical method for describing the joints and links in a mechanical arm and the relative orientation of successive links. This gives precise meaning to the notion of a generalized coordinate system based on the positions of the joints. The remainder of the section discusses methods for converting from one coordinate system to another.

Describing the Relationships Between the Joints and Links In an Arm

Following the development in [Uicker 1965] the links in a mechanical arm are numbered, starting with the reference frame as link 0 and proceeding outward with links 1, 2, ... ,n. The joints are numbered correspondingly, with joint i connecting link i-1 and link i. For example, joint 1 connects link 0 to link 1.

Associated with each link is an orthogonal coordinate system fixed in the link (see Figure 2). The axes of the coordinate system for link i are chosen as follows:

Z_i: lies along the axis of joint i+1. It can be chosen to go in either of two directions. (The choice of Z_n is arbitrary.)

X_i: lies along the common normal between Z_{i-1} and Z_i. Its direction is chosen to go from Z_{i-1} to Z_i. (X_0 can be chosen freely as any vector normal to Z_0.)

Y_i: completes a right-handed coordinate system whose origin is the point where the X_i and Z_i axes intersect.

Given the constraints above, the relative position of two adjacent links is completely described by four parameters as follows (see Figure 2):

a_i: the distance between the origins of the coordinate systems i-1 and i measured along X_i. (This parameter is a fixed quantity determined by the geometry of link i.)

s_i: the distance between the origins of the coordinate systems i-1 and i measured along Z_{i-1}. (If joint i is a linear joint then s_i varies during the motion of the joint and is called the joint variable. Otherwise, it is a fixed quantity determined by the geometry of link i.)

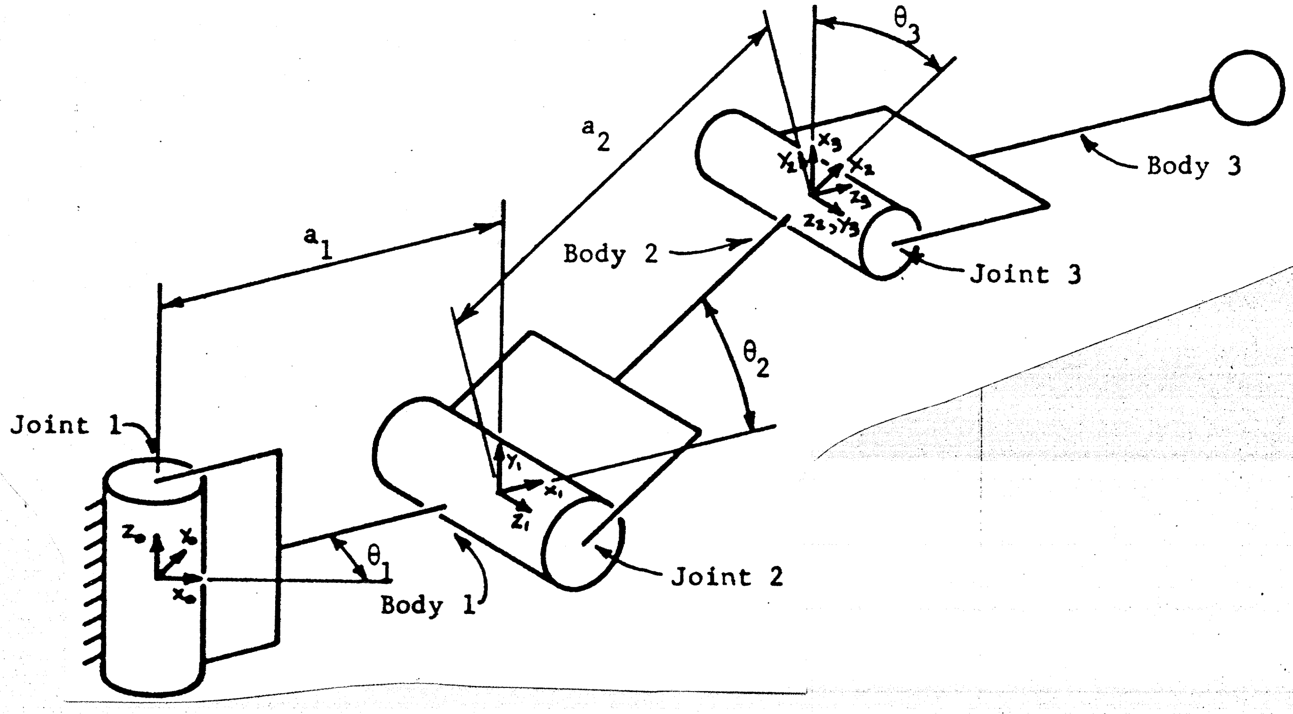

Figure 2: Schematic of a three joint arm.

b_i: the angle between the Z_{i-1} and Z_i axes measured in a right-handed sense about X_i. (This parameter is a fixed quantity determined by the geometry of link i.)

θ_i: the angle between X_{i-1} and X_i measured in a right-handed sense about Z_{i-1}. (If joint i is a rotary joint then θ_i varies during the motion of the joint and is called the joint variable. Otherwise, it is a fixed quantity determined by the geometry of link i.)

The symbol q_i is used to refer to the joint variable no matter whether it is s_i or θ_i. The column vector Q of generalized coordinates

$$Q = (q_1 \ q_2 \ \cdots \ q_n)$$

completely specifies the position of the arm at a given moment.

Transforming Between Link Coordinate Systems

In order to make it easier to state the equations which transform between the different coordinate systems associated with the arm, the position of a point p in link j, as measured in the coordinate system fixed in link i, will by represented by the four element column vector ${}^iR_j$:

$${}^iR_j = (1 \ x \ y \ z)$$

where x, y, and z are the coordinates of point p (in link j) measured in coordinate system i. For convenience, ${}^jR_j$ (the coordinates of p measured in the coordinate system of the link it is in) is abbreviated as r_j, and ${}^0R_j$ (the coordinates of p measured in the fixed reference frame coordinates of link 0) is abbreviated as R_j.

The coordinate systems of adjacent links are related by the following matrix equation:

$${}^{i-1}R_j = A_i\,{}^iR_j$$

where A_i is the following 4x4 transformation matrix derived from a_i, s_i, b_i, and θ_i:

$$A_i = \begin{pmatrix} 1 & 0 & 0 & 0 \\ a_i\cos\theta_i & \cos\theta_i & -\sin\theta_i\cos b_i & \sin\theta_i\sin b_i \\ a_i\sin\theta_i & \sin\theta_i & \cos\theta_i\cos b_i & -\sin\theta_i\sin b_i \\ s_i & 0 & \sin b_i & \cos b_i \end{pmatrix}$$

Note that A_i is a function of the joint variable q_i (θ_i or s_i) and that therefore it will change with time as the joint moves. As a result, ${}^{i-1}R_i$ will in general change with time even though, since point p is fixed in link i, r_i will not change.

