

Parallel Robot Control Using Speculative Computation

Deirdre L. Hamilton, Ian D. Walker, and John K. Bennett
Department of Electrical and Computer Engineering
Rice University, Houston, TX 77005-1892

Abstract

Over time, numerous efforts have been made to overcome the limitations of the computationally intensive dynamics calculations. Due to the coupling in the dynamics equations, coarse-grain parallelism of robot control algorithms is particularly difficult. We have developed a new algorithm based on the Newton-Euler dynamics formulation that overcomes the serial nature of these equations, allowing a high level of parallelism. This new algorithm provides a faster sampling rate by executing in parallel, then takes advantage of that faster sampling rate by assuming that the value difference between sample iterations for the dynamics variables will be small. The result is a highly parallel algorithm that is capable of providing quite accurate results. Our controller uses data from a previous control step in current calculations to allow many more tasks to be executed in parallel, thus providing higher control update rates. The use of ‘old’ data is an effective solution to the speedup problem, but presents some special difficulties. A stability issue when using ‘old’ data that is encountered in previous approaches is discussed here, along with a partial solution to the problem.

1 Introduction

As robots are considered for increasingly difficult and intricate tasks, control requirements increase in complexity to meet these demands. However, these tasks cannot be realized without controller architectures that are capable of executing their control algorithms in a timely manner. Therefore, development of faster controllers is important to robot advancement. In response to more stringent precision and speed goals, kinematics, dynamics, and control algorithms have been studied extensively in search of ways to facilitate these tasks. A variety of techniques have been proposed for decreasing the controller computation time [3, 4].

In uniprocessor robot control systems, which are most common today, the opportunity for speed improvement is limited by the total number of mathematical operations required. With current advances in chip technology and software development, uniprocessor architectures are far more powerful than as recently as the early 1990’s. Thus, for some applications, a uniprocessor controller may provide sufficient computing power. However, synthesis of efficient general parallel techniques for robot control architectures can permit state-of-the-art

serial architectures to be extended to allow other important tasks, such as fault detection, to be executed in real time. Ideally, the most efficient uniprocessor algorithms can be parallelized and executed on multiple processors, to achieve linear speedup. Practically, there are communication and synchronization requirements in the parallel algorithms that do not exist for serial algorithms. However, speed improvements are still quite possible, in addition to extra features such as fault tolerance.

A variety of multiprocessor architectures to support parallel robot control algorithms have been studied [5, 11, 12, 15, 17, 23]. However, due to data dependencies, the robot dynamics equations are not easily parallelized. More recent research efforts in robot kinematics and dynamics specifically consider development of efficient parallel versions of these algorithms [5, 6, 8, 9, 10, 16, 18, 20, 21, 25, 26]. Smaller tasks, such as computation of the inertia matrix, have also been parallelized [7]. This work addresses the dynamics of a robotic system.

There are basically two standard methods for analyzing the dynamics of a robot: the Lagrangian formulation and the Newton-Euler formulation. In the classic Lagrangian (LE) form, the robot is analyzed in closed form. The dynamic model is expressed as:

$$\underline{\tau} = [M(\underline{\theta})]\ddot{\underline{\theta}} + \underline{N}(\underline{\theta}, \dot{\underline{\theta}}) + \underline{G}(\underline{\theta}) \quad (1)$$

where $\underline{\theta}$ is the $n \times 1$ vector of joint angles for an n -joint robot, $\underline{\tau}$ is the $n \times 1$ joint torque vector, $[M]$ is the $n \times n$ inertia matrix, \underline{N} is the $n \times 1$ Coriolis and centrifugal torque vector, and \underline{G} is the $n \times 1$ gravity torque vector [19]. The LE formulation defines a general relationship between the variables of each joint and the torques required to move them, and the structure of the robot model is maintained: inertial, gravitational, and other effects are distinct. This closed form is useful for understanding the overall coupled state of the robot. However, the LE dynamics algorithm is computationally intensive, giving rise to a low controller sampling rate and decreased control precision when implemented in its direct form.

The recursive Newton-Euler (NE) equations define an iterative joint variable/torque relationship corresponding to (1) at a particular time [19]. (The NE equations are shown in the Appendix.) As indicated by the equations, each link is considered separately in the Newton-Euler form. However, since the links are connected, the equations for each link contain coupling terms that are also in the same equations for neighboring links. This interdependence of link pairs restricts the order of execution of the equations. Computations for link i wait for completion of computations for link $(i - 1)$ (or link $(i + 1)$), as shown in Figure 1 (F_i and B_i represent the forward and backward recursion computations for link i , respectively). Also, calculations within the forward and backward recursions, individually, must be executed in a certain order due to data dependencies. However, despite these restrictions, the NE form is generally considered to be more computationally efficient than the classic LE form [13].

The classic Lagrangian is appreciated for its closed-form, while the recursive Newton-Euler form allows faster calculation of the dynamics equations. The strict ordering of the NE equations is the consequence of the coupled nature of the robot. This state is evident in the LE dynamics equations in the form of duplicate computations. However, the structure of these equations does not incorporate the type of data dependencies that limit parallelism in the NE algorithm. The order of computation of the terms in the component vectors and matrix of (1) is not restricted, and large blocks of calculations may be executed in parallel.

In his survey of methods for performing the inverse dynamics computations, Balafoutis concludes that both formulations can be made computationally efficient, and that recursive algorithms are computationally more efficient than closed-form ones [3]. Thus, a study of methods for parallelizing recursive algorithms is quite useful for the further advancement of robot control.

A recursive Lagrangian-based dynamics form was developed by Hollerbach [13] that reduced the dynamics algorithm from $O(n^4)$ for the classic form to $O(n)$, where n is the number of joints. However, the algorithm still requires more multiplications and additions than the NE form of Luh, Walker, and Paul [13]. Since the tasks that the robot can perform are limited by the sampling rate of the controller, most research on speedup techniques has only considered the recursive Newton-Euler form. Our work also concentrates on a Newton-Euler-based control algorithm.

We address the speed/accuracy issue with a new algorithm that uses data from the previous iteration of the control loop in current computations, allowing the calculations that are typically executed serially to be executed in parallel. This method, which we call speculative computation, is based on the assumption that the resulting sampling rate will be high enough that the difference between the current data and data from the previous iteration is small. Because of this small difference, the output of the controller when using “old” data would be effective. Since using data from the previous iteration allows the computations to be done in parallel (completing more quickly), the expectation is that the sampling frequency will be high enough.

This concept of using “old” data for parallelizing the NE dynamics algorithm was first proposed by Binder and Herzog in 1986 [5]. Further work using the method in [5] was reported in [22] in 1990. However, apart from the interesting results of [24], whose approach has some similarities with our own, it does not appear that there has been much investigation of the use of “old” data for the entire dynamics calculations since this early work. The basic concept was applied to the inertia matrix computations in [2] in the form of relaxed interprocess precedence in 1992.

