

# 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

*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. 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 'stale' data is an effective solution to the speedup problem, but presents some special difficulties. One stability issue when using 'stale' data that is encountered in previous approaches is discussed here, along with a partial solution to the problem.*

## 1 Introduction

As robots are used for more difficult 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 [2, 3].

Most robot controllers currently in use are uniprocessor systems. In these systems, the opportunity for speed improvement is limited by the total number of mathematical operations required. However, development of multiprocessor architectures has broadened the possibilities for providing real-time robot control. Ideally, the most efficient uniprocessor algorithms can be parallelized and executed on multiple processors, achieving linear speedup. However, due to communication and synchronization requirements, this goal is seldom achieved.

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 a state-of-the-art serial architectures to be extended to allow other important tasks, such as fault detection, to be executed in real time.

A variety of multiprocessor architectures to support parallel robot control algorithms have been studied [4, 9, 10, 12, 14, 20]. However, due to the dependencies that exist in the robot dynamics equations, they are not easily parallelized. More recent research efforts in robot kinematics and dynamics consider development of efficient parallel versions of these algorithms [4, 5, 6, 8, 13, 15, 17, 18, 22, 23]. Smaller tasks, such as computation of the inertia matrix, have also been parallelized [7].

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:

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

where  $\boldsymbol{\theta}$  is the  $n \times 1$  vector of joint angles for an  $n$ -joint robot,  $\boldsymbol{\tau}$  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 [16]. 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: the 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 [16]. (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 coupled, 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, computations 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 [11].

The coupled nature of the robot is evident in the duplicate computations in the LE dynamics equations. However, the structure of these equations does not incorporate the type of data dependencies that limit parallelism in the NE algorithm. The classic Lagrangian is appreciated for its closed-form, while the recursive Newton-Euler form allows faster calculation of the dynamics equations. 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 [2].

A recursive Lagrangian-based dynamics form was developed by Hollerbach [11] 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 [11]. 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. We term this approach speculative computation. This allows the calculations that are typically executed serially to be executed in parallel. This method 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. This would imply that the controller output when using the “stale” data will be effective. As long as the control equations are executed quickly, this assertion holds. Since using data from the previous iteration allows the computations to be done in parallel, 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 [4]. Further work using the method in [4] was reported in [19] in 1990. However, apart from the interesting results of [21], whose approach has some similarities with our own, it does not appear that there has been much investigation of ‘old data’ for the entire dynamics calculations since this early work. However, the basic concept was applied to the inertia matrix computations in [1] in the

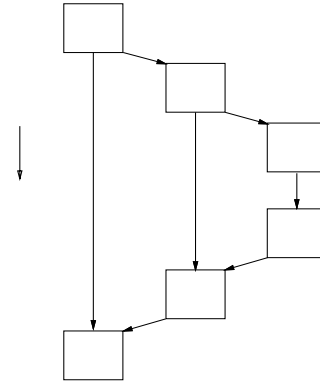


Figure 1: Flow diagram of conventional NE algorithm for a 3-link robot

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 4, we discuss the observed instability, and describe why our algorithm is significantly more stable.

## 2 Parallel Robot Control Using Speculative Computation

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 [4]. 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 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

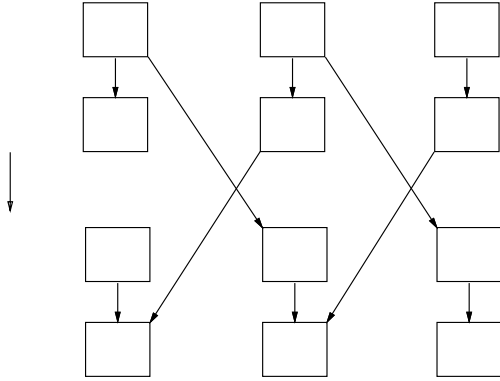


Figure 2: Flow diagram of Binder/Herzog algorithm for a 3-link robot

speed over the conventional approach while maintaining a higher level of accuracy than “zero-order” and “first-order prediction [3].” 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 [4]. 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 [4]. 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.