The transformation process can be extended to non-adjacent coordinate systems as follows:

$${}^iR_k = {}^iW_j\,{}^jR_k \qquad [i \le j \le k]$$

where ${}^iW_j$ is the 4x4 transformation matrix derived by multiplying together A_{i+1} through A_j.

$${}^iW_j = A_{i+1}A_{i+2} \cdots A_{j-1}A_j$$

${}^0W_i$ is abbreviated as W_i so that:

$$R_i = W_i r_i$$

Velocity of the Arm

In the (non-inertial) coordinate system fixed in a link, all of the points in that link have velocity zero. The velocity in the reference frame (link 0) of any point in the arm can be expressed in terms of the rate of change of the joint variables by differentiating the above equation for R_i.

$$\dot{R}_i = \frac{d(W_i r_i)}{dt} = \frac{dW_i}{dt} r_i = V_i r_i \qquad \left[\frac{dr_i}{dt} = 0\right]$$

$$\text{where } V_i = \frac{dW_i}{dt} = \sum_{j=1}^{n} \frac{\partial W_i}{\partial q_j}\frac{dq_j}{dt} = \sum_{j=1}^{i} U_{ij}\dot{q}_j \qquad \left[\frac{\partial W_i}{\partial q_j} = 0 \text{ if } i<j\right]$$

$$\text{where } U_{ij} = \frac{\partial W_i}{\partial q_j} = W_{j-1}\frac{\partial A_j}{\partial q_j}\,{}^jW_i \qquad \left[\frac{\partial A_i}{\partial q_j} = 0 \text{ if } i \ne j\right]$$

Hand Oriented Coordinates

Since the sensors in a mechanical arm typically directly measure the joint variables q_i, the generalized coordinates are a natural way to specify the position and velocity of the arm. However, from the point of view of a system using the arm, this specification is not very useful. Some specification (such as the hand oriented coordinates presented here) in terms of the cartesian coordinate system fixed in the reference frame the arm is attached to is much more likely to be useful. The motion controller has to be able to convert between these specifications.

In order to make the following discussion more concrete, it is based on a specific arm design, the Scheinman arm [Scheinman 1969] (see Figure 3). There are many convenient systems for specifying the position and velocity of a 6 degree of freedom arm such as this one. The one considered here (see Figure 4), is referred to as "hand oriented". In this system, the state of the arm is described by the position, orientation, velocity, and angular velocity of the hand as follows:

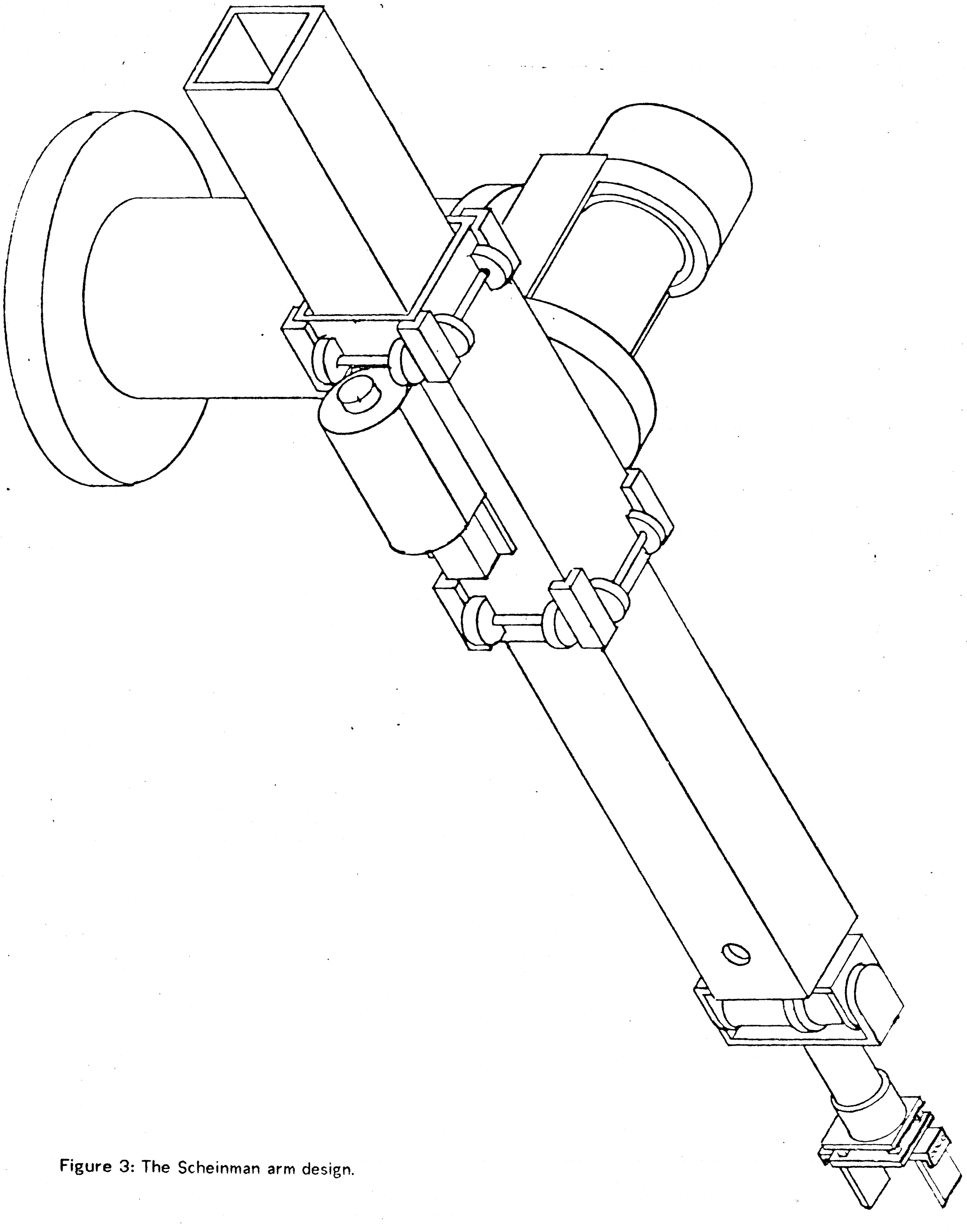

Figure 3: The Scheinman arm design.

P_H: the position vector, gives the coordinates, in the reference frame, for the point H midway between the finger tips of the hand.

α: the orientation angle vector, specifies the angles between the axes of the reference frame ($\emptyset$) and the axes of a frame (H) fixed in the hand with origin at H as follows:

α_x is the angle between the $Y_\emptyset$ and Y_H axes measured about $X_\emptyset$.
α_y is the angle between the $Z_\emptyset$ and Z_H axes measured about $Y_\emptyset$.
α_z is the angle between the $X_\emptyset$ and X_H axes measured about $Z_\emptyset$.

$\dot{P}_H$: is the velocity of point H.

$\dot{\alpha}$: is the angular velocity of the hand about H.

Together these four vectors give 12 parameters that specify the position and velocity of the hand exactly, just as the 6 q_i and their 6 derivatives $\dot{q}_i$ do.

