On the Simulation of Space Based Manipulators with Contact [1]

Michael W. Walker and Joseph Dionise

Robotics Research Laboratory
Electrical Engineering and Computer Science Department
The University of Michigan
Ann Arbor, Michigan 48109

Abstract

This paper presents an efficient method of simulating the motion of space based manipulators. Since the manipulators will come into contact with different objects in their environment while carrying out different task, an important part of the simulation is the modeling of those contacts. An inverse dynamics controller is used to control a two armed manipulator whose task is to grasp an object floating in space. Simulation results are presented and an evaluation is made the performance of the controller.

1 Introduction

Robotic manipulators carried by future spacecraft are expected to perform many important tasks in space. This paper presents a methodology for the simulation and control of these robots. The main idea is the need to include not only the dynamics of the robot in the simulation, but also the dynamics of the objects in which the robot comes into contact. The same is true in the design of the controller. If the robot is to be used in grappling a satellite, then the controller and the simulation must include a model of the satellite dynamics.

The model we use is called the dynamic world model as it includes the dynamics of the objects in the robots environment and the dynamics of the robot itself. The next section presents the formulation of this world model. Included in the model is the dynamics of contact. A recently reported distance algorithm is used to detect contact [4].

The next section deals with the control of the robot. Feedback functions can be utilized by the controller either at the joint level or at the Cartesian level. An example is provided for an implementation of inverse dynamics control.

The next section presents an example simulation. In this simulation, two PUMA 560 manipulators are mounted on a single base. The task is to grapple an object floating in space. The inverse dynamics control algorithm which has been developed for terrestrial-based robots is used for control. Compliance is obtained by using a relatively small position error gains in the controller.

The final section concludes the paper with a discussion of the merits and limitations of the method.

2 Dynamic World Model

This section presents the world model used by the simulation. First, the data structure which is used to store the model is presented. Next, the kinematic modeling of the system is presented. The systems dynamic equations of motion are then presented followed by the dynamics of contact.

[1] This work was supported in part by a grant from the NASA sponsored Center for Autonomous and Man-Controlled Robotic and Sensing Systems, CAMRSS, at ERIM, Ann Arbor, MI

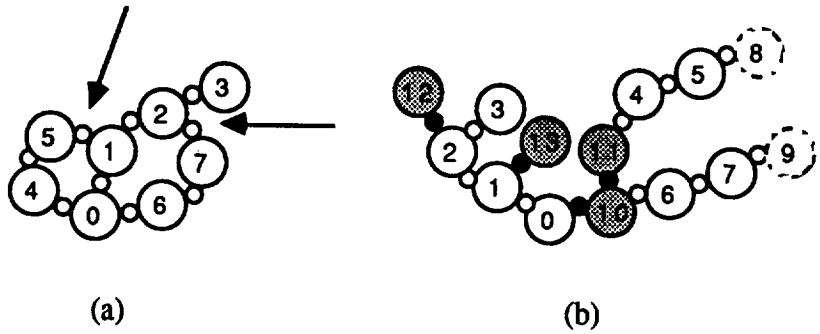

Figure 1: An Example System

2.1 The Data Structure

In the implementation of the controller, several quantities associated with each link are needed, such as: the link velocity, acceleration, and inertial forces and moments. This section presents the data structure used to store this information. The form of this data structure is important since the design of the control algorithm is directly linked to this data structure. An example system is illustrated in Figure 1(a) as a basis for discussion.

In general, the number of rigid links in the system is $m+1$. For the example in Figure 1(a), $m+1 = 8$. The inertial link is a imaginary link fixed with respect to an inertial reference frame and numbered 0. All other links are numbered in an arbitrary order from 1 to m. It is apparent that a graph data structure could be used to store the information. Each record or item in the data structure would be associated with a particular link and all of the information needed about that link would be stored in the associated data record. The problem with using this type of data structure is that it does not lead to a particularly simple or efficient algorithm for the controller. A much better data structure is a binary tree data structure. The remainder of this section develops the method of obtaining this type of data structure.