The new method described in this paper allows greater parallelism than previous methods by using a greater amount of ‘speculative’ data. We also found that the Binder/Herzog algorithm becomes unstable under certain conditions. In Section 8, we discuss the observed instability, and describe why our algorithm is significantly more stable.

2 Speedup for Dynamics

In early robot development, computation time was the limiting factor. Lookup tables were introduced to reduce the time required to compute the dynamics. Later, specialized algorithms were developed to reduce the necessary memory size and compute time. In recent years, processors have reached fantastic speeds. With the ability to compute faster, single-processor architectures are capable of supporting more demanding robot tasks (via more complicated control algorithms) than previously possible. The limitations that prompted the use of lookup tables and specialized algorithms in early robot development no longer exist (for most tasks).

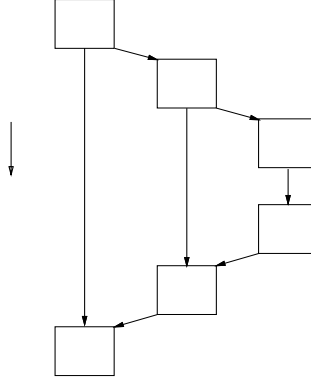


Figure 1: Flow diagram of conventional NE algorithm for a 3-link robot.

While processor technology continues to develop, parallel processing techniques are advancing as well. Thus, with an efficient parallel algorithm, the peak performance of a uniprocessor architecture can always be surpassed. Design of an efficient parallel robot dynamics algorithm would permit improved tracking and precision over the uniprocessor robot controller. One objective of this work was to develop an efficient parallel control algorithm that could also provide processor fault tolerance. Incorporation of processor fault tolerance requires the inclusion of fault detection code in the algorithm. This reduces the speedup made possible with the parallel algorithm. However, careful task scheduling can provide both faster control and processor fault tolerance.

Specialized algorithms can provide the best performance because they are tailored to a particular robot, task, and/or architecture. However, a more general algorithm is less costly because it can be applied to a number of tasks and architectures with few or no modifications. Furthermore, the existence of additional processors makes processor fault tolerance possible. Unfortunately, providing processor fault tolerance reduces controller speed. Therefore, the basic goals for our parallel robot control algorithm are generality and speed performance.

3 Speculative Computation Algorithms for Dynamics

The structure of the NE dynamics equations only allows for some fine-grain parallelism. That is, within the forward and backward recursions, some portions of the calculations of dynamics variables may be executed in parallel. However, coarse-grain parallelism can be accomplished via a method such as the one proposed by Binder and Herzog [5]. Figure 1 shows the order of execution of the forward (F_i) and backward (B_i) recursions in one control loop iteration for a 3-link manipulator. Binder and Herzog state that approximation of various terms allows some of these blocks to be executed in parallel. Their work offers three methods for choosing the approximate values. The first method, called “zero-order prediction,” uses the values computed in the previous control iteration as the approximations. In Figure 2, the three F_i and B_i , respectively, can be executed in parallel because data from the previous control iteration is input to the dependent blocks. This use of “old” data allows one control loop iteration to be completed in two time steps, rather than the six required for the conventional

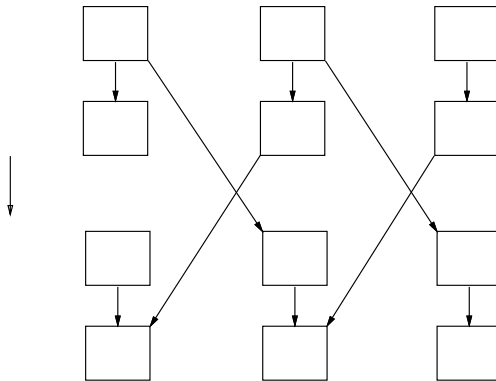


Figure 2: Flow diagram of Binder/Herzog algorithm for a 3-link robot.

method.

“First-order prediction” adds correction terms to the “old” data in order to reduce the errors of “zero-order prediction.” This scheme should provide more accurate results using a reduced sampling rate controller. The third method combines the prediction method with the standard serial method to gain some speed over the conventional approach while maintaining a higher level of accuracy than “zero-” and “first-order prediction” [4]. This method performs the exact calculations for variables selected by the programmer. Binder and Herzog suggest that one might choose those terms that could introduce the largest errors [5]. The remaining terms would be calculated using “zero-” or “first-order prediction.”

The “zero-order prediction” method was simulated for what the authors considered slow (8 sec), fast (2 sec), and very fast (1 sec) robot motion for a Stanford manipulator [5]. These speeds are the total time allowed to complete the trajectory. So, the acceleration and velocity of the manipulator vary with the total time. Their simulation results indicate that for slow and fast movement, the errors produced by “zero-order prediction” are quite small. However, for very fast motion, the errors become noticeable. This is expected because the variations in acceleration and velocity per step become larger as the movement is made faster. Therefore, the difference between the old data and the current data is larger. This example verifies the importance of minimizing the time required to compute the control variables - increasing the number of samples per trajectory (to improve accuracy).

Vuskovic et al. [22] have taken an approach that they consider equivalent to Binder and Herzog’s “zero-order prediction.” Due to the “low inherent parallelism” of recursive formulations, they developed the decoupled recursive Newton-Euler algorithm. In this parallel algorithm, all synchronization between processors is removed. Vuskovic et al. assert that removing the synchronization in a parallel NE algorithm results in “reasonably small” errors. They implemented the exact and decoupled NE algorithms, both with general and customized equations, for the PUMA 560. In their experiments, the errors in output torque due to approximation were measured.

Vuskovic et al. observed that the approximation error in “zero-order prediction” decreases linearly with the controller sampling period. They also experimented with “first-order prediction,” and found that it does not work well. The reasons cited for its poorer performance are the addition of more floating-point operations and increased data communication (i.e.

bus contention). Vuskovic et al. also consider “output prediction,” which is “first-order prediction” of the joint torques, rather than prediction of the recursion variables that are used to calculate the torques [22]. Experimental results showed small approximation errors for this method. As with “first-order prediction”, the results are a function of the number of predictions (calculations) made and bus contention. For “output prediction,” both of these factors are reduced. However, the results were not consistent for different joints and different desired inputs.

Yamakita et al. [24] have also done work similar to that of Binder and Herzog. They proposed a NE algorithm with a cycle time that is independent of the number of links. In this method, each value calculated in a time-step is buffered, to allow the forward and backward portions to be executed in parallel. As with “zero-order prediction,” computations in a control iteration will use data from the previous iteration to overcome serial dependencies. However, since all data is buffered in this approach, the backward recursion computations can be executed in parallel with the forward recursion computations (see Figure 3). As discussed above, errors vary linearly with the controller sampling period. Yamakita et al. reported that the buffering causes delays in the controller sampling period. The delays result in some performance degradation – which corroborate the observations of Vuskovic et al.