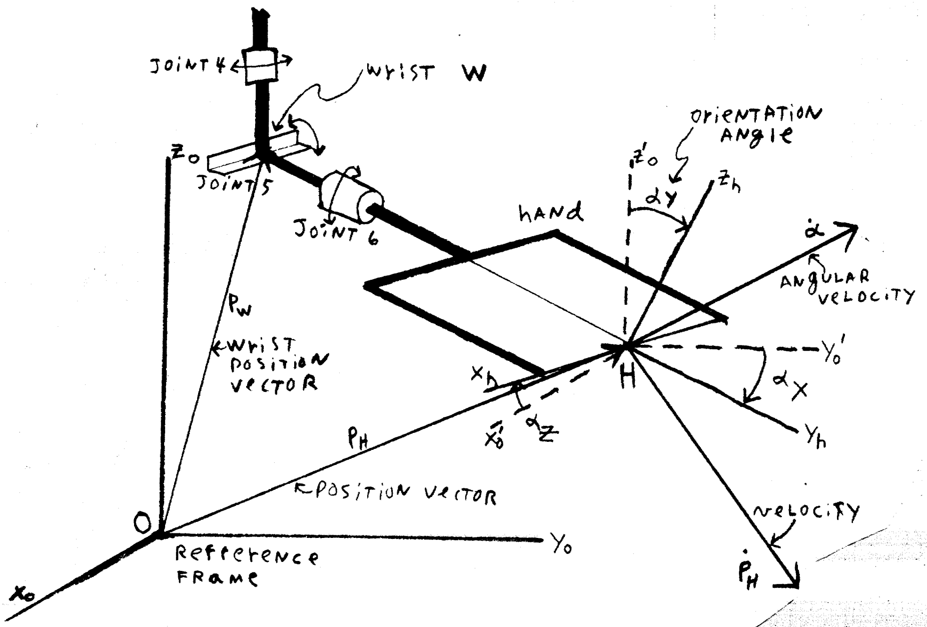

Figure 4: Hand oriented coordinates for the Scheinman arm.

Converting Generalized to Hand Oriented Coordinates

In order to transform from generalized coordinates to the hand oriented coordinates presented above, one need only know h_6 (the position of point H expressed in the coordinates of the hand) and h'_6 (the position of another point in the hand) along with W_6 and V_6 (which are functions of q_i and $\dot{q}_i$ as shown above) and then calculate:

$$P_H = W_6 h_6$$
$$\dot{P}_H = V_6 h_6$$
$$\dot{\alpha} = V_6 h'_6 \times W_6 (h'_6 - h_6) \quad \text{[X: is the cross product]}$$

The orientation angles α can be calculated from the coordinates of three non-colinear points in the hand. The transformations above are cumbersome, but not excessively so.

Converting From Hand Oriented to Generalized Coordinates

The reverse transformation from hand oriented coordinates to generalized coordinates is considerably more difficult. The process outlined above can be inverted. Unfortunately, the resulting equations are in general very complex because W_6 and V_6 are complex functions of q_i and $\dot{q}_i$. (It should be noted that there may be more than one set of generalized coordinates corresponding to a given hand position.)

First, consider the problem of calculating the q_i corresponding to P_H and α. This problem is discussed by Pieper [1968] and is quite complex in the general case. However, with an arm like Scheinman's where the axes of the last three joints intersect, the problem can be divided into two much simpler ones.

Referring to Figure 4, note that it is possible to calculate the wrist position vector P_W directly from P_H and α without calculating any of the values of the q_i. This is possible because the distance between W and H does not depend on any of the q_i and because the orientation of the vector (P_W-P_H) depends only on α. (This in turn is true because joints 5 and 6 are rotary joints whose axes intersect, and because point W was chosen to be at the intersection of these axes, while point H was chosen to be on the axis of joint 6.) Further note that the value of q_4 does not affect the position of W. (This is because the axis of joint 4 also passes through W.)

These facts can be used to decouple the transformation problem as follows: First, q_1, q_2, and q_3 can be calculated separately based on P_W alone. (Note that these coordinates are nearly the spherical coordinates for W. They would be if the axes of the first three joints intersected.) Second, q_4, q_5, and q_6 can be calculated from α and the orientation of links 3 and 4. This is an example of the fact that an arm can be designed in such a way that the calculations which need to be performed are greatly simplified.

Now consider the problem of calculating the $\dot{q}_i$ from $\dot{P}_H$ and $\dot{\alpha}$. Unfortunately, it is not possible to break this problem up into subproblems. However, Whitney [1972] has developed a method for finding the $\dot{q}_i$ based on the equation:

$$\dot{S} = J(Q)\dot{Q}$$

where S, Q, $\dot{S}$, and $\dot{Q}$ are the six element column vectors:

$$S = (P_{H_x}\ P_{H_y}\ P_{H_z}\ \alpha_x\ \alpha_y\ \alpha_z) \qquad Q = (q_1\ \ldots\ q_6)$$

$$\dot{S} = (\dot{P}_{H_x}\ \dot{P}_{H_y}\ \dot{P}_{H_z}\ \dot{\alpha}_x\ \dot{\alpha}_y\ \dot{\alpha}_z) \qquad \dot{Q} = (\dot{q}_1\ \ldots\ \dot{q}_6)$$

and J(Q) is defined by

$$J(Q)_{ij} = \frac{\partial S_i}{\partial q_j}$$

It turns out that the partial derivatives are easy to evaluate, and that therefore it is easy to calculate J(Q). $\dot{Q}$ can then be calculated by inverting J(Q):

$$\dot{Q} = J(Q)^{-1}\dot{S}$$

Unfortunately, inverting a 6x6 matrix is relatively time consuming, and since the values of $\dot{Q}$ calculated are accurate only at the one point Q, J(Q) has to be reevaluated and reinvert often.

There is a simple approximation method which can be used to calculate $\dot{Q}$. Given a position S and a velocity $\dot{S}$, a new position $S'=S+\dot{S}dt$ can be calculated based on the assumption that $\dot{S}$ will remain constant for some small time dt. The corresponding generalized coordinates Q and Q' can then computed. Finally, $\dot{Q}$ can be estimated by the equation:

$$\dot{Q} = \frac{(Q'-Q)}{dt}$$

[Since the writing of this paper, Horn and Inoue [1974] have treated the transformation problem in much more detail.]

III. Trajectory Selection

Referring back to Figure 1, the primary control signals from the robotic system to the motion controller are requests to move the arm from one configuration to another. For the convenience of the robotic system, each request should be concise, containing no more information than is necessary to state the goal, and constrain the trajectory so that the arm does not strike any obstacles. In order for the robotic system to exercise control over the velocity as well as the position of the arm, these requests must contain specifications of velocity as well as position.

A list of position-velocity configurations the arm is constrained to pass through is a convenient form for a motion request from the robotic system to take. By varying the number and spacing of the configurations in the request, the robotic system can vary the amount of control it exercises over the exact details of the trajectory. Additional flexibility can be gained by allowing some of the velocities to be left unspecified. For instance, a request to move the arm from position A to position B while avoiding an obstacle might specify that the arm should pass through a point C, and come to rest at B but not specify what velocity the arm should have at C.