We begin by selecting a set of joints such that if the associated links were disconnected at these joints there would be a unique sequence of links connecting any given link to the inertial link. An example of two joints which accomplish this are indicated in Figure 1(a) by the large arrows. The resulting structure is a tree structure and we can now establish relationships between different links in terms of their descendants and predecessors. For example, the descendants of link 1 are links 2 and 3. The predecessors of link 7 are links 0 and 6. The immediate descendant of link 1 is link 2 and the immediate predecessor of link 2 is link 1. We also note that some links may have more than one immediate descendant. For example, link 0 has three: 1, 6, and 4. The fact that a link could have an arbitrary number of immediate descendants causes practical problems in the implementation of the controller. To get around this problem we introduce the concept of connector links.

An example of a link with four joints is illustrated in Figure 2. The joint connecting the immediate predecessor to the link is assigned the same number as the link. One of the other three joints is used to locate the link coordinates. This determines the D-H kinematic parameters a_i, α_i, d_i, and θ_i as illustrated in the figure. To locate the remaining two joints with respect to link i coordinates we insert fictitious links, j and k, called connector links. First is link j which is located relative to link i coordinates. The D-H kinematic parameters [1], a_j, α_j, d_j, and θ_j are used to locate this link j coordinates with respect to link i coordinates. Next is link k which is located relative to link j coordinates. Again the D-H kinematic parameters, a_k, α_k, d_k, and θ_k are used to locate this link k coordinates with respect to link j coordinates. Note that all of the kinematic parameters, a, α d, and θ, associated with connector links are constant and that for both the connector and successor links the position and orientation of the descendant with respect to the current link

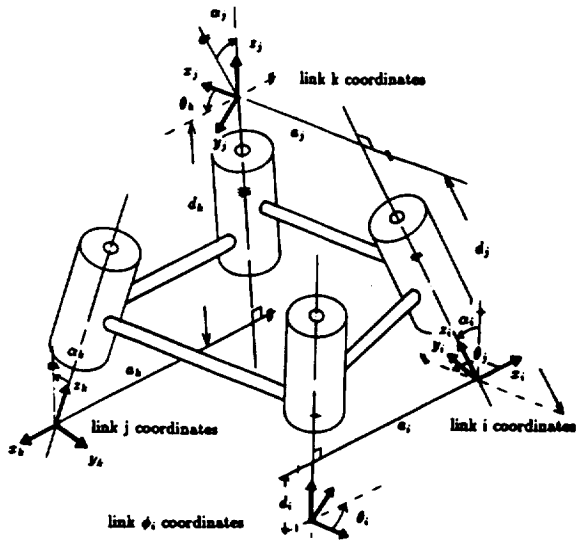

Figure 2: An Example Link, Numbered i

is described using homogeneous transforms parameterized by the D-H kinematic parameters a, α, d and θ. The real links are called successor links to distinguish them from the imaginary connector links.

Therefore, each link can only have to two immediate descendants. One would be a connector link and the other would be a successor link. Only one of each type is allowed. Hence, with the introduction of the connector links we have converted a general tree data structure into a binary tree data structure.

For a given link, the first joint encountered when moving in the direction of the inertial link is numbered the same as the given link. Thus, all of the joints except those where the kinematic loops have been broken have been assigned a number. There are m of these. Assuming there are r independent loops the remaining r joints located at the points where the kinematic loops have been broken are numbered from $m+1$ to $m+r$ making a total of $m+r$ joints. Finally, a fictitious successor link called a terminating link is associated with these joints and is assigned the same number as the associated joint. The purpose of the terminating link is to allocate a item in the data structure to store all of the information concerning the associated joints, for example, their position and the viscous friction coefficients. The D-H kinematic parameters for the terminating links are chosen so that the loop closure equations can be written in terms of homogeneous transforms. This will be described in more detail in the section concerning the constraint equations.