The simulation results indicate that buffering alone results in diverging output, especially for joints near the end effector. Therefore, Yamakita et al. use what they call “one step ahead prediction” to alleviate the performance loss due to data buffering. Their method is to predict the values for joint angles as a function of desired values and error terms, which are the differences between desired and actual joint positions from the previous control loop iteration. Their parallel algorithm with “one step ahead prediction” gives more accurate results than buffering alone, with the divergence partially depending on the controller sampling rate [24].

4 A New Speculative Computation Algorithm

Toward the goal of synthesizing a general robot control algorithm, we use the classic recursive Newton-Euler dynamics equations in their complete form. Code optimizations in the dynamics algorithm that are specific to the system we simulated were avoided. For example, multiplications by zero and one that are due to our planar example were not removed. A more customized program might “unroll” loops in the program, remove multiplications by one, and replace multiplications by zero with zeroes. Our program, therefore, does not yield the optimal execution time. However, our simulation provides a conservative sampling rate for comparison to conventional methods.

Controller execution performance was the impetus for developing a robot control algorithm that uses speculative data. The conventional NE algorithm does not offer significant opportunities for coarse-grain parallelism without a method such as “zero-order prediction” or “one step ahead prediction”. Breaking the algorithm into subtasks that can be executed in parallel by using data from a previous time step avoids the delay of waiting for the current data to be computed. Results reported by Binder and Herzog [5], Vuskovic et al. [22], and Yamakita et al. [24] indicate that these techniques are feasible. Since the “zero-order prediction” method does not increase calculation overhead, and is simple to compute, we have chosen to similarly use previously calculated (speculative) values of some data in our

NE control algorithm. We simulate a computed torque PD controller executing on a shared memory multiprocessor [11].

We chose a shared memory architecture due to the data dependencies inherent in robot control. This memory architecture eliminates the need to explicitly move data from one processor to another during the computation of the control equations. Since a robot control architecture would not likely use hundreds, or even tens, of processors, the contention and latency problems often associated with large shared memory multiprocessors do not affect performance in our application. Therefore, a shared memory environment appears particularly appropriate for this application.

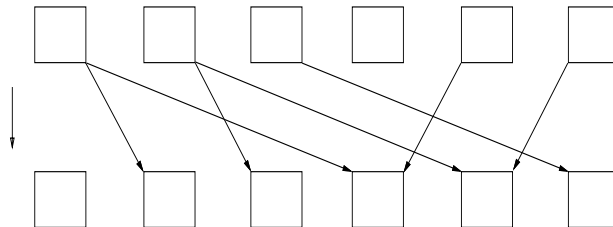


Figure 3: Flow diagram of our algorithm (and Yamakita et al.) for a 3-link robot.

The difference between our algorithm and “zero-order prediction” lies in which terms are replaced by prior values. In “zero-order prediction”, a configuration of one processor per link is assumed, where each processor is responsible for the motion of a corresponding link. When performing calculations for link i , the terms in the equation associated with another link (i.e. having index $(i - 1)$ or $(i + 1)$) are replaced by prior values. Current values are used for the terms corresponding to link i . This formation allows the processors to execute the forward recursion equations for each link in parallel, followed by execution of the backward recursion in parallel (Figure 2). However, the order of operations within each recursion must remain the same as for the conventional NE algorithm, as depicted in Figure 4. In Table 1, the double-boxed variables indicate those variables that are replaced with prior data when using “zero-order prediction.”

Alternatively, all values of recursion variables can be replaced by prior values. These variables are shown in Table 1 as the single- and double-boxed variables. This method allows greater parallelism because some of the synchronization between forward and backward recursions in the conventional and “zero-order prediction” methods is no longer required. Comparison of Figure 3 to Figure 2 shows that completion of the control loop with our method requires half the number of time steps of the Binder/Herzog method. Additionally, the forward iteration can be completed faster using our algorithm since each calculation can be executed in parallel, compared to the serial execution of the Binder/Herzog forward recursion (Figures 4 & 5). This is also true for the backward recursion, but with less impact because there are fewer calculations in the backward recursion. Theoretically, all terms of the forward and backward recursions for each link could be calculated in parallel (i.e. ω_i through τ_i , all in parallel), if enough processors were dedicated to the task. In this case, the length of the control loop is only as long as the longest single term calculation. This may not be the most efficient use of processing resources. However, the flexibility is a key advantage of our method.

Forward Recursion

$$\begin{aligned}
\underline{\omega}_i &= (R_{i-1}^i)^T \boxed{\underline{\omega}_{i-1}} + (R_0^i)^T \underline{z}_{i-1} \dot{\theta}_i \\
\dot{\underline{\omega}}_i &= (R_{i-1}^i)^T \boxed{\dot{\underline{\omega}}_{i-1}} + (R_{i-1}^i)^T \boxed{\underline{\omega}_{i-1}} \times (R_0^i)^T \underline{z}_{i-1} \dot{\theta}_i + (R_0^i)^T \underline{z}_{i-1} \ddot{\theta}_i \\
\underline{\alpha}_i &= (R_{i-1}^i)^T \boxed{\underline{\alpha}_{i-1}} + \boxed{\underline{\omega}_i} \times (R_0^i)^T \underline{z}_{i-1} \dot{\theta}_i + (R_0^i)^T \underline{z}_{i-1} \ddot{\theta}_i \\
\underline{a}_{e,i} &= (R_{i-1}^i)^T \boxed{\underline{a}_{e,i-1}} + \boxed{\dot{\underline{\omega}}_i} \times \underline{r}_{i,i+1} + \boxed{\underline{\omega}_i} \times (\boxed{\underline{\omega}_i} \times \underline{r}_{i,i+1}) \\
\underline{a}_{c,i} &= (R_{i-1}^i)^T \boxed{\underline{a}_{c,i-1}} + \boxed{\dot{\underline{\omega}}_i} \times \underline{r}_{i,ci} + \boxed{\underline{\omega}_i} \times (\boxed{\underline{\omega}_i} \times \underline{r}_{i,ci})
\end{aligned}$$

Backward Recursion

$$\begin{aligned}
\underline{f}_i &= R_i^{i+1} \boxed{\underline{f}_{i+1}} + m_i \underline{a}_{c,i} - m_i \underline{g}_i \\
\underline{\tau}_i &= R_i^{i+1} \boxed{\underline{\tau}_{i+1}} - \boxed{\underline{f}_i} \times r_{i,ci} + (R_i^{i+1} \boxed{\underline{f}_{i+1}}) \times r_{i+1,ci} + I_i \underline{\alpha}_i + \boxed{\underline{\omega}_i} \times (I_i \boxed{\underline{\omega}_i})
\end{aligned}$$

Table 1: NE-based algorithms using “old” data.