Criteria For a Good Trajectory

The motion controller must expand the trajectory description it receives into a completely specified trajectory. There are several requirements which the resulting detailed trajectory should satisfy.

<u>Closeness to intended trajectory</u>: The most important criterion is that the trajectory selected must be close to the one intended by the robotic system. In particular it must be close enough so that the arm does not strike any obstacles when it moves. The essence of this requirement boils down to three subcriteria:

1) There must be some easy to compute algorithm known to both the motion controller and the robotic system which specifies what the intended trajectory is. For example, the intended trajectory might be taken as the piecewise linear trajectory through the positions specified. Alternatively, if the algorithm used by the motion controller is simple enough, it could be used by the robotic system when planning the trajectory.

2) There should be some agreed upon notion of closeness. For example, requiring that the actual trajectory not make any excursions outside of an envelope centered on the intended one.

3) There must be some way for the robotic system to exercise more precise control over the motion when it has to in order to maneuver the arm in a tight spot. To this end, the trajectory selection algorithm used by the motion controller should have the

property that the difference between the selected trajectory and the intended one decreases when the points in the trajectory description get closer together.

Physical Limitations: The range of motion, velocity, and acceleration of the arm are all bounded by the design of the arm. Further, the physical reality is that all these variables take on continuous values. As a result of this, it is not possible for the arm to actually follow the piecewise linear trajectory through the points. The robotic system must be aware of these limitations, and not ask for impossible motions.

Optimal Time: All things being equal, the fastest trajectory through the requested points should be selected. However, there are two problems with this. First, the fastest trajectory between two points may be quite different from the intended one. Second, it is not easy to compute what the optimal trajectory is. In order to get from one point to another in minimal time, one must use bang bang control (where the force on each joint is either zero, maximally positive, or maximally negative at each instant of time). With this kind of control, the trajectory is completely specified by the points at which the forces are switched. Unfortunately, in order to find the switching points, a set of partial differential equations must be solved (see Kahn [1969]). The key problem is that the relationship between force and acceleration (see the next section) varies depending on what the position and velocity of the arm is. Kahn develops some approximate methods for determining the switching points. However, they are still too complex for use in a real time application.

Computational limitations: The detailed trajectory should be easy to compute and concisely representable. In particular, it should be possible to compute the trajectory in real time.

Constant Maximum Acceleration Trajectories

A simple trajectory calculation method results from applying bang bang control while assuming that the maximum acceleration of the arm is a constant independent of the position and velocity of the arm. This constant value must be chosen so that it can always be achieved. This method produces suboptimal trajectories because it ignores the fact that larger accelerations are possible in most positions.

The algorithm for finding the points at which the acceleration should be switched is based on representing a trajectory for the arm as a curve in phase space. Phase space for a 6 link arm has 12 dimensions: 6 for the joint variables, and 6 for their derivatives. The method is illustrated in Figure 5. The figure is a phase space graph of trajectories for a one link arm. The vertical axis is made proportional to the square root of the velocity so that trajectories with constant acceleration become straight lines. The box around the figure signifies that the position and velocity are both bounded by: $-1 \leq x \leq 9$ and $-3 \leq \dot{x} \leq 3$. It is assumed that the acceleration is bounded by $-\frac{1}{2} \leq \ddot{x} \leq \frac{1}{2}$.

Referring to the figure, assume that the arm is at point A (position 1, velocity 1) and that the robot system has requested that the motion controller move the arm to point B. Any trajectory leaving A must go out through the horizontally shaded region, and any trajectory approaching B must go through the vertically shaded region (arrows on the trajectories indicate direction of travel). Such a trajectory is A-E-F-B. The acceleration changes at E, there is no trajectory with constant acceleration from A to B. (There is no change of acceleration at F.) Under the assumption of constant maximal acceleration, The time optimal trajectory is A-C-D-B. This trajectory accelerates from A to C and then decelerates from C to B through D.

It can be shown that the optimal trajectory between any two points has at most one place

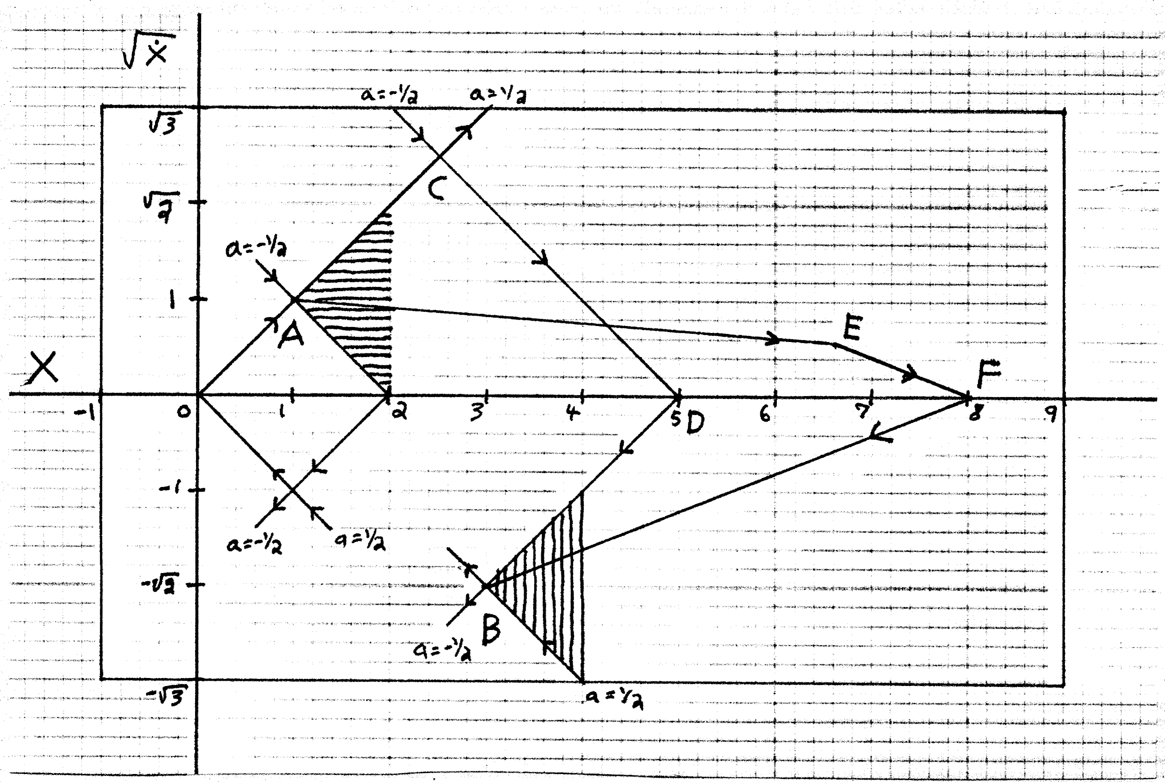

Figure 5: Phase space graph for trajectories.

where the acceleration changes. As a result, this change point, together with the end points completely specifies the trajectory. The problem of finding the change point can be reduced to finding where straight lines intersect on the graph. Using points A and B as an example, the optimal trajectory must leave A on one of the two extremal acceleration lines. Similarly it must enter B on one of the two extremal acceleration lines terminating on B. The optimal trajectory can be found by seeing which of the extremal lines leaving A intersect which of the extremal lines entering B. (There can be two intersecting pairs.) The algorithm can be easily extended to arms with more than one joint. The trajectory can be found for each joint separately, and then modified to synchronize the motions with each other by adding segments of zero acceleration into the motions of individual joints.

