Technical Report 10/97, University of Karlsruhe,
Computer Science Department, ISSN 1432-7864

A Review of Parallel Processing Approaches to Robot Kinematics and Jacobian

Dominik HENRICH, Joachim KARL und Heinz WÖRN

Institute for Real-Time Computer Systems and Robotics
University of Karlsruhe, D-76128 Karlsruhe, Germany
e-mail: dHenrich@ira.uka.de[1]

Abstract

Due to continuously increasing demands in the area of advanced robot control, it became necessary to speed up the computation. One way to reduce the computation time is to distribute the computation onto several processing units. In this survey we present different approaches to parallel computation of robot kinematics and Jacobian. Thereby, we discuss both the forward and the reverse problem. We introduce a classification scheme and classify the references by this scheme.

Keywords: parallel processing, Jacobian, robot kinematics, robot control.

1 Introduction

Due to continuously increasing demands in the area of advanced robot control, it became necessary to speed up the computation. Since it should be possible to control the motion of a robot manipulator in real-time, it is necessary to reduce the computation time to less than the cycle rate of the control loop. One way to reduce the computation time is to distribute the computation over several processing units.

There are other overviews and reviews on parallel processing approaches to robotic problems. Earlier overviews include [Lee89] and [Graham89]. Lee takes a closer look at parallel approaches in [Lee91]. He tries to find common features in the different problems of kinematics, dynamics and Jacobian computation. The latest summary is from Zomaya et al. [Zomaya96]. They summarize by saying that the field of research is still widespread and can be roughly divided into two categories: approaches using general-purpose processing units and approaches using customized chips. Their conclusion is that further research in this field is still required.

---

1 Further information can be found on the Web pages of the PaRo group (Parallel Robotics) of the IPR at http://wwwipr.ira.uka.de/~paro/

Here, we show how the computation of robot control can be done in parallel and try to give a complete overview. In Section 2, we introduce a classification scheme. In Section 3, we discuss the achievements in the field of parallel computation of robot kinematics. The parallel computation of Jacobian is discussed in Section 4. Both of the last two sections are again split up into the forward and the reverse problem. In the summary, we give an overview of parallel computation and point out the most efficient approaches to robot control.

2 Classification Scheme

To review the literature of the past years, we will introduce a classification scheme, which allows us to structure and compare the approaches up to now (Table I). With each reference in the list at the end of this article, there is a list of keywords indicating the classification of the work.

The classification scheme is divided into a couple of aspects. The aspect *Problem* refers to the problem that is treated in the reference. The aspect *level of parallelism* indicates on which level the algorithm exploits the parallelism. The aspect *Interconnection topology* indicates about the logical or physical topology of the communication network. The aspect *processing paradigm* describes which basic approach is used in parallel processing. The aspect *Scalability* is used to determine whether the design is scalable by adding or removing processing units or whether it is designed for a fixed number of processors. Other aspects include *coordinate rotation digital computer (CORDIC) arithmetic* that cover a large number of references. But there are approaches that are only used in very few references and that are introduced to show that there is an alternative.

We introduce four levels of parallelism with decreasing granularity as described in [Sadayappan89, Zomaya92]. If several problems (e.g., forward kinematics and forward Jacobian) are solved in parallel, we call it parallelism on the task level. The characteristic of parallelism on the joint level is that the computations are done for all joints at once. Another level involves the exploitation of parallelism in the computation chain, e.g. if there are several independent equations. The fine-grained parallelism is the parallelism on the operation level, such as matrix or matrix/vector multiplication etc.

We classify the *processing paradigm* in *multiple instruction, multiple data (MIMD), single instruction multiple data (SIMD)* and *systolic architectures*. There is a slight difference between SIMD and systolic architectures. In SIMD, the same instruction is applied to many pieces of data at the same time and the instruction stream is a stream with respect to time, whereas in systolic architectures, several instructions are applied to many pieces of data at the same time, but the instructions in each processing unit depend only on the arrival of the data. So there is nothing like an instruction stream with respect to time.

We also distinguish between static and dynamic load balancing. If static load balancing is used, the task schedule is computed before the program is started. Dynamic load balancing also considers the current workload of the processors and different communication times at run-time. In the literature, both possibilities are called scheduling.

Aspect	**Keywords**
Problem	forward kinematics reverse kinematics forward Jacobian reverse Jacobian
Level of parallelism	task level joint level computation chain level operation level
Interconnection topology	array tree complete mesh pipeline hypercube special purpose
Processing paradigm	MIMD SIMD systolic architectures
Scalability	scalable not scalable
Others	CORDIC dynamic load balancing static load balancing VLSI line-oriented robot model

Table I: List of Keywords for the classification of the different approaches to parallel computation of robot kinematics and Jacobian.