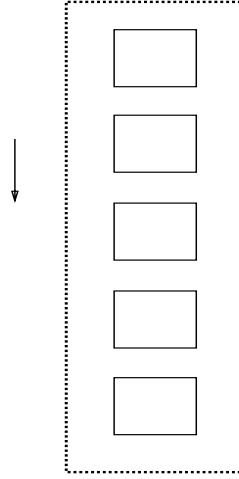


Figure 4: Flow diagram of parallel forward recursion for Binder/Herzog method.

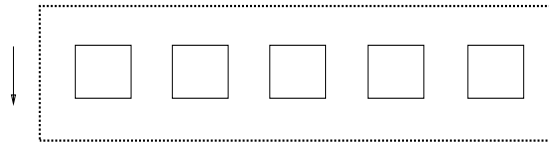


Figure 5: Flow diagram of parallel forward recursion for our method.

Our method is quite similar to the buffering method presented by Yamakita et al. [24]. Like our algorithm, their method uses old data for all dynamics variables in the equations to allow maximum parallelism among the equations. However, due to the NE equations chosen for their algorithm, some speculative terms are older than others in their computations (i.e. some data comes from two control loop iterations prior). This disparity may be the cause of our differing results: their simulations using old data without correction terms sometimes produced diverging outputs; the output of our algorithm, with no correction terms, converges. Due to the need for correction terms in the approach of Yamakita et al., we concentrate on comparison to the Binder/Herzog method.

5 Relative Speedup

Sampling rate is an integral part of accuracy of control. Intuitively, more frequent sampling of a continuous time signal results in less information loss [1]. The Shannon sampling theory states that the sampling frequency must be greater than twice the maximum frequency of the signal to avoid divergence [14]. Thus, a robot control algorithm that is capable of achieving a very high sampling frequency can support a wider range of tasks than a controller with a more limited sampling rate. This section shows, using operation counts, that our algorithm is capable of providing a faster sampling rate than both the serial algorithm and the Binder/Herzog algorithm by adding more parallelism to the algorithm, as discussed in the previous section.

The number of multiplications and additions necessary for calculation of the NE dynamics terms (from Table 1) is used to estimate the relative execution times for the conventional serial NE dynamics algorithm, the Binder/Herzog parallel algorithm, and versions of the new parallel algorithm introduced here. The values shown in Table 2 are for the general case. Terms that appear in the equations of multiple variables are assumed to be calculated once

Term	number of Multiplications				number of Additions			
	$i = 1$	$1 < i < n$	$i = n$	Total	$i = 1$	$1 < i < n$	$i = n$	Total
$\underline{\omega}_i$	12	21	21	$21n - 9$	6	15	15	$15n - 9$
$\underline{\dot{\omega}}_i$	14	32	32	$32n - 18$	10	27	27	$27n - 17$
$\underline{\alpha}_i$	9	18	18	$18n - 9$	8	17	17	$17n - 9$
$\underline{a}_{e,i}$	27	36	36	$36n - 9$	18	27	27	$27n - 9$
$\underline{a}_{c,i}$	27	27	27	$27n$	18	21	21	$21n - 3$
\underline{f}_i	15	15	6	$15n - 9$	12	12	3	$12n - 9$
$\underline{\tau}_i$	42	42	24	$42n - 18$	33	33	16	$33n - 17$

Table 2: Operation count for NE dynamics variables.

and stored in a register for reuse for the serial and Binder/Herzog methods. This assumption can be made for the Binder/Herzog parallel algorithm because the forward variables per link are calculated serially, as in the serial algorithm.

With our method, it is possible for the recursion variables to be computed in parallel. However, this requires the redundant terms to be calculated by multiple processors. The

alternative is to have these terms each calculated by one processor and stored in the shared memory. Sharing terms would require synchronization to avoid illegal accesses and could cost more time than recomputation for a short calculation. This synchronization is one of the reasons that parallel speedup could not be linear. However, a primary goal of this work is to provide a more coarse-grain parallel algorithm than is generally possible with the conventional NE dynamics equations. We seek to execute large portions of the dynamics algorithm in parallel, rather than only executing a few individual addition and multiplication operations in parallel. We show in Table 3 the number of multiplications and additions required per variable when each is calculated independently of the others.

Term	number of Multiplications				number of Additions			
	$i = 1$	$1 < i < n$	$i = n$	Total	$i = 1$	$1 < i < n$	$i = n$	Total
$\underline{\omega}_i$	12	21	21	$21n - 9$	6	15	15	$15n - 9$
$\underline{\dot{\omega}}_i$	14	53	53	$53n - 39$	10	39	39	$39n - 29$
$\underline{\alpha}_i$	35	44	44	$44n - 9$	24	33	33	$33n - 9$
$\underline{a}_{e,i}$	27	36	36	$36n - 9$	18	27	27	$27n - 9$
$\underline{a}_{c,i}$	27	36	36	$36n - 9$	18	27	27	$27n - 9$
\underline{f}_i	15	15	6	$15n - 9$	12	12	3	$12n - 9$
$\underline{\tau}_i$	51	51	24	$51n - 27$	39	39	16	$39n - 23$

Table 3: Modified operation count for NE dynamics variables.

Using the data in the tables discussed above with some example cycle times, we determine the minimum length of the control loop in cycles for a three-link planar manipulator for the three algorithms under consideration. For performance analysis, we assume that floating point multiplications and additions require 4 and 3 cycles, respectively. In the serial case, summing the totals for each recursion variable with $n = 3$ gives a length of 3153 cycles (Figure 6). Using the Binder/Herzog method, each of n processors (P1 - Pn) executes

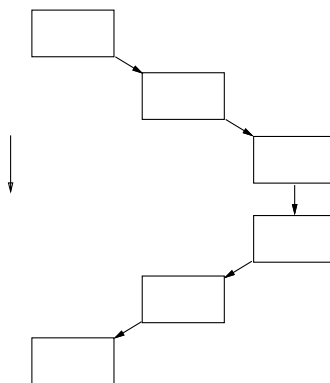


Figure 6: Total cycle time for serial algorithm for a 3-link planar manipulator.

the forward and backward recursions for its corresponding link. Figure 7 shows that the minimum cycle time for the control loop, i.e. the longest cycle time of all three processors, is 1220 cycles.