This method of calculating a detailed trajectory meets the criteria set forth above very well. It is fast to calculate, and the trajectory can be concisely represented by the acceleration switching points. The trajectories produced are reasonably fast, and do not violate any of the physical limitations of the arm. The only problem with the trajectories produced is that they are not very close to piecewise linear trajectories through the requested points. However, they do have the property that as the points get closer together, they do get closer to the piecewise linear paths. Further, the trajectory planning algorithm is simple enough that it could be used as the definition of intended trajectory. [Since the writing of this paper, this approach to trajectory calculation was used in the robot control system described in [Waters 1974].]

IV. Dynamics

Once a trajectory has been determined, the arm has to be guided along the trajectory. At each time interval, it is easy to determine, from the position and velocity of the joints, what accelerations should be applied to the joints so that the arm will be on the trajectory at the next time interval. However, the motors which move the joints produce forces not accelerations. The controller must determine what force F_i should be applied to joint i in order to produce the required acceleration $\ddot{q}_i$. Due to the way the joints are interconnected, the force which needs to be applied at a given joint depends not just on the acceleration desired at that joint, but also on what is happening at the rest of the joints. One way to take this interaction into account is to use the equations of motion for the arm in order to calculate the forces. It should be noted that the equations of motion developed below are based on an idealized model of the arm, and that therefore they do not model the arm exactly. For example, they ignore the forces generated by friction in the joints. If the friction forces are large, then the dynamic equations alone may not be adequate to control the arm.

The Lagrangian Equations of Motion

Following the development of Uicker [1965] as presented in Kahn [1969], the n equations of motion for the arm can be obtained through the use of Lagrange's equations for a nonconservative system.

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = F_i \qquad i=1,2, \ldots ,n$$

where the Lagrangian L is the difference between the kinetic and potential energies.

$$L = K - P$$

The kinetic energy of a particle of infinitesimal mass dm on link i having coordinates $R_i = W_i r_i = (1\ X_i\ Y_i\ Z_i)$ is:

$$dk_i = \frac{1}{2}(\dot{X}_i^2+\dot{Y}_i^2+\dot{Z}_i^2)\,dm$$

or $$dk_i = \frac{1}{2}\mathrm{tr}(V_i r_i r_i^T V_i^T)\,dm \qquad \text{[tr: is the trace]}$$

where $$V_i r_i = \frac{dW_i}{dt} r_i = \left(\sum_{j=1}^{i} U_{ij}\dot{q}_j\right) r_i \qquad \text{[see Section II]}$$

The total kinetic energy of the link can be found by integrating over the mass of the link.

$$k_i = \frac{1}{2}\int \mathrm{tr}(V_i r_i r_i^T V_i^T)\,dm = \frac{1}{2}\mathrm{tr}(V_i \int (r_i r_i^T)\,dm V_i^T) = \frac{1}{2}\mathrm{tr}(V_i J_i V_i^T)$$

V_i is independent of dm allowing the integral to be shifted inward. The inner integral can be precalculated due to the fact that r_i does not depend on q_i or $\dot{q}_i$ and the fact that the mass distributions of the links in the arm do not change with time (with the notable exception of the hand, see below). The result, the inertia tensor (designated J_i), is a complete description of the inertial properties of the link.

$$J_i = m_i \begin{pmatrix} 1 & \bar{x}_i & \bar{y}_i & \bar{z}_i \\ \bar{x}_i & \frac{1}{2}(-k_{i11}+k_{i22}+k_{i33}) & k_{i12} & k_{i13} \\ \bar{y}_i & k_{i12} & \frac{1}{2}(k_{i11}-k_{i22}+k_{i33}) & k_{i23} \\ \bar{z}_i & k_{i13} & k_{i23} & \frac{1}{2}(k_{i11}+k_{i22}-k_{i33}) \end{pmatrix}$$

Where m_i is the mass of the link, $(1\ \bar{x}_i\ \bar{y}_i\ \bar{z}_i) = \bar{r}_i$ is the position of the center of mass, and k_{ijk} is the square of the jkth radius of gyration about the origin of coordinates in link i.

When the hand picks up something, its inertial characteristics change, particularly if the object picked up is either heavy or long. In this formulation, it is possible to take account of the object in the hand by changing the J matrix for the hand. This could be done in several ways. First, a priori knowledge of the object's inertial characteristics could be used to alter J_{hand}. Second, partial information about the object such as its mass and longest dimension could be used to make an approximate correction to J_{hand}. For instance, if the hand picks up a block which has relatively small dimensions, then probably only its mass has to be taken into account. If the mass is also small the the object can probably be ignored. Third, the arm system could be used to discover the inertial characteristics of the object it is holding. This could be done by applying small canonical forces to the joints near the hand, and observing the accelerations produced. These experiments could then be used to estimate the components of J_{hand}.

The total kinetic energy of the arm is given by:

$$K = \frac{1}{2}\sum_{i=1}^{n} \mathrm{tr}(V_i J_i V_i^T)$$

The potential energy of the system is equal to the sum of the work required to displace the center of mass of each link from a horizontal reference plane.

$$P = \bar{P} - \sum_{i-1}^{n} m_i G W_i \bar{r}_i$$

$$\text{where } G = (0\ G_x\ G_y\ G_z)$$

The row vector G is the acceleration due to gravity, $\bar{r}_i$ is the position of the center of mass of link i, and $\bar{P}$ is a constant which depends on the particular reference plane chosen.

Performing the differentiations in Lagrange's equations and simplifying gives the following n equations of motion:

$$F_i = \sum_{j=i}^{n} (\mathrm{tr}(U_{ji} J_j (P_j + L_j)^T) - G_{ji})$$

where:

$P_j = \sum_{k=1}^{j} U_{jk}\ddot{q}_k$: The reaction forces due to the inertial properties of the links.

$L_j = \sum_{k=1}^{j}\sum_{l=1}^{j} U_{jkl}\dot{q}_k\dot{q}_l$: The forces produced by the interaction of the link velocities.

$G_{ji} = m_j G U_{ji} \bar{r}_j$: The forces due to gravity.

$U_{ij} = \frac{\partial W_i}{\partial q_j}$: The effect of the motion of one joint on the position of another.

$U_{ijk} = \frac{\partial U_{ij}}{\partial q_k} = \frac{\partial^2 W_i}{\partial q_j \partial q_k}$: The interaction of joint motions.

A_i, $\frac{\partial A_i}{\partial q_i}$, $\frac{\partial^2 A_i}{\partial q_i^2}$: The fundamental effects of moving a joint.

G, J_j, m_j, $\bar{r}_j$: The static properties of the arm.