3 Parallel Kinematics

3.1 Forward kinematics

In order to determine the current position and orientation of the end-effector, it is necessary to compute matrix multiplications and trigonometric functions. The matrices are 4x4 following

the notation of Denavit-Hartenberg. Altogether, this computation defines a function from the joint angle set to the end-effector position and orientation.

For a manipulator with *n* joints, there are *n* matrices to multiply. It is possible to do these *n*-1 matrix-multiplications in parallel (joint level parallelism). It is also possible to do a matrix-multiplication in parallel, since the matrices have 4 rows and columns, which can be multiplied independently (operation level parallelism). The computation, the trigonometric functions, cannot be easily done in parallel. But there are also approaches that go in this direction.

In the following, we present the different approaches to the operation level, computation chain level, joint level and task level.

Hamisi and Fraser exploit the parallelization possibilities on the operation level in [Hamisi89]. They use a transputer network with seven T414s. In this article, the developing process and design of the MIMD computation, analysis and optimization is shown in detail. Because of limited transputer communication paths, their system is designed with only seven Transputers. Due to the small number of processors, the process is mapped onto the processing units (PEs) without an extra scheduling mechanism. Since they compute the forward kinematics incrementally, they compute the forward kinematics by computing the forward and reverse Jacobian in parallel. Hence, there is also parallelism on the task level. (See Section 4 for Jacobian.) For a six degree-of-freedom (DOF) manipulator, the forward kinematics is computed in 8.4 ms.

Another approach exploiting parallelism on the operation level can be found in [Funda90]. Funda and Paul test several distinct line-oriented representations of a spatial screw displacement. They consider line-based geometry of screws as a valuable tool for transforming lines (for example joint axes) between coordinate frames (of the joints). They obtain parallel solutions with up to 64 PEs and a speed-up by a factor of 12 to 38. One of the conclusions is that the point-based representations are simple, compact and computationally very efficient, and are therefore preferable in real-time applications in robotic control. This approach should be studied more closely, as it was never proved that line-oriented approaches are slower on special-purpose architectures.

Another article dealing with operation level parallelism is [Fijany92]. Fijany and Bejczy offer an architecture that uses parallelization on the joint and operation level. They combine the computation of forward kinematics with the computation of the Jacobian. The design called Algorithmically Specialized Parallel Architecture for Robotic Computations (ASPARC) is a MIMD architecture on the processing unit level and each processing unit is a SIMD architecture to exploit the parallelism on the operation level. The speed-up from the joint level parallelism is 2.3 and the overall speed-up is 9.6 for a six DOF manipulator.

Walker and Cavallaro use an array of CORDIC PEs to compute forward kinematics in [Walker93]. The coordinate transformation between the single joint frames are done in

CORDIC PEs as vector rotations, so they exploit operation level parallelism. Since they compute all four types of problems at the same time, they make use of task level parallelism and achieve total cycle rates of 400 µs.

There is only one implementation of forward kinematics on a single VLSI chip. This architecture is developed in [Leung87]. Leung and Shanblatt exploit parallelism on the operation level to achieve a cycle rate of about 10 µs for a six DOF manipulator. They conclude that the kinematic computation is no longer the bottleneck, but rather the data communication.

Barhen uses a scalable hypercube architecture for robot control. Barhen presents an implementation for robot dynamics but the architecture is also suitable for the computation of robot kinematics [Barhen87]. Static and dynamic load balancing techniques are used for distributing the computation tasks on the processors. In this way he considers the data transfer caused by the fact that the processors are not equally suited for the tasks since data transfer to neighbour processors is cheaper than data transfer to further processors.

Similar to this, Ahmad and Li propose a robot controller based on a multiple arithmetic processing unit (APU) in [Ahmad87]. They claim that special-purpose architectures are expensive to design and inflexible, so they decide to use a MIMD architecture similar to [Hamisi89]. With APUs or microprocessors able to execute floating point instructions in 100 ns, it is possible to design a fast multi-processor network. Also, such a design may be updated with faster processor types as they come onto the market. The authors present simulation results for a PUMA manipulator computed with a MC 68881 Arithmetic Coprocessor. They ignore the communication overhead and get a computation time with 8 APUs of 77 µs with a multiplying time of 6 µs and a speed-up of 6.9. They use a scheduling algorithm of the type depth-first/implicit heuristic search (DF/IHS) for the distribution of the computation on multiple APUs. In [Ahmad89], they modify the DF/IHS algorithm and consider the communication overhead and even a possible contention. They compare several scheduling algorithms and conclude that the depth-first/minimized overhead heuristic search (DF/MOHS) is more suitable for forward kinematics since it minimizes the data transfer. Under the same conditions, they now get a computation time of 95 µs. Another result of their simulations is that there is an optimal number of processors, since the communication time increases with an increasing number of processors.