If link i is a successor link then the associated joint is either translational or rotational. In either case the joint position is denoted by q_i. If the joint is translational then $q_i = d_i$. If the joint is rotational then $q_i = \theta_i$.

Finally, connector links are numbered starting at $m+r+1$ on up to the total number of links, both imaginary and real, contained in the system.

The resulting data structure for the example system is illustrated in Figure 1(b). The convention we use is that the items in the data structure associated with connector links are connected to their predecessor with small solid circles and items associated with successor links are connected to their predecessor with small white circles.

2.2 System Kinematics

From Figure 1(b) one can see that the position of each link can be determined with respect to the inertial coordinates by starting at the inertial link and successively computing each link position while moving out from the inertial link. Let ϕ_i denote the number of the immediate predecessor of link i. If the homogeneous transform of link ϕ_i coordinates with respect to the inertial link 0 coordinates, $T_0^{\phi_i}$, is known then the homogeneous transform of link i coordinates with respect to the inertial link is T_0^i and can be computing using the following equation.

$$T_0^i = T_0^{\phi_i} T_{\phi_i}^i$$

where $T^i_{\phi_i}$ is a function of the a_i, d_i, α_i, θ_i. Note that the same equation applies if link i is a connector link or a successor link.

This paper uses the spatial notation by Featherstone, [3,5]. With this notation transformation matrices are 6 x 6 matrices. The spatial transformation matrix from link i to link ϕ_i is:

$$X^i_{\phi_i} = \begin{bmatrix} A^i_{\phi_i} & O \\ K(p^i_{\phi_i})A^i_{\phi_i} & A^i_{\phi_i} \end{bmatrix}$$

where O is the 3 x 3 null matrix, $A^i_{\phi_i}$ and $p^i_{\phi_i}$ are the upper left 3 x 3 submatrix and upper right 3 x 1 submatrix of the homogeneous transformation matrix $T^i_{\phi_i}$, respectively. The 3 x 3 matrix $K()$ is a skew symmetric matrix such that $K(a)b = a \times b$ for any 3 x 1 vectors a and b.

As with homogeneous transforms, the spatial transformation from link i coordinates to link 0 coordinates can be computed given that of its immediate predecessor by multiplying the transforms together.

$$X^i_0 = X^{\phi_i}_0 X^i_{\phi_i} \tag{1}$$

The spatial velocity and acceleration of link i can be determined given those of its immediate predecessor, ϕ_i.

$$v_i = \begin{cases} v_{\phi_i} + s_i\dot{q}_i & \text{if link i is a successor} \\ v_{\phi_i} & \text{if link i is a connector} \end{cases} \tag{2}$$

$$\dot{v}_i = \begin{cases} \dot{v}_{\phi_i} + s_i\ddot{q}_i + v_i \times s_i\dot{q}_i & \text{if link i is a successor} \\ \dot{v}_{\phi_i} & \text{if link i is a connector} \end{cases} \tag{3}$$

where s_i is the third column of $X^{\phi_i}_0$ if joint i is rotational or the sixth column of $X^{\phi_i}_0$ if joint i is translational.

Thus, one starts at the root of the binary tree data structure and works out along the branches using the above equations to successively compute the spatial transformation matrix, velocity and the acceleration of each link.

2.3 Kinematic Constraint Equations

In general, the system is graph structured with n degrees of freedom. For this reason, we partition the set of joints into two mutually exclusive sets. There are n joints in the first set. They are called primary joints since their positions are independent of any other joint positions in the system. These joint positions are combined into a single n x 1 vector called, Q. The joints in the second set are called secondary joints, as their positions are strictly functions of the primary joint positions. Thus, the positions of all the joints in the system are functions of the primary joint positions. We can write this fact in the form of the following constraint equation.

$$q = U(Q) \tag{4}$$

Note that for each Q_i there is a q_j such that $Q_i \equiv q_j$. Given $U(Q)$, $\dot{Q}$, and $\ddot{Q}$, the velocity and acceleration of all the joints can be determined.