If the equations are evaluated straightforwardly just as written they require an inordinate amount of computation. The dominant computational cost comes from doing 4x4 matrix multiplications each of which requires 4^3=64 scalar multiplications. A secondary cost comes from doing 4x4 matrix additions and other operations of order 4^2=16 (such as multiplying a scalar times a matrix, multiplying a vector times a matrix, and multiplying two matrices where only the trace of the answer is required). Lower order operations such as adding scalars will be ignored in the discussion below. For a 6 link arm, a naive calculation of the forces requires 2254 matrix multiplications and 973 order 4^2 operations. (For convenience, this kind of calculation estimate will be abbreviated as 2254m+973a in the discussion below.) This would take 8 seconds in floating point on a PDP/11-45.

Methods For Speeding Up the Calculations

This section describes five approaches to speeding up the calculation of the required forces: reformulating the equations so that they can be evaluated faster, simplifying the equations by eliminating terms, using direct acceleration servoing of the arm, using extrapolation and interpolation, and using precomputed values of the equations.

Reformulating the Equations More Efficiently

There is a large amount of structure in the equations which can be used to speed up their evaluation by several orders of magnitude. The key insight is that there are many subexpressions which appear over and over again during the computation. To start with, in a naive evaluation of the equations U_{66} is computed 12 times and L_6 is computed 6 times. If the values P_j, L_j, G_{ji}, U_{ij}, and U_{ijk} are recorded when they are evaluated so that they are never evaluated more than once, then the computation is reduced to 486m+275a, a savings by a factor of 5.

Deeper levels of repetitiveness are illustrated by the calculation of the U_{ij}. Section II showed that:

$$U_{ij} = W_{j-1} \frac{\partial A_j}{\partial q_j} {}^jW_i \qquad i \geq j$$

The fact that $U_{ij} = 0$ when i<j has already been used to simplify the equations of motion by making the upper limits of the inner sums j instead of n and the lower limit of the outer sum i instead of 1 so that only instances of U_{ij} where i≥j appear. These reductions in range save a great deal of useless computation. The formula above in conjunction with the formula for ${}^jW_i$ can be used to derive the following recurrence relations:

$$ {}^jW_i = {}^jW_{i-1}A_i \qquad {}^iW_i = I \text{ [the identity matrix]}$$
$$ U_{ii} = W_{i-1} \frac{\partial A_i}{\partial q_i}$$
$$ U_{ij} = U_{i-1\,j}A_i \qquad i>j$$

The key advantage of this formulation is that it makes it possible to compute all of the $\frac{n(n+1)}{2}$ U_{ij} in the same amount if time it takes to compute the n U_{nj}.

The recurrences above can be used to develop a recurrence relation for P_j as follows:

$$\begin{aligned} P_j &= \sum_{k=1}^{j} U_{jk}\ddot{q}_k \\ &= (\sum_{k=1}^{j-1} U_{j-1k}\ddot{q}_k)A_j + U_{jj}\ddot{q}_j \\ &= P_{j-1}A_j + U_{jj}\ddot{q}_j \\ P_1 &= U_{11}\ddot{q}_1 \end{aligned}$$

A similar approach can be taken to the calculation of U_{ijk} and L_j. Since $U_{ijk}=0$ if i<j or i<k and $\frac{\partial A_i}{\partial q_j} = 0$ if $i\neq j$ the U_{ijk} have the following typical form:

$$U_{ijk} = \frac{\partial}{\partial q_k}(W_{j-1}\frac{\partial A_j}{\partial q_j}\,{}^jW_i) = W_{j-1}\frac{\partial A_j}{\partial q_j}\,{}^jW_{k-1}\frac{\partial A_k}{\partial q_k}\,{}^kW_i \qquad j<k<i$$

The following recurrence relations can be derived for the U_{ijk}:

$$\begin{aligned} U_{ijk} &= U_{ikj} \\ U_{iii} &= W_{i-1}\frac{\partial^2 A_i}{\partial q_i^2} \\ U_{iji} &= U_{i-1j}\frac{\partial A_i}{\partial q_i} && i>j \\ U_{ijk} &= U_{i-1jk}A_i && i>k \text{ and } i>j \end{aligned}$$

These can be used to get recurrence relations for L_j.

$$\begin{aligned} L_j &= \sum_{k=1}^{j}\sum_{l=1}^{j} U_{jkl}\dot{q}_k\dot{q}_l \\ &= (\sum_{k=1}^{j-1}\sum_{l=1}^{j-1} U_{j-1kl}\dot{q}_k\dot{q}_l)A_j + \sum_{k=1}^{j-1} U_{jkj}\dot{q}_j\dot{q}_k + \sum_{l=1}^{j-1} U_{jjl}\dot{q}_l\dot{q}_j + U_{jjj}\dot{q}_j^2 \\ &= L_{j-1}A_j + 2(\sum_{k=1}^{j-1} U_{j-1k}\dot{q}_k)\frac{\partial A_j}{\partial q_j}\dot{q}_j + U_{jjj}\dot{q}_j^2 \\ L_1 &= U_{111}\dot{q}_1^2 \end{aligned}$$

Note that in this formulation, only the U_{jjj} have to be calculated. The other U_{ijk} do not need to be calculated at all. The sum in the middle term is the quantity V_{j-1} introduced in Section II. It also has a recurrence relation.

$$\begin{aligned} V_j &= \sum_{k=1}^{j} U_{jk}\dot{q}_k \\ &= V_{j-1}A_j + U_{jj}\dot{q}_j \\ V_1 &= U_{11}\dot{q}_1 \end{aligned}$$

Using these relations the forces can be computed using only 70m+106a. This is an additional improvement by a factor of 6. This reduces the computation time to around .25 second.

[Since the writing of this paper, Luh [1978], working with the Newton-Euler equations of motion, came up with an additional recurrence relation. Hollerbach [1979] has applied this relation to the Lagrangian formulation used above. The new recurrence relation is based on the quantity D_i:

$$\begin{aligned}
F_i &= tr(U_{ii}D_i) - \sum_{j=i}^{n} G_{ji} \\
D_i &= \sum_{j=i}^{n} {}^iW_j J_j (P_j+L_j)^T \\
&= A_{i+1}\Big(\sum_{j=i+1}^{n} {}^{i+1}W_j J_j (P_j+L_j)^T\Big) + J_i(P_i+L_i)^T \\
&= A_{i+1}D_{i+1} + J_i(P_i+L_i)^T \\
D_n &= J_n(P_n+L_n)^T
\end{aligned}$$

This reduces the computation to 60m+81a. The reason the impact is not greater is that the U_{ij} $i \neq j$ still need to be calculated in order to calculate G_{ji}. (This was not an issue in the derivations of Luh and Hollerbach because they chose to ignore the gravity forces.)]