Pedicone and Johnson present a Concurrent Processor Architecture for Control (CPAC) in [Pedicone87]. They design this special purpose architecture with two concurrent processors. One is a continuous processing element (CPE), the other is a discrete processing element (DPE). The CPE is a vector signal processor for computing the joint angles, the DPE controls discrete inputs and outputs. It also monitors, and if necessary intervenes in, the operation of the CPE. Since the CPE computes the joint angles for all joints at the same time, this design exploits the joint level parallelism.

Further exploitation of parallelism at the joint level is described in [Lin91]. Lin and Lee try to compute all the kinematic and dynamic problems with one architecture. The characteristics of the kinematic, Jacobian and dynamic algorithms are analyzed and the common features are extracted. Starting from this, they propose a reconfigurable dual network SIMD with *n* PEs. Thus, they are on the joint level of parallelization. They are able to solve all kinematic and dynamic problems by changing the configuration of the network by software. Another aim of their work is to make the network fault-tolerant.

An article exploiting the parallelism on the task level besides [Walker93] is [Chen86]. Chen et al. present a multiprocessor architecture with a bus network using five PEs as a linear array. While a single floating point operation needs 6 µs, their kinematics computation achieves rates of 150 Hz. They divide orientation and position computations. With five PEs, there are two for orientation, two for position and one as the "Programming level". Their system manages the dynamics computation at the same time. They achieve a relative speed-up from 3 to 5 PEs of 1.36.

The only articles dealing with scheduling extensively are [Ahmad87] and [Ahmad89]. However, Ahmad and Li also achieve speed-ups of 7.6, which is less than the predicted speed-up of 24 calculated by the product of the joint number (6) and the number of fine grained operations done in parallel (4). With a higher number of PEs, no further speed-up is possible. This indicates that scheduling is not necessary for forward kinematics and that parallelism can be more easily achieved by exploiting the parallelism on several levels, like in [Fijany92].

None of the speed-ups are higher than 10 in the presented approaches. Parallelization on the joint level will achieve a maximum speed-up of *n* for a manipulator with *n* joints. Most approaches parallelize on the operation level. On this level, the maximum speed-up is four due to the representation as 4x4 matrices. Therefore, for common industrial robots like PUMA or Stanford arm, there will be a speed-up of less than 4 *n*. Since most of the articles address only one level of parallelism the actual speed-up is much less.

3.2 **Reverse Kinematics**

The task of reverse kinematics is to compute the joint angle set from the position and orientation of the end-effector. This is the inverse of the function defined in the previous section. There are several ways to compute this inverse function as described by Lee and Chang in [Lee87]: inverse transform (closed form), screw algebra, dual matrices, dual quaternion, iterative, and the geometric approach. In the following, we present the iterative form and the closed form, where the closed form references are classified as to whether they use the CORDIC architecture or not.

. The iterative way, as proposed by [Tsai85], has not been further investigated for parallel processing. This is because the iterative solution requires more computations than the closed

form. Additionally, neither guarantees convergence nor is it suited for parallelizing since the next iteration step can be computed as soon as the previous one has been finished.

For the most common industrial robots like PUMA, Stanford, ASEA or MINIMOVER, there is the solution for the joint angle set in closed form. In this case, the computation is much easier because the problems of existence and uniqueness are solved. In [Lee87], the necessary computations are given: multiplications, additions, square root, and transcendental function operations. All these are floating point operations, and so the main goal should be to speed up the computation of these floating point operations rather than to speed up the matrix operations as for forward kinematics. They propose that a CORDIC architecture should do the computation work, and they present an architecture with a latency time of 720 μs and a new computation result every 40 μs. They decompose the joint angle equations manually in computational modules, which are executed by a CORDIC processor. Then, the acyclic directed graph resulting from the data dependencies is balanced by inserting delay buffers. By using special tapped-delay-line-buffers, the number of buffers resulting in a maximum pipelined design is reduced.

In the same year, Wang and Butner presented a new architecture in [Wang87], for which they also used a CORDIC subsystem, which computes new results every 50 μs. A special purpose processor is designed to do the computational work. Up to four of these can work in parallel, and hence parallelize on the highest level. The processors can be controlled independently, so that besides reverse kinematics, other computations can be done on idle processors. A minimal configuration consists of one processor and can control up to ten synchronized axes. The maximal configuration has four processors and can control up to 40 axes. Since there are three separate buses and three independent data paths in the special purpose processor, it parallelizes the vector additions and multiplications (operation level parallelism). But there is a CORDIC subsystem which does the expensive floating point and trigonometric evaluations efficiently.