$$\dot{q} = E(Q)\dot{Q} \tag{5}$$

$$\ddot{q} = E(Q)\ddot{Q} + \dot{E}(Q)\dot{Q} \tag{6}$$

where

$$E(Q) = \frac{\partial U(Q)}{\partial Q} \tag{7}$$

For a given system these equations are usually fairly simple. Often the matrix $E(Q)$ is constant. For example, it is simply the identity matrix for a tree structured system. However, for some systems these equations can become complex. For these systems we have found that the ϵ-algebra is very convenient for evaluating the time derivatives of the joint positions [9]. Using this algebra one only has to program to solution to Equation 4, change the order of the algebra and automatically obtain the solution to equations 5 and 6.

The equations of constraint are obtained from the loop closure equations. For the system shown in Figure 1(a) there are two independent loops. The two loop closure equations are:

$$T_0^1(q_1)T_1^{13} = T_0^{10}T_{10}^{11}T_{11}^4(q_4)T_4^5(q_5)T_5^8(q_8) \tag{8}$$

and

$$T_0^1(q_1)T_1^2(q_2)T_2^{12} = T_0^{10}T_{10}^6(q_6)T_6^7(q_7)T_7^9(q_9) \tag{9}$$

These two matrix equations consist of a total of 32 scalar equations of which only six are independent. Thus, we can determine q_1, q_2, q_4, q_6, q_8, and q_9 as a function of the joint positions q_5 and q_7. Using the above notation, $q_5 = Q_1$, $q_7 = Q_2$ and $q_3 = Q_3$. These three equations along with the six obtained from equations 8 and 9 give us the function $\boldsymbol{U}(\boldsymbol{Q})$ defined above.

2.4 System Dynamics

The equations of motion of a system can be written in the following form:

$$\boldsymbol{\tau} = \boldsymbol{H}(\boldsymbol{Q})\ddot{\boldsymbol{Q}} + \boldsymbol{C}(\boldsymbol{Q}, \dot{\boldsymbol{Q}})\dot{\boldsymbol{Q}} + \boldsymbol{G}_e(\boldsymbol{Q})$$

where

$\boldsymbol{\tau}$	=	$n \times 1$ vector of active forces
$\boldsymbol{H}(\boldsymbol{Q})$	=	$n \times n$ pseudo moment of inertia matrix
$\boldsymbol{C}(\boldsymbol{Q}, \dot{\boldsymbol{Q}})\dot{\boldsymbol{Q}}$	=	$n \times 1$ vector of friction, centrifugal and Coriolis terms
$\boldsymbol{G}_e(\boldsymbol{Q})$	=	$n \times 1$ vector of external force components

In the simulation program the state variables are the position and velocity of the primary variables, $\boldsymbol{Q}$. To simulate the system one determines the acceleration of these variables from the above equation:

$$\ddot{\boldsymbol{Q}} = \boldsymbol{H}(\boldsymbol{Q})^{-1}(\boldsymbol{\tau} - \boldsymbol{C}(\boldsymbol{Q}, \dot{\boldsymbol{Q}})\dot{\boldsymbol{Q}} - \boldsymbol{G}_e(\boldsymbol{Q}))$$

We use the first method presented in [10] to compute the $\boldsymbol{H}(\boldsymbol{Q})$ matrix. Although this method was originally was presented for serial link manipulators, the approach is still applicable if one uses an inverse dynamics algorithm for systems with closed kinematic loops [6,7,8].

The inverse dynamics algorithm we use is a three step process [8].

1. Given the desired positions, velocities, and accelerations of the independent variables, $\boldsymbol{Q}$, calculate the corresponding positions, velocities, and accelerations of the joints, $\boldsymbol{q}$, using equations, 4 through 6.

2. For each link, determine ρ_j, the sum of the friction force and the projection of the sum of the inertial forces on the axis of motion of joint j.