The technique used by Luh and Hollerbach to derive a recurrence relation for D_i can be straightforwardly applied in order to derive an anologous recurrence relation for the gravity forces.

$$\begin{aligned}
F_i &= tr(U_{ii}D_i) - GU_{ii}G_i \\
G_i &= \sum_{j=i}^{n} {}^iW_j \bar{r}_j m_j \\
&= A_{i+1}\Big(\sum_{j=i+1}^{n} {}^{i+1}W_j \bar{r}_j m_j\Big) + \bar{r}_i m_i \\
&= A_{i+1}G_{i+1} + \bar{r}_i m_i \\
G_n &= \bar{r}_n m_n
\end{aligned}$$

With this reformulation, the U_{ij} $i \neq j$ no longer have to be calculated at all. The calculation is reduced further to 45m+77a. This gives an overall improvement by a factor of 50. Another aspect of the way the efficiency of the calculation has been increased is that while the effort involved in a naive calculation is proportional to the cube of the number of links, the effort in the improved formulation is linear in the number of links.

Before going on to other methods for improving the calculation consider how this computation is divided between the three basic parts of the computation. The computation needed solely to compute the contribution of the gravity forces is 17a (6%). The computation needed solely to compute the contribution of the second order forces L_j (this includes the computations of V_j and U_{jjj}) is 20m+38a (46%). The remaining computation needed to compute the contribution of the basic inertial forces P_j is 25m+22a (48%).

[Hollerbach [1979] has noted that a further improvement by a factor of 2 can be gained by reformulating the equations using 3x3 matrices instead of 4x4 matrices. The 4x4 matrices A_i are a mathematically convenient way to express the relationship between adjacent coordinate systems, but they are computationally inefficient. The resulting formulation has essentially the same number of matrix multiplies and adds, but each one only takes about half as long.]

It is probably possible to compute the equations in fixed point rather than in floating point. This gives an added improvement by a factor of 4 on a PDP/11-45. Putting all these factors together, the calculation is speeded up 400 times, and the time required is reduced to 20ms on a PDP/11-45. Though it would be nice if the time could be reduced by another order of magnitude, this is really quite fast, and it makes it hard for approximation techniques to compete with complete evaluation.

Simplifying the Dynamic Equations

One way to speed up the evaluation of the equations is to simplify them by omitting some of the terms in the expressions. This can be done in two basic ways. First, the arm can be designed so that some of the terms in the equations vanish. Second, nonvanishing terms can be ignored yielding approximate values.

Counterbalancing: If an arm is counterbalanced in order to eliminate the effects of gravity, then all of the G_i will be zero, and need not be calculated. However, as shown above, the G_i are so easy to compute that almost nothing is saved by omitting them. If the arm is not counterbalanced, then it is usually not possible to ignore the G_i because they are some of the largest forces on the arm.

Joint locking: If the arm is designed so that individual joints can be physically locked with a pin or with friction so that they cannot move, then this can be used to speed up calculation. If locking is used when a joint is to be kept motionless, then the joint is in effect eliminated from the arm. No force has to be calculated for it, and it does not contribute to the forces required at the other joints. If the computation is properly arranged, the links before and after the locked joint can be consolidated into one link, and the calculations can be performed as if the arm had only n-1 links. For example, if three of the links of a six link arm were locked, the calculation would be reduced to 19m+34a, a savings of 57%.

Ignoring L_j: If the arm is moving slowly, the terms L_j will be small and can be ignored. This saves 46% of the calculation required. If the arm is moving rapidly, the L_j can be quite large and cannot be ignored.

Reducing joint interaction: The arm can be designed so that interaction between joints is minimized and some of the U_{ij} become zero. For example, if the first three joints are mutually perpendicular linear joints, then the motion and acceleration of one will not affect the others. Unfortunately, making a few U_{ij} zero will not speed up the computation any because recurrence relations are being used to compute the terms in the equations, and no computation is saved in this process unless the last term in a series can be eliminated. One approximation approach would be to design the arm so that each link is significantly smaller and lighter than the previous link and then ignore all of the effects of a link on the links before it. This would have the effect of eliminating the outermost sum in the equations so that:

$$F_i = tr(U_{ii} J_i (P_i + L_i)^T) - G_{ii}$$

Due to the efficient way in which D_i and G_i are computed, this saves only 5m+5a. As a result, the approximation required is not worth the savings obtained.

Direct Acceleration Servoing

One way to calculate the dynamic equations very rapidly would be to use special purpose hardware such as fast matrix multipliers. This is expensive, but would solve all problems. It is interesting to note that there is one piece of hardware around (the arm itself) which can very rapidly compute the inverse of the equations. Namely, given a particular force some acceleration is produced. This yields a force-acceleration pair satisfying the equations.

The most direct method to utilize this information would be to use high speed servoing to directly control the acceleration of the joints. (The time delay in the servo loop must be quite small in order to keep the error acceptably small.) This approach requires almost no computation and would be very effective. It is inherently better than using the equations above, because the dynamic equations ignore friction and describe the arm only in an idealized sense.

There is however a problem with this approach. In order for it to work, it must be possible to obtain accurate measurements of the actual acceleration over short time intervals. (Note that the dynamic equations do not require that you be able to measure acceleration, but only that you know what acceleration you want.) The most direct way to obtain acceleration measurments would be to put accelerometers on the joints. This is complicated by the fact that the sensors must be arranged so that they measure the angular (rather than linear) acceleration of rotory joints. Further, the measurements must be converted so that they give the acceleration of a joint in terms of the (non-inertial) frame the joint is in, rather than in terms of the reference frame. This would probably require two sets of accelerometers at each joint: one to measure the accleration of the frame, and one to measure the acceleration of the joint.

If accelerometers are not used, then the acceleration must be estimated by using differences in velocity, or second order differences in position. This requires very high accuracy in the velocity (position) measurements. Consider the kind of typical values of position, velocity, and acceleration one might find in a joint. A rotary joint might have a range of motion of 3 radians, a maximum velocity of 1 radian/sec, and a maximum acceleration of 1 radian/sec^2. If the sampling interval is T then the smallest change (A) in acceleration which can be detected by differences in velocity measurements is related to the smallest change (V) in velocity which can be detected by the formula $A=\frac{V}{T}$. Similarly $V=\frac{P}{T}$ where P is the smallest change in position which can be detected.

If T was as large as 10^{-3}sec and only 7 bits of accuracy in acceleration were needed ($A=10^{-2}$), then if accelerometers were not available at the joint, V would have to be equal to 10^{-5} (17 bits of velocity). If tachometers were not available either, then P would have to be equal to 10^{-8} (29 bits of position information). It would be prohibitively expensive to obtain that much accuracy in position measurement, and very difficult to obtain the required accuracy in velocity measurement. As a result, this approach is probably not possible without accelerometers on the joints, and certainly not possible without accurate tachometers.

Extrapolation and Interpolation

Extrapolation and interpolation can be used to compute the forces for one set of parameters to the equation based on the values for other sets of parameters. The values to extrapolate from could come either from some data base of values (see below), from evaluating the equations directly, or from the arm itself. (As discussed above, the arm itself can only be used if it is possible to measure acceleration accurately). The purpose of doing extrapolation is to spread the cost of evaluating the equations over more data points.