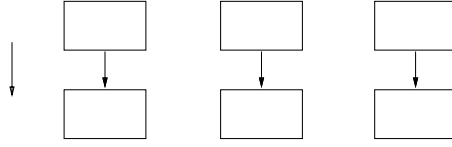


Figure 7: Total cycle time for Binder/Herzog algorithm for a 3-link planar manipulator.

The structure of the new algorithm allows each recursion variable to be computed in parallel, giving rise to a 21-processor system for this 3-link example (because there are seven recursion variables). Thus, the limiting factor is the single recursion variable with the longest cycle time ($\dot{\omega}_2$ requires 329 cycles). However, since the shortest cycle time is 33, for \underline{f}_3 , at least one processor would be idle for close to 300 cycles. Therefore, using a separate processor to calculate each recursion variable would not utilize the 21 processors in the most efficient manner. However, the high parallel potential of this algorithm allows development of the most efficient processor configuration. An example of a 4-processor architecture using our algorithm is shown in Figure 8. For this case, the entire forward and backward portions per

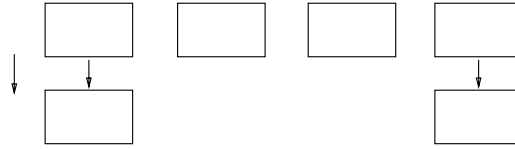


Figure 8: Total cycle time for the new algorithm for a 3-link planar manipulator.

link are executed on one processor (processor 1 performs the forward portion for link 1 and the backward portion for link 3, processor 4 performs the backward portions for links 2 and 3). Therefore, the cycle times corresponding to Table 2, where redundant computations are counted only once, were used. For this example, the minimum cycle time for the control loop is 857 cycles, which is an improvement over the Binder/Herzog method (see Table 4).

Speedup:	over serial algorithm	over Binder/Herzog
Serial (3153 cycles)		
Binder/Herzog (1220 cycles)	$3153/1220 = 2.58$	
New method:		
7n procs (329 cycles)	$3153/329 = 9.58$	3.71
4 procs (857 cycles)	$3153/857 = 3.68$	1.42

Table 4: Speedup of parallel dynamics algorithms for 3-link planar manipulator.

The method presented here is capable of providing a higher sampling rate than the other methods being compared by removing the data dependencies that exist in those algorithms. Simulation results shown in Section 7 demonstrate this improved performance. In addition to greater performance potential, this algorithm can improve the recovery rate from processor

Processor :	P1	P2	P3
	$\dot{\omega}_1$		ω_1
	$\underline{a}_{\epsilon,1}$	$\underline{a}_{\epsilon,1}$	$\underline{a}_{c,1}$
	$\dot{\omega}_2$		ω_2
	$\underline{a}_{\epsilon,2}$	$\underline{a}_{\epsilon,2}$	$\underline{a}_{c,2}$
time \Downarrow	$\dot{\omega}_3$		ω_3
	$\underline{a}_{\epsilon,3}$	$\underline{a}_{\epsilon,3}$	$\underline{a}_{c,3}$
			\underline{f}_3
		\underline{f}_2	$\underline{\tau}_3$
	\underline{f}_1	$\underline{\tau}_2$	
	$\underline{\tau}_1$		

Table 5: Example parallel conventional NE for 3-link manipulator.

failures, also due to the elimination of data dependencies. Fault tolerance in speculative computation methods is discussed in the following section.

6 Fault Tolerance Issues

In our previous work, we demonstrated the trade-off between controller speedup and processor fault tolerance in a multiprocessor robot control architecture. Logically, a program that executes failure detection and isolation routines frequently within the control loop will take longer to complete the loop than a program that executes these routines less often. However, the former program is more likely to detect a failure closer to the time that it occurred, meaning that less erroneous data has been produced by the failed processor and propagated further in the control algorithm. In a recursive NE algorithm where variables have several terms that are dependent upon previous calculations, a single failure can quickly corrupt numerous data. Therefore, when a failure is detected, all calculations of the failed processor and all calculations dependent upon those must be recomputed on the remaining functioning processors. This is one area where the use of speculative computation offers an advantage for fault tolerance. Since some data dependencies are relaxed due to the use of data from a prior iteration, recovery from a processor failure does not require as many calculations as recovery for a conventional method. This idea is expressed by example below.

Table 5 shows a possible parallel implementation of a conventional NE-based algorithm. The recursion variables are assigned to processors in a manner that distributes the workload based on the cycle times used above and data dependencies. The separating lines represent synchronization points. For this example, failure detection code may be implemented at each sync point or at the end of the loop. If failure detection is not implemented until the end of the control loop, and P3 is found to be faulty, then not only would the variables calculated on P3 have to be recomputed, but also the variables on the other processors that are dependent upon them – $\underline{a}_{\epsilon,3}$, \underline{f}_2 , \underline{f}_1 , $\underline{\tau}_2$, and $\underline{\tau}_1$. Implementing failure detection code at each sync would alleviate the problem of corrupting computations on other processors, but it would also adversely affect speed performance. However, when using speculative computation, failure detection code may be executed at the end of the loop without concern for contaminating

Processor :	P1	P2	P3
	$\underline{\omega}_1$	$\underline{\omega}_2$	$\underline{\omega}_3$
	$\underline{\dot{\omega}}_1$	$\underline{\dot{\omega}}_2$	$\underline{\dot{\omega}}_3$
	$\underline{\alpha}_1$	$\underline{\alpha}_2$	$\underline{\alpha}_3$
time \Downarrow	$\underline{a}_{e,1}$	$\underline{a}_{e,2}$	$\underline{a}_{e,3}$
	$\underline{a}_{c,1}$	$\underline{a}_{c,2}$	$\underline{a}_{c,3}$
	\underline{f}_1	\underline{f}_2	\underline{f}_3
	$\underline{\tau}_1$	$\underline{\tau}_2$	$\underline{\tau}_3$

Table 6: Binder/Herzog algorithm for 3-link manipulator.

Processor :	P1	P2	P3	P4
	$\underline{\omega}_1$	$\underline{\omega}_2$	$\underline{\omega}_3$	\underline{f}_2
	$\underline{\dot{\omega}}_1$	$\underline{\dot{\omega}}_2$	$\underline{\dot{\omega}}_3$	$\underline{\tau}_2$
	$\underline{\alpha}_1$	$\underline{\alpha}_2$	$\underline{\alpha}_3$	\underline{f}_1
time \Downarrow	$\underline{a}_{e,1}$	$\underline{a}_{e,2}$	$\underline{a}_{e,3}$	$\underline{\tau}_2$
	$\underline{a}_{c,1}$	$\underline{a}_{c,2}$	$\underline{a}_{c,3}$	
	\underline{f}_3			
	$\underline{\tau}_3$			

Table 7: Our algorithm for 3-link manipulator on 4 processors.

data on working processors because the computations use data from the previous control loop iteration. Thus, when P3 fails, only those variables calculated on P3 must be recomputed.