3. Determine the active forces, $\boldsymbol{\tau}$ using the constraint equations and the ρ_j computed in step 2.

$$\boldsymbol{\tau} = \boldsymbol{E}(\boldsymbol{Q})^T\boldsymbol{\rho}$$

where $\boldsymbol{\tau}$ is an $n \times 1$ vector of the τ_j and $\boldsymbol{\rho}$ is an $(m + r) \times 1$ vector of the ρ_j.

Step 2 is exactly the same procedure which would be used if the system was a tree structure system. In this case the ρ_j would be identical to the active forces. The only difference is in the first step, wherein joint positions, velocities, and accelerations are determined which are consistent with the constraint equations, and the last step, wherein the constraint equations are again used in determining the active forces.

Step 2 can be implemented in three steps. First, the position, velocity and acceleration of each link is computed using equations 1 through 3. This is done by starting at the inertial link and working out along the branches of the tree. Next, for each link j we determine $\boldsymbol{f}_j$, the sum of the external forces and inertial forces of link j and all of its descendants. This is done by starting at the tips of the branches of the tree and working back toward the inertial link of the tree using the following equations. Let $\boldsymbol{F}_j$ denote the inertial force of link j. Then:

$$\boldsymbol{F}_j = \dot{\boldsymbol{M}}_j = \boldsymbol{I}_j\dot{\boldsymbol{v}}_j + \boldsymbol{v}_j \times \boldsymbol{I}_j\boldsymbol{v}_j \tag{10}$$

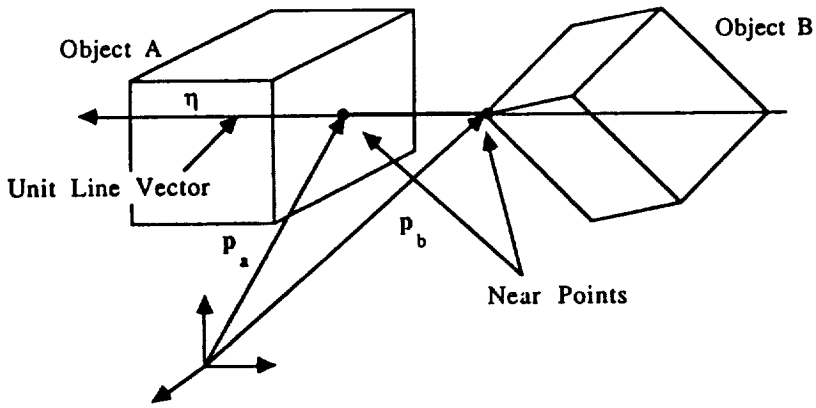

Figure 3: Illustration of distance between two convex polytopes

where $M_j = I_j v_j$ is the spatial momentum vector for the $j-th$ link and I_j is the spatial moment of inertia matrix referred to inertial coordinates:

$$I_j = X_0^j \mathcal{I}_j X_0^{j\prime}$$

In this equation the matrix $\mathcal{I}_j$ is the spatial moment of inertia matrix referred in link j coordinates:

$$\mathcal{I}_j = \begin{bmatrix} -m_j K(\underline{p}_j) & m_j I \\ \mathcal{I}_j & m_j K(\underline{p}_j) \end{bmatrix}$$

where m_j is the mass of link j, $\underline{p}_j$ is the location of the center of mass, and $\mathcal{I}_j$ is the moment of inertia matrix. Both $\underline{p}_j$ and $\mathcal{I}_j$ are defined with respect to link j coordinates. The 3×3 skew symmetric matrix $K(\underline{p}_j)$ is a function of the vector $\underline{p}_j$ such that for any vector a, $K(\underline{p}_j)a = \underline{p}_j \times a$.