The authors of [Harber88] use the results of [Lee87] and enhance them to construct the system on a single VLSI chip. The original work has been modified to allow the use of bit-serial CORDIC processors. The computation time is reduced and the throughput of the pipeline is increased, since normalization of the output of a CORDIC processor is avoided. They do the normalization not as usual done by introducing additional CORDIC iterations, but by adding stand-alone bit-serial constant multipliers, which do this in less time.

Another architecture using CORDIC processors is the one by Walker and Cavallaro [Walker93]. They use an array of 16 CORDIC processors as a systolic array and combine it with a general-purpose processor. They compute the reverse kinematics in parallel by computing the Jacobian matrix and its pseudoinverse in parallel. Then they use a ‘resolved-rate’ approach to compute the joint velocities from the pseudoinverse Jacobian in the general-purpose

processor. Therefore, matrix-vector multiplications have to be executed. The desired joint angles can be obtained from the current joint velocities and joint angles. The array is also able to perform the computation of forward kinematics. Thus, all kinematic and Jacobian computations can be done by this architecture. They have a higher flexibility than [Wang87] because they use the same systolic array for multiple algorithms and, thus, the CORDIC processors are no longer a part of one pipeline but of many. It is more suitable than most approaches since they do not depend on an existing closed form solution and their architecture is able to manage non-redundant manipulators. They implement the systolic array in VLSI since they achieve shorter computation times with it. Using this, they exploit the parallelism at the task level and at the operation level.

The authors of [Ahmad87] and [Ahmad89] propose a MIMD architecture with static load balancing. They compare different scheduling algorithms and compute the reverse kinematics in about 400 µs. They use a scalable number of arithmetic processing units (APU), so that they can adapt to closed form solutions for different manipulators independent of the number of joints. As mentioned in Section 3.1, they parallelize at the computation chain level. The authors Zhang and Paul also apply the MIMD processing paradigm in the references [Zhang90] and [Zhang91]. They use the closed form solution to compute the reverse kinematics. They extrapolate the joint angles for the current computation from that from earlier computations, in order to remove the data dependencies between the single equations. With this loss of dependency the equations can be easily done in parallel. Not only do they try to increase the data throughput as in [Lee87], but also to reduce the latency time. They use a multiprocessor system with one computing processor for each joint and one supervisor. They achieve a speed-up of approximately 2.

For parallel manipulators, Gosselin exploits the mechanical architectur to obtain parallel computational algorithms [Gosselin96]. For the reverse kinematics, he obtains an independent equation for each joint computing the length of each manipulator with given position and orientation of the platform. Thus, this is joint level parallelism. He achieves a theoretical speed-up of n.

The last article [Nabhan95] in this section deals with the general parallelization of robotic computational tasks. Nabhan and Zomaya use a graph-based approach to map computational tasks on a MIMD architecture. Their algorithm automatically generates the task graph for a given architecture. The algorithm considers the network topology and communication constraints, making it scalable for a different number of processors. The whole parallelizing process is done automatically: For a chosen robot manipulator, the computational model is generated and simplified; a task graph is generated, compressed and the tasks are assigned to the processors. They develop a special annealing algorithm, since the dynamic-highest-level-first/most-immediate-successors-first (DHLF/MISF) is time consuming.

Other computation methods than the iterative or the closed form are listed in [Lee87]. None seem to be interesting enough to encourage further research work.

4 Parallel Jacobian

4.1 Forward Jacobian

The computation of the forward Jacobian is similar to the forward kinematics, so many references deal with both problems. The velocity of the end-effector should be computed from the joint rates. This requires coordinate transformations from one joint to every other joint. Coordinate transformations require, for example, the multiplication of matrices as introduced by Denavit and Hardenberg.

There are three main approaches to the forward Jacobian: General-purpose architectures not specially designed for the problem of robot control, spezialized architectures with general-purpose processors or arithmetic units, and architectures especially designed for robot control with microprocessors that are not off-the-shelf.