For the Binder/Herzog algorithm, the task (variable calculation) assignment is as shown in Table 6. If failure detection code is implemented at the end of each control loop, then we can be sure that a detected failure occurred in the current iteration and that data from previous loop iterations was not affected. Therefore, the variables for link 2 that use speculative data from link 3 do not have to be recomputed if P3 is found to be faulty. Thus, recovery is faster than for the conventional case. However, because the data dependencies within the dynamic equations for a link still exist in this algorithm, reconfiguration is restricted. Either the entire forward and backward recursions must be moved to the same processor (nearly doubling the minimum cycle time following a processor failure), or the variables can be rescheduled in parallel with more sync points added to maintain correctness.

In our new algorithm (shown in Table 7), variables are not assigned to particular processors, and they may be calculated in any order. Since all calculations use speculative data in all terms, only the variables computed on the failed processor must be recomputed. Furthermore, the variables that must be recalculated can be distributed among the remaining processors in any order for quicker recovery. The Binder/Herzog algorithm does not allow such flexibility, and thus would have a larger recovery time in any instance.

The most obvious advantage of the algorithm presented here over the Binder/Herzog algorithm is that it is designed to use a varying number of processors. Thus, if five processors are desired for hardware fault tolerance, for example, each processor can be utilized during normal operation for greater speedup. The Binder/Herzog algorithm is designed for one

processor per link. Therefore, processors added for fault tolerance beyond n are not used until a failure occurs and, therefore, cannot contribute to the performance during normal operation.

7 Results

We have performed simulations of the new algorithm proposed in this paper for a number of tasks. For each task, we simulated manipulator control with an inverse dynamics or ‘computed torque’ control algorithm using Newton-Euler dynamics in the controller. The results have been compared to those (for the same tasks) using the dynamics algorithms of Binder and Herzog and Yamakita et al., and also with a serial version of the controller using all current data.

In all cases, our results compared favorably with the other speculative data approaches. In the case of slow tasks with low frequency dynamics, the serial controller with no old data out-performed the speculative computation approaches, as expected. For tasks with significant high-frequency dynamics, the higher sample rate of the speculative computation approaches provides an advantage over the serial method. The Binder/Herzog algorithm (which uses less old data than the one proposed here) provided slightly better trajectory tracking than our method. However, it also exhibited some instability in some cases (see below).

Our approach is similar to the buffering method presented by Yamakita et al. Like our algorithm, their method uses prior data for all variables in the equations to allow maximum parallelism among the equations. However, due to the NE equations chosen for their algorithm, some terms are older than others in their computations (i.e. the data comes from two control loop iterations prior). This disparity may be the cause of our differing results: in their simulations, using prior data without correction terms produced diverging outputs; the output of our algorithm, with no correction terms, converged.

Although the Binder/Herzog algorithm produces slightly more accurate results than our NE-based algorithm at times, this method also exhibits instability under certain conditions. Numerous simulations demonstrate that the controller can cause oscillation about the desired trajectory for the robot joint accelerations, ultimately resulting in complete loss of control of the manipulator. On the other hand, simulation of our algorithm under the same initial conditions results in acceptable performance. Typical results are shown in Figures 9 and 10. The vertical line at $t = 1.57$ identifies where the algorithms begin using prior data.

8 Stability of Speculative Computation Methods

Simulations of our control algorithm have produced accurate results for a range of controller sampling rates. However, our simulations of the Binder/Herzog algorithm reveal that it does not always perform well for like ranges of sampling rates. The Binder/Herzog method produces results similar to our method for high sampling rates. Yet, for lower sampling rates, the Binder/Herzog algorithm yields oscillating and ultimately unstable output (note again Figures 9 and 10).

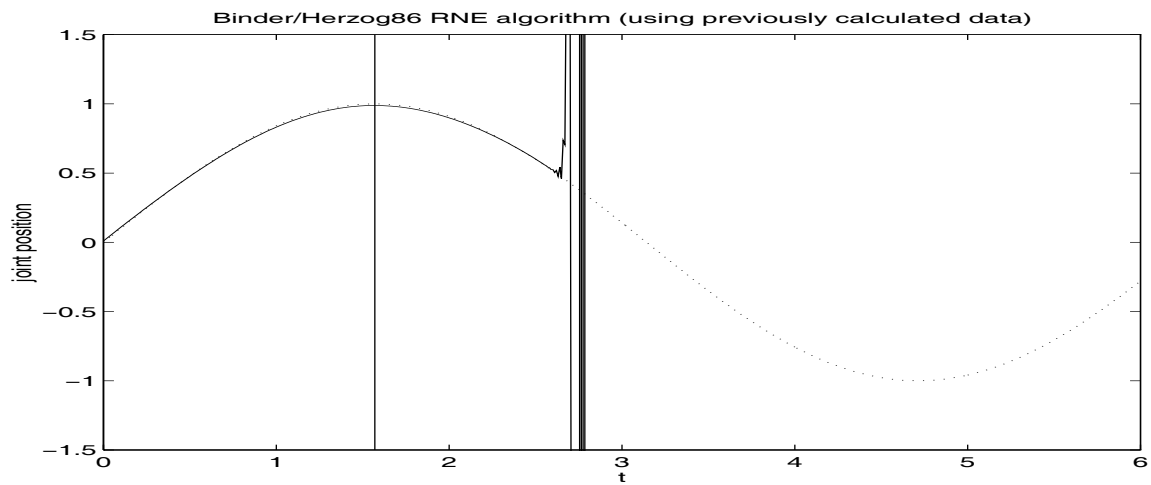


Figure 9: Final joint of 3-link planar robot - desired trajectory (dotted line) vs. actual (solid line).

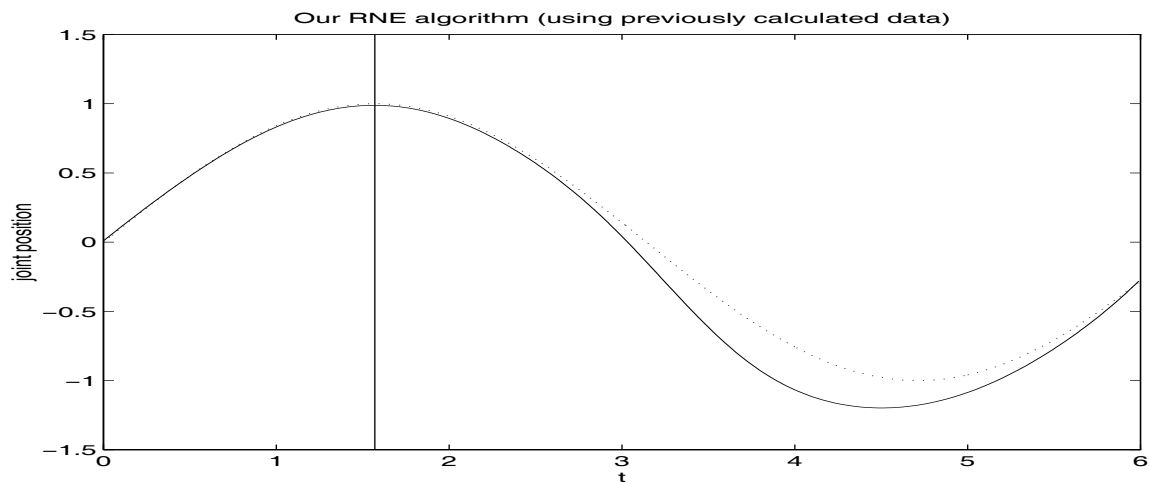


Figure 10: Final joint of 3-link planar robot - desired trajectory (dotted line) vs. actual (solid line).