For connector and terminating links the F_j are zero since they have no mass. Let j be the index of the current link. and let k and l be the index of the its successor and connector links. The vector f_j is used to denote the sum of the inertial and external forces for link j and all of it's descendants.

$$f_j = F_j + F_j^e + f_k + f_l \tag{11}$$

The last part of step 2 is to determine ρ_j, the force due to friction, η_j, plus the projection of f_j onto the axis of motion of joint j.

$$\rho_j = s_j' f_j + \eta_j \tag{12}$$

where $\eta_j = d_j \dot{q}_j$ is the component of joint force due to viscous friction.

2.5 Contact Dynamics

To simulate contact we need a geometric model of the links in the system and a method of computing the distance between these objects. We use unions of convex polytopes to model the geometry of each link. An example of two objects is illustrated in figure 3. Polytopes were used as a model since the shape of links can be closely approximated by using a suitable number of vertices in the polytopes and a very efficient distance algorithm exist for computing the distance between convex polytopes, [4]. Given the position of the two polytopes the algorithm returns the distance, d, between them as well as the location of the point in each polytope which is nearest the other polytope, p_a and p_b. In addition, the unit line vector, η, is returned which passes through the near points of the two objects.

The distance between objects is calculated after each integration step in the simulation. The integration algorithm is a variable step size algorithm. When collision has not occurred, the integration step size is limited to ensure that each joint position changes by no more than a specified amount. As objects near each other, the step size is also limited so the objects will just touch at the next integration step. We say that two objects are just touching when the distance between them is equal to a small constant, ϵ. An estimate of the time when the objects will collide is given by the following equation.

$$collision\ time = \frac{\epsilon - d}{\dot{d}}$$

If the collision time is greater than zero, the integration step size is selected to be less than or equal to this value. If the collision time is negative, then the objects are moving apart from each other and this collision time is ignored.

The value of $\dot{d}$ is easily computed from the unit line vector, η, returned from the distance algorithm and the spatial velocities of the two objects, v_a and v_b, which are obtained from the simulation algorithm.

$$\dot{d} = \eta'(v_a - v_b)$$

The dynamics of contact is modeled like a spring. When the distance between the objects is less than ϵ a force is applied to object A in the direction of η with magnitude, f_c, given by:

$$f_c = K_p \frac{\epsilon - d}{d} - K_d \dot{d}$$

where K_p and K_d are positive constants. Thus, the magnitude approaches infinity as the actual distance, d, approaches zero. The objective of this is to ensure that the objects never intersect. The derivative term is added to provided damping. The spatial force, f_e exerted on object A is given by:

$$f_e = f_c \eta$$

Thus, we are modeling hard contacts with no sliding friction. The spatial force exerted on object B is the negative of this value. These forces are included into the dynamic simulation as external forces exerted on the links as described in the above section on dynamics.

3 Control

Feedback signals used be the controllers come in two forms: spatial feedback vectors, $\hat{F}_j$, which are associated with the links of the manipulator and scalar feedback functions, $\hat{\eta}_j$, which are associated with the joints of the manipulator. These can be any functions of the desired trajectory and the actual trajectory of the manipulator. The method used to calculate the required actuator forces is as follows.

1. The feedback control law is implemented through the use of a set of spatial feedback vectors, $\hat{F}_j$, which are defined for each link in the system, and scalar feedback functions, $\hat{\eta}_j$, which are defined for each joint of the manipulator.

2. The τ_j are computed using the $\hat{F}_j$ and $\hat{\eta}_j$ as in the following recursive equations:

$$\begin{aligned} \hat{f}_j &= \hat{F}_j + \hat{f}_k + \hat{f}_l \\ \hat{\rho}_j &= s_j' \hat{f}_j + \hat{\eta}_j \\ \tau &= E(Q)^T \hat{\rho} \end{aligned}$$

   where k and l are the immediate descendants of link j, $\hat{\rho}$ is an $(m + r) \times 1$ vector of the $\hat{\rho}_j$.

These types of calculations are common to a variety of controller algorithms and are included only for the users convenience. The choice of the $\hat{F}_j$ and $\hat{\eta}_j$ dictates the character of the controller being implemented. The following example is for an inverse dynamics controller.