In the following, we will discuss these three approaches of robot arm control. The early approaches using general-purpose architectures use only scheduling approaches. Luh and Lin divided the entire task into subtasks in [Luh82] and rearranged them to distribute the computational tasks efficiently on multiple PEs. They ignored the data transfer time since this consumes less time then their simulated time of 50 µs per multiplication. They assumed a MIMD architecture with general-purpose PEs and they assigned one PE to each joint exploiting parallelism at the joint level. Although they mainly consider dynamic computations, their work is adaptable for Jacobian computations since the main issue of their work is the design of a near-optimal scheduling algorithm. This work is continued by Kasahara and Narita in [Kasahara85] who use common 8086 and 8087 processors and a scalable MIMD architecture. They also consider the dynamics problem, but their results in scheduling and load balancing are also applicable for the forward Jacobian. They employ two scheduling algorithms: DF/IHS, as discussed in Section 3.1, and Critical Path / Most Immediate Successors First (CP/MISF). They recommend DF/IHS since with CP/MISF the optimality and accuracy of the result can not be guaranteed. Another approach based on general-purpose architectures was investigated by Kokusho et al. in [Kokusho95]. In a multi-transputer network, they use a double-layered load distribution with a DF/IHS for the logical layer and a Distributed Problem Solver for Task Scheduling (DPSTS) for the physical layer. This is because the DF/IHS cannot consider time-delay due to inter-processor communication and the cost for message routings. The DPSTS modifies and optimizes the schedule received from DF/IHS according to the architectural features. The DPSTS is implemented as a part of the firmware on each PE. Finally, an article

handling the parallel processing on general MIMD architectures is [Nabhan95] by Nabhan and Zomaya. They exploit the parallelism on the level of the computation chain using a graph-based approach that considers the communication overhead, congestion possibilities, and the network topology. Therefore, it is independent of the concrete hardware implementation. They achieved computation times for the forward Jacobian in the range of 50 μs and a speed-up of 1.5 with 4 PEs for a PUMA560. Thus, they showed that it is not necessary to develop an expensive special-purpose architecture or even special PEs to solve the forward Jacobian. These approaches with general-purpose architectures can be scalable and a change in the algorithm does not cause any change in the hardware.

As mentioned in Section 3.1, Hamisi and Fraser described the implementation on a network of transputers in [Hamisi89]. This article covers another approach: They use a special-purpose network topology and general-purpose PEs. Therefore, they can use cheap components to adapt their hardware to the problem of robot control and exploit the potential of parallelism in the algorithm. The main difference from the previously discussed approaches is to adapt the specialization of the network topology to the problems of robot control. In [Hamisi89], the control itself is done by computing the actual position of the end-effector with forward kinematics, and at the same time computing the forward Jacobian. From that the reverse Jacobian is calculated to compute the necessary joint rates from the difference of the actual and desired positions of the end-effector in order to correct the position. The level of parallelism is the joint level, since it assigns a PE to each column of the Jacobian matrix. The columns of the Jacobian describe the corresponding joint rates. There is also parallelism at the task level since Hamisi and Fraser compute forward kinematics and the Jacobian in parallel. They achieve a computation time of 6.1 ms and a speed-up of 2.9 for a six DOF manipulator with 6 PEs. A second approach uses common PEs with no special assumptions and a specialized hardware and software architecture [Lin91]. Contrary to [Hamisi89], Lin and Lee do not use a MIMD but a SIMD topology. They assume that advanced robot control schemes require all four kinematics and Jacobian problems and additionally the dynamics problem. So they designed a machine providing the flexibility to solve all problems while maintaining high efficiency. To achieve this, they study these features of the robot control problems and describe a parallel architecture that matches the common features. They exploit parallelism at the joint-level using identical PEs for each joint and two reconfigurable networks. They show the implementation of dynamics computations, but as the authors point out, it is also suited for kinematics and Jacobian computation. In [Zomaya92], Zomaya presents different architectures to implement parallel Jacobian computation. In all architectures, the parallelism at the joint level is exploited using T414 and T800 transputers. Various network schemes to minimize the data transfer between the PEs are investigated. As a result, the minimization of data transfer between the PEs is more important than a uniform load of each PE. With an architecture with 7 PEs for a six DOF manipulator, a computation time of 0.38 μs and a speed-up of 5.5 is achieved. The

article by Gosselin [Gosselin96] presents no processing paradigm, but since the algorithms for reverse kinematics and forward Jacobian are presented with each PE doing the same, we suggest the SIMD topology. Hence, this reference with a speed-up of *n* is classified as one with a special-purpose network topology and general-purpose PEs.