Vuskovic et al. [19] 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 rate. 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 other recursion variables that are used to calculate the torques [19]. Experimental results showed small approximation er-

rors 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. [21] 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). Yamakita et al. reported that the buffering causes delays in the controller sampling rate, which result in some performance degradation. As discussed in the previous paragraph, errors decrease (and increase) with the controller sampling rate. The simulation results indicate that buffering alone results in diverging output, especially for joints near the end effector.

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 [21].

### 3 Our Approach

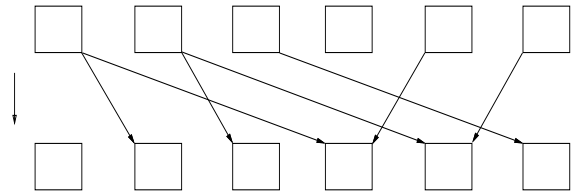


Figure 3: Flow diagram of our algorithm (and Yamakita et al.) for a 3-link robot

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 [4], Vuskovic et

al. [19], and Yamakita et al. [21] 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 (prior) values of some data in our NE control algorithm. We simulate a computed torque PD controller [9].

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 values are shown in Table 1 as the single- and double-boxed variables. This method allows greater parallelism because synchronization between term calculations 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, but with less impact, for the backward recursion. Theoretically, all terms of the forward and backward recursions for each link could be calculated in parallel (i.e.  $\omega_i$  through  $\tau_i$ , 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 an advantage of our method.

### 3.1 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

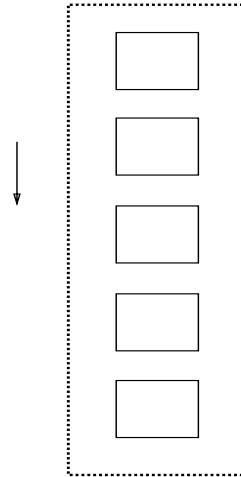


Figure 4: Flow diagram of parallel forward recursion for Binder/Herzog method

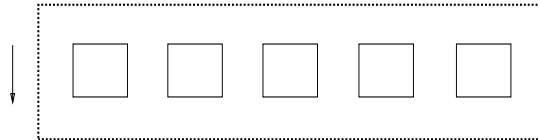


Figure 5: Flow diagram of parallel forward recursion for our method

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 al-

$$\begin{aligned}
& \text{Forward Recursion} \\
\underline{\omega}_i &= (R_{i-1}^i)^T \boxed{\underline{\omega}_{i-1}} + (R_0^i)^T \underline{z}_{i-1} \dot{\theta}_i \\
\underline{\dot{\omega}}_i &= (R_{i-1}^i)^T \boxed{\underline{\dot{\omega}}_{i-1}} + (R_{i-1}^i)^T \boxed{\underline{\omega}_{i-1}} \times (R_0^i)^T \underline{z}_{i-1} \dot{\theta}_i + (R_0^i)^T \underline{z}_{i-1} \ddot{\theta}_i \\
\underline{\alpha}_i &= (R_{i-1}^i)^T \boxed{\underline{\alpha}_{i-1}} + \boxed{\underline{\omega}}_i \times (R_0^i)^T \underline{z}_{i-1} \dot{\theta}_i + (R_0^i)^T \underline{z}_{i-1} \ddot{\theta}_i \\
\underline{a}_{e,i} &= (R_{i-1}^i)^T \boxed{\underline{a}_{e,i-1}} + \boxed{\underline{\dot{\omega}}}_i \times \underline{r}_{i,i+1} + \boxed{\underline{\omega}}_i \times (\boxed{\underline{\omega}}_i \times \underline{r}_{i,i+1}) \\
\underline{a}_{c,i} &= (R_{i-1}^i)^T \boxed{\underline{a}_{c,i-1}} + \boxed{\underline{\dot{\omega}}}_i \times \underline{r}_{i,ci} + \boxed{\underline{\omega}}_i \times (\boxed{\underline{\omega}}_i \times \underline{r}_{i,ci}) \\
& \text{Backward Recursion} \\
\underline{f}_i &= R_i^{i+1} \boxed{\underline{f}_{i+1}} + m_i \underline{a}_{c,i} - m_i \underline{g}_i \\
\underline{\tau}_i &= R_i^{i+1} \boxed{\underline{\tau}_{i+1}} - \boxed{\underline{f}}_i \times \underline{r}_{i,ci} + (R_i^{i+1} \boxed{\underline{f}_{i+1}}) \times \underline{r}_{i+1,ci} + I_i \underline{\alpha}_i + \boxed{\underline{\omega}}_i \times (I_i \boxed{\underline{\omega}}_i)
\end{aligned}$$

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

gorithm 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 6 and 7. The vertical line at  $t = 1.57$  identifies where the algorithms begin using prior data.