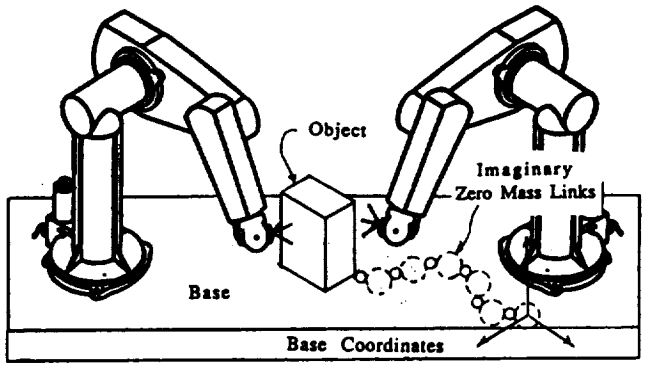

Figure 4: Example System Used in Simulation

In the inverse dynamics controller, both the spatial feedback vectors and feedback functions are used. The following variables are used to compute them.

$$\begin{aligned} Q_e &= Q_d - Q & \ddot{\hat{q}} &= E(Q)\ddot{Q} + \dot{E}(Q)\dot{Q} \\ \ddot{\hat{Q}} &= \ddot{Q}_d + K_d\dot{Q}_e + K_p\dot{Q}_e & \hat{v}_0 &= 0 \\ \dot{\hat{q}} &= E(Q)\dot{Q} & \dot{\hat{v}}_0 &= 0 \end{aligned}$$

where K_d and K_p are positive constants.

For $i > 0$, the spatial acceleration vectors are computed using recursive equations:

$$\begin{aligned} v_i &= \begin{cases} v_{\phi_i} + s_i\dot{q}_i & \text{if link i is a successor} \\ v_{\phi_i} & \text{if link i is a connector} \end{cases} \\ \dot{\hat{v}}_i &= \begin{cases} \dot{\hat{v}}_{\phi_i} + v_i \times s_i\dot{q}_i + s_i\ddot{\hat{q}}_i & \text{if link i is a successor} \\ \dot{\hat{v}}_{\phi_i} & \text{if link i is a connector} \end{cases} \end{aligned}$$

The spatial feedback vectors are functions of these vectors. For $i > 0$:

$$\hat{F}_i = \begin{cases} \hat{I}_i\dot{\hat{v}}_i + v_i \times \hat{I}_i v_i & \text{if link i is a successor} \\ 0 & \text{if link i is a connector or terminator} \end{cases}$$

The scalar feedback functions are:

$$\hat{\eta}_i = \begin{cases} \hat{d}_i\dot{q}_i & \text{if link i is a successor} \\ 0 & \text{if link i is a connector} \end{cases}$$

The hat over the $\hat{d}_i$ and the $\hat{I}_i$ indicates they are only estimates of the corresponding components of the actual system. Thus, $\hat{I}_i$ is the spatial moment of inertia matrix of link i using the estimated values of the link inertial parameters.

This control law is a recursive implementation of the following function:

$$\tau = H(Q)\ddot{Q} + C(Q, \dot{Q})\dot{Q}$$

4 Simulation Results

A simulation was performed of a two armed robot consisting of two PUMA 560 manipulators mounted on a single base. The objective of the simulation is to test the ability of the manipulators to grasp a floating object using the above inverse dynamics control algorithm. Each manipulator is equipped with an end-effector having three rigid fingers. There are three distinct phases of this task: approach, contact, and coordination. The approach phase is when both manipulators are approaching the object, but nether has contacted the object. The contact phase is the time after the first contact is made by either manipulator and before the coordination phase. The coordination phase is after both manipulators have made three point contacts with the object. A relatively low position feedback gain at each joint of the manipulators to provide the positional compliance required during contact with the object.