One of the first articles using systolic networks to compute the forward Jacobian is [Orin87] by Orin, Olson and Chao. Orin and Schrader have published a serial algorithm to compute the Jacobian in [Orin84]. The difficulties in implementing in real time led them to a parallel approach. They discuss a serial, a pipeline and a parallel algorithm for the computation of the Jacobian. They conclude that it is possible to reduce the computation time from O(*n*) to O(log *n*), and for n = 7, to achieve a speed-up of 2.35 with 8 PEs with a parallel algorithm exploiting the joint level parallelism. The pipeline algorithm can reduce the computation time and therefore improve data throughput, but the latency time is the same as in the serial case. They introduce additional redundant computations in the parallel algorithm to achieve local communication. They use the hypercube topology, since the single matrix operations can be simply mapped on it. Furthermore, it is possible to propagate a result in log *n* steps to *n* PEs, so that the desired O(log *n*) time could be achieved. After the algorithm and the architecture were designed, they designed a robotics processor on a VLSI chip with four ports and four internal buses based on a Harvard architecture. In [Yeung88] and [Yeung89], Yeung and Lee compared the Jacobian computation on the uniprocessor, SIMD, and systolic architectures. They offer two possibilities for the systolic case: a linear pipeline with 18 PEs and a computation time of O(*n*) or a parallel pipeline with O(*n* log *n*) PEs and a computation time of 3 cycles. The systolic architectures are completely implemented in VLSI. The main point of these four articles on systolic architectures and their implementation in VLSI is the fact that the Jacobian computation is not only faster because of faster computation times of primitive operations and communication times in a systolic array, but also because the algorithms implementable in VLSI are in O(log *n*). In [Sadayappan89] and in the later publication [Orin91], Sadayappan, Ling, Olson and Orin introduced the idea of parallelism on different levels. They designed an architecture on two levels. They developed a Robotics Vector Processor (RVP), which contains three multipliers and three adders controlled as a SIMD to exploit parallelism on the operation level, since the main task was to multiply matrices and vectors. The RVPs themselves are controlled as MIMD, so that their task can be presented in the usual task graph for applying schedule optimization. Hence, the level exploited here is the computation chain level. Two further references, [Walker93] and [Hemkumar94], consider the implementation of CORDIC arithmetic for Jacobian computation. The CORDIC arithmetic is implemented in VLSI. In [Walker93], the architecture is systolic, since that is the most suitable for CORDIC processors. Hemkumar and Cavallaro consider the difficulty of computing the forward and reverse Jacobian for redundant manipulators. See Section 4.2. They also use

CORDIC PEs to combine them in a systolic array to form a systolic machine, which is then implemented in VLSI.

Comparing the different approaches, it seems that there are advantages and disadvantages for all of them. But since the general-purpose PEs in special-purpose interconnection networks achieve a computation time of 0.38 μs in [Zomaya92], making real-time control possible, it is questionable whether the further speed-up of VLSI implementation justifies its inflexibility. On the other hand, the exploitation of the operation level is only valuable in VLSI implementation, since the communication overhead would otherwise be too big.

4.2 Reverse Jacobian

The problem of the reverse Jacobian is to compute the joint velocities from the desired end-effector velocity. There are more computations to be made than for the former three problems. The computations are matrix multiplication, vector addition, matrix-vector multiplication and the computation of the inverse or pseudo-inverse of the Jacobian.

For non-redundant manipulators, the computation of the reverse Jacobian is the inversion of the Jacobian matrix computed for the forward Jacobian. The articles [Hamisi89, Zomaya92, Fujioka93, Kokusho95] consider this case. In the redundant case, where the inverse matrix does not exist, a pseudo-inverse must be computed. This case is treated in [Chang89, Walker93, Hemkumar94, Maciejewski94]. There are two equations to compute the joint velocities using the pseudo-inverse: one with a projection of an arbitrary vector on the null space of the Jacobian, the other using some secondary criterion to fix the arbitrary vector. The first equation is the faster one in most references, since there are less operations to do.

The first article considering the singularities in robot work space is [Chang89]. Chang and Lee concentrate on the problem of numerical instabilities in the computation in the pseudo-inverse of the Jacobian. They use a residue arithmetic to avoid floating point errors. Their parallelization is hence on three levels: they use a data-driven processor array for exploiting the operation level parallelism, a linear array as a pipeline to exploit the joint level parallelism, and separate processing units for any modulus in the residue arithmetic as a very fine grained level under the operation level. The number of moduli is in the range of two to ten, depending on the desired numerical precision. Especially exploiting parallelism in the residue arithmetic makes the parallel design valuable. The theoretical speed-up is $4 \cdot m \cdot n$, where n is the joint number and m the number of moduli. For a six DOF manipulator the speed-up is about 120. They achieve a latency time of 7 μs for a 12-DOF manipulator.

The architecture proposed by Walker and Cavallaro [Walker93] is also capable of computing the pseudo-inverse Jacobian since their CORDIC array is able to compute the singular value decomposition (SVD). With this, the pseudo-inverse can be easily computed. As

mentioned in Section 3, they exploit parallelism at the task level, and, similar to [Chang89], at the operation level by computing the rows of the pseudo-inverse Jacobian.