#### 4 Stability Issues of Speculative Computation

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 6 and 7).

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 [14]. 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.

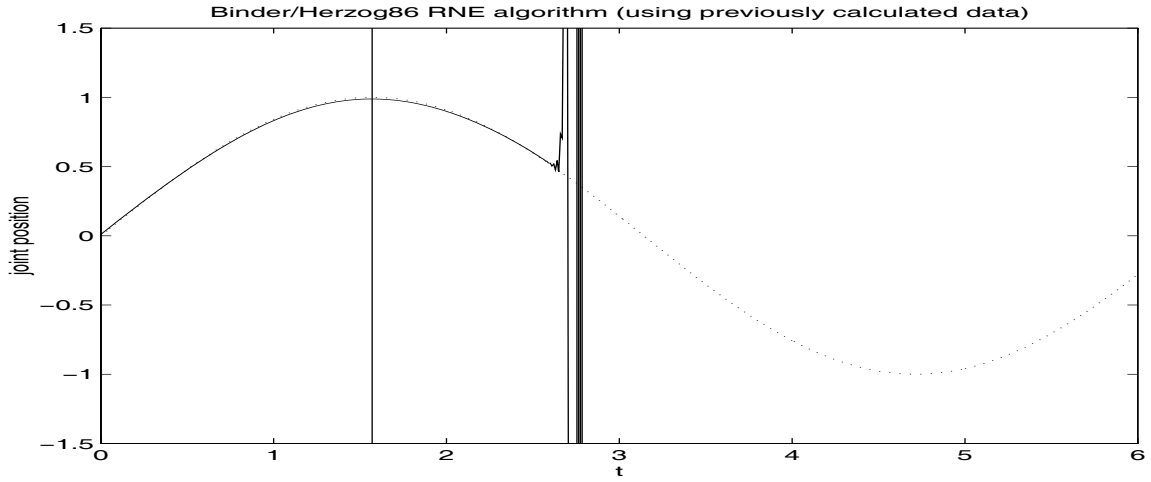


Figure 6: Final joint of 3-link planar robot - desired trajectory (dotted line) vs. actual (solid line)

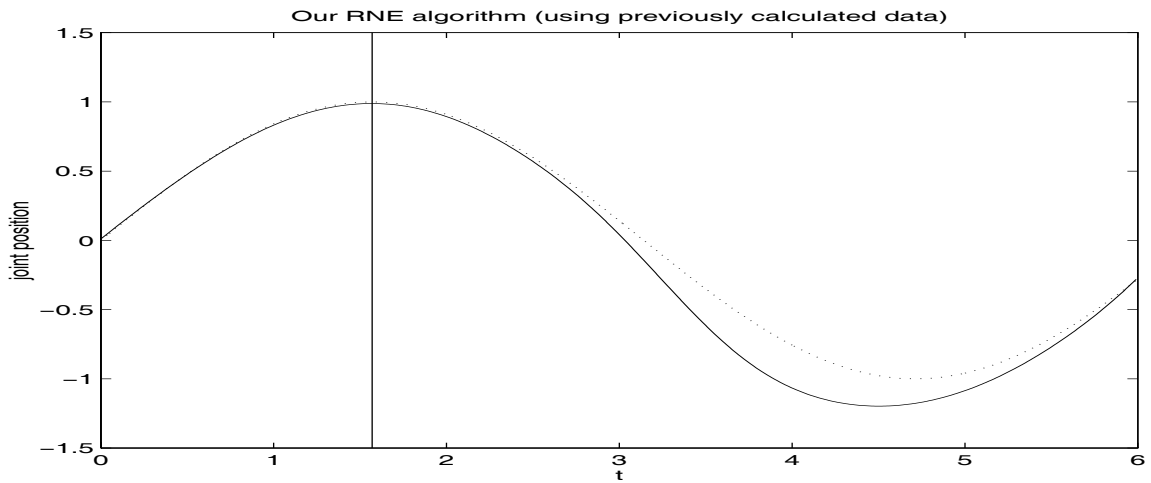


Figure 7: Final joint of 3-link planar robot - desired trajectory (dotted line) vs. actual (solid line)

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’, 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.