In order to get first order accuracy (for a 6 link arm), the partial derivatives of each force with respect to the 18 parameters (q_i, $\dot{q}_i$, and $\ddot{q}_i$) to the equations must be estimated. In order to do this quickly 18 pairs of function values are needed, where only one input varies in each pair. If the values are coming from the recent behavior of the arm or recent evaluations of the equations then such pairs will not be available, and the estimation will have to be based on whatever pairs are available. (At a minimum, 7 different function values are required in order to have 18 pairs of values.) In order to estimate the derivatives using random pairs, 6 18x18 matrices have to be inverted. This approach is impractical because inverting such large matrices takes much more time than computing the exact equations. Further, if the 7 points are very far apart in parameter space, the resulting estimates may not be very accurate at all.

Path length extrapolation: Full first order extrapolation is only possible using stored data. In order for extrapolation from dynamic data to be fast enough to be preferable to just computing the equations, very low order extrapolation would have to be used. The problem with first order extrapolation is that there are just too many parameters to the equations. The logical limit of the extrapolation approach is to view the equations for the forces as functions of a single parameter (distance along the trajectory) rather than 18. If Q, $\dot{Q}$, and $\ddot{Q}$ are all slowly varying, then the F_i will be slowly varying functions of path length. F can then be approximated by extrapolating from prior points on the path. It is important to note that in general, $\ddot{Q}$ tends to change abruptly, and that extrapolation is not accurate beyond such a change. This problem is reduced when working with the kind of trajectories discussed in the last section where there are relatively large periods of constant acceleration. [Since the writing of this paper, path length extrapolation of computed force values was used in the arm control system described in [Waters 1974].]

Precomputaton of the Equations

One way to deal with the problem of computing the dynamic equations in real time is to compute them off line and create a data base of function values. There are two main ways to use precomputation: create a tabular representation of the function over its entire domain, or precompute just those values which will be used. In either case interpolation has to be used in order to apply the precomputed data to the actual situations which arise.

Complete coverage: Trying to obtain complete coverage is impractical. The key difficulty is that a huge number of values has to be stored because the equations have so many parameters. As discussed above, in order to be able to efficiently interpolate between the stored data points, the stored points must be orthogonal in that each point is surrounded by other points which differ from it in only one parameter. With a 6 joint arm, if the stored data covered just 5 different values for each parameter there would have to be $5^{18}=4\times10^{12}$ sets of 6 values. It is not possible to precompute that much data, nor to store it. Also note that if the information was precomputed, it becomes obsolete as soon as the hand pickes something up because this changes the inertial properties of the arm.

It is possible that it might be more efficient to store the 6x18=108 partial derivatives with each point explicitly, and to store fewer points. However, it is unlikely that this approach would lead to the required reduction of 9-10 orders of magnitude in the number of data points stored. (A reduction of 2 orders of magnitude is needed just in order to break even.)

The storage problem can be partly ameliorated by storing partial precomputation information which factors out some of the parameters. The dynamic equations can be rearranged (see Paul [1971]) so that:

$$F = D(Q,\dot{Q})\ddot{Q} + E(Q,\dot{Q})$$

This formulation makes it inherently easier to adjust for changes in acceleration. Complete coverage with this information at 5 values per parameter requires $5^{12}=2x10^8$ sets of 42 values. (If derivatives are stored, 42x12=504 more values are required at each point.) This is still much to much data to be practical.

Covering only a single trajectory: One way to solve the space problem is to store only the information which is actually going to be needed. Paul [1971], who apparently evaluated the equations as written rather than by using recurrence relations, evaluated them off line before a motion was started. He planned a detailed trajectory consisting of the exact position and velocity of the arm in each time interval. Then he precomputed the matrix D and the vector E at each time interval. He did not use interpolation, but rather just used the D,E pair closest to the current phase space position.

With this method, small errors in trajectory can be corrected by varying the accelerations. However, if a large error ever develops then the controller is unable to get the arm back on the correct trajectory because its precomputed D's and E's are accurate only in the vicinity of the points for which they were calculated. Note that this method has the additional defect that overall there is no time saved. The arm can move very fast, but then it must sit and wait while the D's and E's for its next trajectory are computed off line.

Storing Typical Trajectories: A compromise between the two methods above is to store precomputed information about small restricted areas of the domain. This is predicated on the idea that most of the motions of the arm are stereotyped, and follow a reasonably small number of typical trajectories. Further, any non-standard motions the arm makes can be made slowly. Introspection suggests that this could be true of human motions. One of the prime tenents of sports is that you should always get into a standard position before performing a motion in order to increase your precision.

One way to store information about typical trajectories is to describe the regions of the domain they pass through. If interpolation is to be used rather than just using the nearest available data point, derivatives would have to be stored with the points. It is interesting to note that if information about typical trajectories is stored this way, then it is clear how the arm controller can learn about a new trajectory by doing it, and remembering the properties of that region of phase space. [Since the writing of this paper, this basic approach to arm control was taken by Raibert [1977].]

Another way of storing typical trajectories would be to use a procedural form. There would be a procedure for each motion, and the procedure would be parameterized by values appropriate to the motion. For instance, throwing a ball would be parameterized by the velocity the ball was supposed to have, and the point at which it was to be released. However, it is not obvious how these procedures can be derived.

BIBLIOGRAPHY

Hollerbach, J.M., [1979], "A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity", MIT/AIM-533, June, 1979.

Horn, B.K.P; and Inoue, H., [1974], "Kinematics of the MIT-AI-VICARM Manipulator", MIT/AI/WP-69, May 1974.

Kahn, Michael E., [1969], "The Near-Minimal-Time Control of Open-loop Articulated Kinematic Chains," PhD Thesis Stanford University, December, 1969, (also Stanford AIM 106).

Luh, J.; Walker, M.; Paul, R., [1979], "Newton-Euler Formulation of Manipulator Dynamics for Computer Control", to be presented at the 2nd IFAC/IFIP Symposium on Information Control Problems in Manufacturing Technology, Stuttgart, Germany, October, 1979.

Marion, Jerry B., [1965], "Classical Dynamics of Particles and Systems," Academic Press, New York, 1965.

Paul, Richard, [1971], "Trajectory control of a Computer Arm," Stanford University, February, 1971.

Pieper, D.L., [1968], "Kinematics of Manipulators Under Computer Control," PhD Thesis Stanford University, October, 1968 (also Stanford AIM 72).

Raibert, M.H., [1977], "Motor Control and Learning by the State Space Model", MIT/AI/TR-439, September 1977.

Scheinman, V.D., [1969], "Design of a Computer Controlled Manipulator," Stanford AIM 92, June, 1969.

Uicker, J.J. Jr., [1965], "On the Dynamic Analysis of Spatial Linkages Using 4 x 4 Matrices," PhD Thesis Northwestern University, Evanston Illinois, August, 1965.

Waters, R.C., [1974], "A Mechanical Arm Control System", MIT/AIM-301, January, 1974.

Whitney, D.E., [1972], "The Mathematics of Coordinated Control of Prosthetic Arms and Manipulators", ASME Journal of Dynamic Systems Measurement and Control, December, 1972, pp. 303-309.