A further approach for redundant manipulators is discussed by Hemkumar and Cavallaro in [Hemkumar94]. They use an array of CORDIC PEs to exploit parallelism on the operation level. Different implementations of CORDIC PEs are investigated to speed-up the SVD computation and matrix multiplication. The use of CORDIC leads to an operation level parallelism. In contrast to [Walker93], they do not use an array of CORDIC PEs but a special-purpose topology. The latest work investigating parallel computations of reverse Jacobian for redundant manipulators is [Maciejewski94]. Maciejewski and Reagin achieve a speed-up of 3.61 with four digital signal processors (DSP) for a seven DOF manipulator (CESAR). As opposed to other approaches, they exploit parallelism at the joint level, since they calculate two columns of the SVD in each PE. Each of four PEs communicate with two other PEs since the PEs are lined up in a linear array.

An approach not considering singularities in the manipulator work space is presented in [Hamisi89]. They compute the reverse Jacobian as an inversion of the forward Jacobian using Gaussian elimination. Their transputer cluster exploits joint level parallelism since each of the six PEs calculates a column needed for a six DOF manipulator. They use the MIMD as the processing paradigm, though they do not use a scheduling algorithm. Since they use a special purpose interconnection topology, their approach is not scalable without complete redesign. Zomaya uses a Transputer network like Hamisi and Fraser in [Zomaya92]. He discusses the methods for parallel inversion. As a result, he concludes that the Gaussian elimination method is faster then the Gauss-Jordan method. Although the speed-ups with Gauss-Jordan are higher. The computation time for a six DOF is 0.64 μs and the speed-up is 2.2 with 3 PEs.

An interesting approach is [Fujioka93]. Fujioka and Kameyama design a PE using VLSI techniques. They combine two adders and two multipliers on each PE to exploit operation level parallelism. Since each PE has a switching unit for its input and output, they can be configured in different topologies for different problems. E.g., for vector addition, they can be configured as a linear array, and for matrix multiplication they can be configured as an two-dimensional array. Since the PEs are reconfigurable, the architecture is scalable. Fujioka and Kameyama achieve a latency time of less than 32 μs for a twelve DOF redundant manipulator. As the number of PEs increases with increasing number of controlled joints, they also exploit the parallelism of joint level.

Finally, Kokusho et al. discuss the use of scheduling algorithms and load-distribution mechanisms for MIMD architectures for computing robot control in [Kokusho95]. Since they use scheduling algorithms, it is obvious that they exploit parallelism at the computation chain level. They achieve speed-ups up to 3 with 25 PEs for the Stanford arm. They assume a square array as the interconnection topology.

As presented, there are approaches using all levels of parallelism. The most promising approaches seem to be the ones using parallelism on the joint and operation levels, whereas the computation chain level suffers from low efficiency. As shown in [Zomaya92], for non-redundant manipulators it is neither necessary to implement the algorithm in VLSI nor to use special-purpose PEs like CORDIC. Even the most complex problem of robot position control, the reverse Jacobian, can be computed in real-time using appropriate parallel architectures with general-purpose PEs. For redundant manipulators, the additional computation work requires a parallelization on many levels like in [Chang89], but this case seems to be solvable in real-time also.

5 Summary

We have shown that the parallel processing approaches of robot kinematics and Jacobian render short computation times. Hence, the architectures and algorithms developed for parallel robot control allow real-time applications, since common latency times are less than 50 µs for all four problems of robot control. This is far below the cycle rates of a manipulator (a few ms). There is still a great demand for faster computation, since with increasing number of manipulator joints, e.g., for a replica of the human hand or for mechanically stiff, parallel manipulators, only parallel approaches can calculate this in about the same time as for six DOF manipulators [Gosselin96]. The other main reason for perpetual demand on more computational power is the wish to reduce positioning errors through further reduction of the sample rate [Zhang91, Khosla87]. Other incentives for reducing computation time are the skills of advanced robots. Tactile and vision sensors make it possible to avoid moving obstacles, but to do this, the robot kinematics has to be fast enough to evade or stop effectively [Khosla87].

The individual articles reflect the variety of possible approaches. Our list of keywords in Table I serves as an overview of these approaches. The conclusion as to which architecture or parallelization is best cannot be made. There are too many advantages and disadvantages for each approach. There is no such thing as a universal parallel robot control architecture, though some authors claim this. There is still a wide field of research to be done for developing small, cheap, flexible, fast robot control systems with parallel processing.

Acknowledgements

The work was performed at the Institute for Real-Time Computer Systems and Robotics (IPR) under Prof. Dr.-Ing. U. Rembold, Prof. Dr.-Ing. H. Wörn, and Prof. Dr.-Ing. R.

Dillmann at University of Karlsruhe. We wish to thank Albert Y. Zomaya for his interest in our work. Many thanks to Stefanie Grupp for proof-reading.