The system is illustrated in figure 4. To model the floating object, five massless links were inserted between the base coordinate frame and the object coordinates. These are denoted by the five large dotted line circles in the figure.

As expected the controller performed fairly well during the approach phase. Because of the small gains the settling time was about one second. This could be improved by increasing the gains with the result of reducing the compliance during the contact and coordination phases. The controller did not perform as well during the contact and coordination phase. Significant position errors and forces were observed due to the interaction of the object with the manipulator.

5 Conclusion

This paper presented an efficient method of simulating the motion of space based robots. The simulation includes not only the dynamics of the manipulators but also the dynamics of the objects in which the manipulators come into contact. Thus, the effect the robots have on the world in which they are a part is an important aspect of the simulation.

An example simulation was provided of a two armed robot grappling a satellite. It was found that the inverse dynamics controller worked reasonably well during the approach phase of this task, but it did not work very good during the contact and coordination phases. The controller was not successful in obtaining a stable grappling of the object. The conclusion is that a different control law is required for the three distinct phases of this task: approach, contact, and coordination. A position control should be used during the approach phase. A compliant motion control should be used during the contact phase and a dual-arm coordinated motion control should be used to manipulate the satellite.

In our simulation and the real robot controller we have the capability of switching between any one of a number a control laws from one sample period to the next. The problem we face is in deciding when a new control law should used. This is a complex decision in which several factors must be considered. The decisions for the switching times and the selection of the control laws should be based upon the task definition and available sensory measurements, such as tactile and force sensors. Except in a very simple manner, we currently have no way of either simulating this decision process or implementing it in the real robot controller. We believe the complexity dictates the use of artificial intelligence techniques such as expert systems to make these decisions. The lack of this decision making component is the greatest limitation of our system.

Finally, the last phase of controller development is the installation and testing with the real robot. This process is facilitated in the our laboratory by programming both the simulation and the real robot controller in the same language, Ada. Also, the definition of the spatial and scalar feedback functions are incorporated into the simulation as well as the real controller with identical pieces of Ada software. Thus, once the simulation results are acceptable, the controller code from the simulation is transfered to the robot's control computer, where it is recompiled and linked into the robot controller software. Future goals will be to integrate an expert system into both the simulation and the real robot motion controller to handle the control law switching decisions.

References

[1] J. Denavit and R. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *ASME Journal of Applied Mechanics*, pages 215–221, June 1955.

[2] R. Featherstone. The calculation of robot dynamics using articulated-body inertias. *The Int. J. of Robotics Res.*, 2(1):13–30, Spring 1983.

[3] R. Featherstone. *Robot Dynamics Algorithms.* Kluwer Academic Publishers, Norwel, MA, 1987.

[4] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, April 1988.

[5] R. Lathrop. Constrained (closed-loop) robot simulation by local constraint propagation. In *I.E.E.E. Int'l Conference on Robotics and Automation*, San Francisco, CA, 1986.

[6] J. Y. S. Luh and Y. F. Zheng. Computation of input generalized forces for robots with closed kinematic chain mechanisms. *I.E.E. Trans. on Systems, Man and Cybernetics*, RA-1(2):95–103, June 1985.

[7] Y. Nakamura and M. Ghodoussi. A computational scheme of closed link robot dynamics derived by d'alembert principle. In *IEEE Int'l Conf. on Robotics and Automation*, Raleigh, North Carolina, April 1988.

[8] M. W. Walker. An efficient algorithm for the adaptive control of a manipulator. In *IEEE Int'l Conf. on Robotics and Automation*, Raleigh, North Carolina, April 1988.

[9] M. W. Walker. Manipulator kinematics and the epsilon algebra. *I.E.E.E. Journal of Robotics and Automation*, 4(2):186–192, April 1988.

[10] M. W. Walker and D. E. Orin. Efficient dynamic computer simulation of robotic mechanisms. *AME Journal of Dynamic Systems, Measurement and Control*, 104:205–211, Sept. 1982.