Initially, this performance difference seems counter-intuitive—it seems logical that an algorithm using less data from prior iterations (and, thus, more current data), such as the Binder/Herzog algorithm, would be more accurate than a method (such as the one proposed in this paper) using more “old” data when the sampling rates are the same. However, recognizing that the key difference between the two algorithms is the amount and placement of prior data used, we are examining the stability effects of combining “current” data with “obsolete” data in different terms when computing the terms of the NE equations.

A similar phenomenon was described by Nigam and Lee in their presentation of a parallel pipeline architecture [17]. They note that manipulator oscillation can result if the sampling period is less than the time required to calculate the joint torques. In an m -stage pipeline, joint torques for times $t + 1$ through $t + m - 1$ are being calculated while the joint torque for time t is being applied. Thus, these future torques cannot consider the correction torques, and will then either overcompensate or undercompensate. In an effort to offset the errors, while still not having the proper correction terms, the controller will cause oscillation. Intuitively, it seems that this is the type of phenomenon that the Binder/Herzog method is experiencing.

A closer analysis of the terms in the controller yields additional insight. Assuming that in the time interval between consecutive iterations the joint positions remain constant (not unreasonable for typical joint motions and sampling rates), the map from the desired trajectory and the tracking errors to the control torques is linear. Note that for old data methods, the tracking error in the controller will be not only that for the present iteration, but also from the previous time step(s). The control mapping therefore has (potentially conflicting) inputs from the tracking errors of more than one sample period. It is clear that in some cases this situation can lead to instability. In fact, a closer inspection of the Binder/Herzog algorithm under the above assumptions reveals a marginally stable system.

If this combination of input data incorporating tracking errors from different iterations is the cause of the instability observed in the Binder/Herzog algorithm, then it is reasonable to assume that this is also the reason why the pure buffering method used by Yamakita et al. produced diverging results in their implementations. Since there are only two terms that mix data from different time steps in the Yamakita et al. method, it seems intuitively reasonable that the controller may be able to compensate for the errors better than in the Binder/Herzog method.

The algorithm proposed in this paper did not exhibit the same instability as did the Binder/Herzog algorithm. However, it is a speculative computation algorithm, and the above concern applies. A closer analysis of our algorithm reveals that (under the same assumptions as before) it is also marginally stable, but with much greater relative stability than the Binder/Herzog method.

The above discussion reveals much about the nature of “old” data methods, or speculative computation algorithms. From a computational speedup perspective, the methods offer significant promise for efficient parallel computation of manipulator dynamics. However, this is obtained at the cost of mixing old with current data in the controller. In the overwhelming majority of cases, this is not a problem. However, in a few cases (typically when the task contains high-frequency dynamics) the Binder/Herzog method exhibits instability. The method introduced in this paper remains stable for these cases, despite the fact that it uses

a greater amount of old data.

Clearly, it is the way that the old data is introduced that is important, as a more in-depth analysis revealed. Note that in general for reasonably fast sampling rates, none of the speculative computation methods exhibited problems. However, it is clear that care must be taken in general in applying methods using temporally mixed data. An in-depth stability analysis needs to be performed to identify ‘safe’ sampling rates which must be achieved for stability.

9 Conclusions

As technology advances, the tasks that are considered for robots become more demanding. In order to perform such tasks, which will incorporate high frequency dynamics, more computationally intensive control algorithms are proposed by researchers. The key bottleneck in robot control has been the computation of manipulator dynamics. Thus, development of efficient parallel algorithms for robot control is a logical direction for improvement.

One approach to speeding up the computation of manipulator dynamics is to release the serial bindings between the variables by using ‘old data’, calculated at previous timesteps in the controller iterations. In this paper, we introduce a new algorithm for parallel computation of manipulator dynamics using old data, using an approach termed speculative computation. The algorithm is shown to be more efficient than previous methods proposed in the literature, and exhibits better convergence than other algorithms without the use of correction terms. In addition, we identify and discuss a stability issue inherent in using ‘old data’ approaches. The algorithm presented in this paper is seen to be more stable than those presented previously.

Acknowledgements

This work was supported in part by NASA Graduate Student Fellowship NGT-70251, in part by the National Science Foundation under grant #IRI-9526363, and in part by DOE Sandia National Laboratory Contract #AL-3017.

References

- [1] A. Ambardar. *Analog and Digital Signal Processing*. PWS Publishing Company, Boston, 1995.
- [2] M. Amin-Javaheri and D.E. Orin. Non-Strict Computation of the Manipulator Inertia Matrix. In A. Fijany and A.K. Bejczy, editors, *Parallel Computation Systems for Robotics: Algorithms and Architectures*, pages 53–76. Academic Press, New York, 1992.
- [3] C.A. Balafoutis. A Survey of Efficient Computational Methods for Manipulator Inverse Dynamics. *Journal of Intelligent and Robotic Systems*, 9:45–71, 1994.
- [4] E.E. Binder. *Distributed Architecture and Fast Parallel Algorithms in Real-Time Robot Control*. PhD thesis, Oregon State University, Oregon, April 1985.