## 5 Conclusion

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 calculations. 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 the 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] 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, 1992.
- [2] C.A. Balafoutis. A Survey of Efficient Computational Methods for Manipulator Inverse Dynamics. *Journal of Intelligent and Robotic Systems*, 9:45–71, 1994.
- [3] E.E. Binder. *Distributed Architecture and Fast Parallel Algorithms in Real-Time Robot Control*. PhD thesis, Oregon State University, Oregon, April 1985.
- [4] 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.
- [5] 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 *Proc. of 1988 IEEE International Conference on Robotics and Automation*, pages 1146–1151, Philadelphia, PA, 1988.
- [6] A. Fijany and A.K. Bejczy. Parallel Computation of Manipulator Inverse Dynamics. *Journal of Robotic Systems*, 8(5):599–635, October 1986.
- [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, I. Sharf, and G.M.T. D’Eleuterio. Parallel  $\mathcal{O}(\log \mathcal{N})$  Algorithms for the Computation of Manipulator Forward Dynamics. In *Proc. of 1994 IEEE International Conference on Robotics and Automation*, pages 1547–1553, San Diego, CA, 1994.
- [9] D.L. Hamilton, J.K. Bennett, and I.D. Walker. Parallel Fault-Tolerant Robot Control. In *1992 SPIE Conference on Cooperative Intelligent Robotics in Space III*, pages 251–261, Boston, MA, November 1992.
- [10] 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.

- [11] 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.
- [12] N. Kirčanski, T. Petrović, and M. Vukobratović. Parallel Computation of Symbolic Robot Models on Pipelined Processor Architectures. *Robotica*, 11:37–47, 1993.
- [13] 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.
- [14] 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.
- [15] D.E. Orin, H.H. Chao, K.W. Olson, and W.W. Schrader. Pipeline/Parallel Algorithms for the Jacobian and Inverse Dynamics Computations. In *1985 IEEE International Conference on Robotics and Automation*, pages 785–789, St. Louis, MO, 1985.
- [16] M.W. Spong and M. Vidyasagar. *Robot Dynamics and Control*. Wiley, New York, 1989.
- [17] R.W. Toogood. Efficient Robot Inverse and Direct Dynamics Algorithms Using Micro-computer Based Symbolic Generation. In *Proc. of 1989 IEEE International Conference on Robotics and Automation*, pages 1827–1832, Scottsdale, AZ, 1989.
- [18] 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.
- [19] M. Vuskovic, T. Liang, and K. Anantha. Decoupled Parallel Recursive Newton-Euler Algorithm for Inverse Dynamics. In *1990 IEEE International Conference on Robotics and Automation*, pages 832–838, Cincinnati, OH, 1990.
- [20] I.D. Walker and J.R. Cavallaro. Parallel VLSI Architectures for Real-Time Kinematics of Redundant Robots. In *Proceedings of 1993 IEEE International Conference on Robotics and Automation*, pages 870–877, Atlanta, GA, 1993.
- [21] M. Yamakita, Y. Hoshino, K. Morimoto, and K. Furuta. Parallel Implementation of Newton-Euler Algorithm with One Step Ahead Prediction. In *Proc. of 1991 IEEE International Conference on Robotics and Automation*, pages 1842–1849, Sacramento, CA, 1991.
- [22] H. Zhang and R.P. Paul. A Parallel Inverse Kinematics Solution for Robot Manipulators Based on Multiprocessing and Linear Extrapolation. In *Proc. of 1990 IEEE International Conference on Robotics and Automation*, pages 468–474, Cincinnati, OH, 1990.
- [23] 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 $i$