- [5] E.E. Binder and J.H. Herzog. Distributed Computer Architecture and Fast Parallel Algorithms in Real-Time Robot Control. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-16(4):543–549, 1986.
- [6] C.L. Chen, C.S.G. Lee, and E.S.H. Hou. Efficient Scheduling Algorithms for Robot Inverse Dynamics Computation on a Multiprocessor System. In *Proceedings of the 1988 IEEE International Conference on Robotics and Automation*, pages 1146–1151, Philadelphia, PA, 1988.
- [7] A. Fijany and A.K. Bejczy. A Class of Parallel Algorithms for Computation of the Manipulator Inertia Matrix. *IEEE Transactions on Robotics and Automation*, 5(5):600–615, 1989.
- [8] A. Fijany and A.K. Bejczy. Parallel Computation of Manipulator Inverse Dynamics. *Journal of Robotic Systems*, 8(5):599–635, 1991.
- [9] A. Fijany, I. Sharf, and G.M.T. D’Eleuterio. Parallel $\mathcal{O}(\log \mathcal{N})$ Algorithms for the Computation of Manipulator Forward Dynamics. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 1547–1553, San Diego, CA, 1994.
- [10] D.L. Hamilton. *Effectiveness and Performance Analysis of a Class of Parallel Robot Controllers with Fault Tolerance*. PhD thesis, Rice University, Houston, TX, May 1996.
- [11] D.L. Hamilton, J.K. Bennett, and I.D. Walker. Parallel Fault-Tolerant Robot Control. In *Proceedings of the 1992 SPIE Conference on Cooperative Intelligent Robotics in Space III*, pages 251–261, Boston, MA, 1992.
- [12] K. Hashimoto, K. Ohashi, and H. Kimura. An Implementation of a Parallel Algorithm for Real-Time Model-Based Control on a Network of Microprocessors. *International Journal of Robotics Research*, 9(6):37–47, 1990.
- [13] J.M. Hollerbach. A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-10(11):730–736, November 1980.
- [14] A. Joukhadar and Ch. Laugier. Dynamic Modeling and its Robotic Applications (*The Robot Φ System*). In *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, pages 2684 – 2689, Nagoya, Japan, 1995.
- [15] N. Kirćanski, T. Petrović, and M. Vukobratović. Parallel Computation of Symbolic Robot Models on Pipelined Processor Architectures. *Robotica*, 11:37–47, 1993.
- [16] C.S.G. Lee and P.R. Chang. Efficient Parallel Algorithms for Robot Inverse Dynamics Computation. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(4):532–542, July/August 1986.
- [17] R. Nigam and C.S.G. Lee. A Multiprocessor-Based Controller for Control of Mechanical Manipulators. *IEEE Journal of Robotics and Automation*, RA-1(4):173–182, 1985.
- [18] D.E. Orin, H.H. Chao, K.W. Olson, and W.W. Schrader. Pipeline/Parallel Algorithms for the Jacobian and Inverse Dynamics Computations. In *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, pages 785–789, St. Louis, MO, 1985.
- [19] M.W. Spong and M. Vidyasagar. *Robot Dynamics and Control*. Wiley, New York, 1989.

- [20] R.W. Toogood. Efficient Robot Inverse and Direct Dynamics Algorithms Using Micro-computer Based Symbolic Generation. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, pages 1827–1832, Scottsdale, AZ, 1989.
- [21] M. Vukobratović, N. Kirčanski, and S.G. Li. An Approach to Parallel Processing of Dynamic Robot Models. *International Journal of Robotics Research*, 7(2):64–71, 1988.
- [22] M. Vuskovic, T. Liang, and K. Anantha. Decoupled Parallel Recursive Newton-Euler Algorithm for Inverse Dynamics. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pages 832–838, Cincinnati, OH, 1990.
- [23] I.D. Walker and J.R. Cavallaro. Parallel VLSI Architectures for Real-Time Kinematics of Redundant Robots. In *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, pages 870–877, Atlanta, GA, 1993.
- [24] M. Yamakita, Y. Hoshino, K. Morimoto, and K. Furuta. Parallel Implementation of Newton-Euler Algorithm with One Step Ahead Prediction. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, pages 1842–1849, Sacramento, CA, 1991.
- [25] H. Zhang and R.P. Paul. A Parallel Inverse Kinematics Solution for Robot Manipulators Based on Multiprocessing and Linear Extrapolation. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pages 468–474, Cincinnati, OH, 1990.
- [26] A.Y. Zomaya and A.S. Morris. Parallel Computation of Robot Inverse Dynamics for High Speed Motions. *1992 IEE Proceedings-Part D*, CTA-139(2):226–236, 1992.

Appendix

Forward Recursion

$$\begin{aligned}
 \underline{\omega}_i &= (R_{i-1}^i)^T \underline{\omega}_{i-1} + (R_0^i)^T \underline{z}_{i-1} \dot{\theta}_i \\
 \underline{\dot{\omega}}_i &= (R_{i-1}^i)^T \underline{\dot{\omega}}_{i-1} + (R_{i-1}^i)^T \underline{\omega}_{i-1} \times (R_0^i)^T \underline{z}_{i-1} \dot{\theta}_i \\
 &\quad + (R_0^i)^T \underline{z}_{i-1} \ddot{\theta}_i \\
 \underline{\alpha}_i &= (R_{i-1}^i)^T \underline{\alpha}_{i-1} + (R_0^i)^T \underline{z}_{i-1} \ddot{\theta}_i \\
 &\quad + \underline{\omega}_i \times (R_0^i)^T \underline{z}_{i-1} \dot{\theta}_i \\
 \underline{a}_{e,i} &= (R_{i-1}^i)^T \underline{a}_{e,i-1} + \underline{\dot{\omega}}_i \times r_{i,i+1} \\
 &\quad + \underline{\omega}_i \times (\underline{\omega}_i \times r_{i,i+1}) \\
 \underline{a}_{c,i} &= (R_{i-1}^i)^T \underline{a}_{c,i-1} + \underline{\dot{\omega}}_i \times r_{i,ci} + \underline{\omega}_i \times (\underline{\omega}_i \times r_{i,ci}) \\
 \underline{g}_i &= -(R_0^i)^T (G_j)
 \end{aligned}$$

Backward Recursion

$$\begin{aligned}
 \underline{f}_i &= R_i^{i+1} \underline{f}_{i+1} + m_i \underline{a}_{c,i} - m_i \underline{g}_i \\
 \underline{\tau}_i &= R_i^{i+1} \underline{\tau}_{i+1} - \underline{f}_i \times r_{i,ci} + (R_i^{i+1} \underline{f}_{i+1}) \times r_{i+1,ci} \\
 &\quad + I_i \underline{\alpha}_i + \underline{\omega}_i \times (I_i \underline{\omega}_i)
 \end{aligned}$$

- R_j^i rotation matrix translating from
frame j to frame i ,
 $\underline{\omega}_i$ angular velocity of link i ,
 $\underline{\dot{\omega}}_i$ derivative of $\underline{\omega}_i$,
 $\underline{\alpha}_i$ angular acceleration of link i ,
 $\underline{a}_{e,i}$ linear acceleration of end of link i ,
 $\underline{a}_{c,i}$ linear acceleration of center of link i ,
 \underline{g}_i acceleration due to gravity for link i ,
 G_j gravitational constant,
 \underline{f}_i force acting on link i ,
 $\underline{\tau}_i$ torque acting on link i