References

[Ahmad87] S. Ahmad & B. Li: Optimal design of multiple arithmetic processor-based robot controllers. Proc. 1987 IEEE Int. Conf. on Robotics and Automation, pp. 660-663, 1987.
Keywords: forward and reverse kinematics, computation chain level, one dimensional array, MIMD, static load balancing, scalable.

[Ahmad89] S. Ahmad & B. Li: Robot control computation in microprocessor systems with multiple arithmetic processors using a modified DF/IHS scheduling algorithm. IEEE Trans. on Systems, Mman, and Cybernetics, Vol. 19, No. 5, September/October, pp. 1167-1178, 1989.
Keywords: forward and reverse kinematics, computation chain level, one dimensional array, MIMD, static load balancing, scalable.

[Barhen87] J. Barhen: Hypercube ensembles: an architecture for intelligent robots. Computer architectures for robotics and automation.(Eds. J. H. Graham), Gordon and Breach science publishers. New York, pp.195-236, 1987.
Keywords: forward kinematics, dynamics, computation chain level, hypercube, MIMD, dynamic load balancing, static load balancing, scalable.

[Capriot91] F. D. Capriot & S. S. Khanuja & M. M. Stanisic & O. Duta: A singularity free six degree of freedom manipulator. in Advances in robot kinematics by S. Stifter & Jadran Lenarcic, Springer-Verlag/Wien, pp. 128-135, 1991.
Keywords: General.

[Chang89] P. R. Chang & C. S. G. Lee: Residue arithmetic VLSI array architecture for manipulator pseudo-inverse Jacobian computation. IEEE Trans. on Robotics and Automation, Vol. 5, No. 5, October, pp. 569-582, 1989.
Keywords: reverse Jacobian, operation and joint level, array network, linear pipe, systolic array, SIMD, VLSI, integer arithmetic, scalable.

[Chen86] J. B. Chen & R. S. Fearing & B. S. Armstrong & J. W. Burdick: NYMPH: A multiprocessor for manipulation applications. Proc. 1986 IEEE Int. Conf. on Robotics and Automation, pp. 1731-1736, 1986.
Keywords: forward kinematics, task level, bus network, MIMD, scalable.

[Fijany92] A. Fijany & A. Bejczy: Parallel computation systems for robotics algorithms and architectures. New York World Scientific Publishing Co. Pte. Ltd., 1992.
Keywords: forward kinematics and forward Jacobian, operation and joint level, SIMD, MIMD, non-scalable.

[Fujioka93] Y. Fujioka & M. Kameyama: 2400-MFLOPS reconfigurable parallel VLSI processor for robot control. Proc. 1993 Int. Conf. on Robotics and Automation, pp.149-154, 1993.
Keywords: forward kinematics and reverse Jacobian, operation and joint level, one or two dimensional array, SIMD, VLSI, scalable.

[Funda90] J. Funda & R. P. Paul: A computational analysis of screw transformations in robotics. IEEE Trans. on Robotics and Automation, Vol. 6, No. 3, June, pp. 348-356, 1990..
Keywords: forward kinematics, operation level, line-oriented.

[Gosselin96] C. M. Gosselin: Parallel computational algorithms for the kinematics and dynamics of planar and spatial parallel manipulators. Trans. of the ASME: Journal of dynamic Systems, Measurement, and Control , Vol. 118, No. 1, March, pp. 22-28, 1996.
Keywords: reverse kinematics, forward Jacobian, joint level, SIMD, non-scalable.

[Graham89] J. H. Graham: Special computer architectures for robotics, tutorial and survey. IEEE Trans. on Robotics and Automation, Vol. 5, No. 5, October, pp. 543-554, 1989.
Keywords: Overview.

[Hamisi89] F. Hamisi & D. A Fraser: Transputer-based implementation of real-time robot position control. Microprocessors and microsystems, Vol. 13, No. 5, December, pp. 644-652, 1989.
Keywords: forward kinematics, forward and reverse Jacobian, operation, task and joint level, special purpose, MIMD, non-scalable.

[Harber88] R. G. Harber & J. Li & X. Hu & S. C. Bass: The application of bit-serial CORDIC units to the design of inverse kinematics processors. Proc. 1988 IEEE Conf. on Robotics and Automation, pp. 1152-1157, 1988.
Keywords: reverse kinematics, computation chain level, CORDIC, VLSI, non-scalable.

[Hemkumar94] N. D. Hemkumar & J. R. Cavallaro: Redundant and on-line CORDIC for unitary transformations. IEEE Trans. on computers, Vol. 43, No. 8, August, pp. 941-954, 1994..
Keywords: forward and reverse Jacobian, operation level, systolic array, VLSI, CORDIC, non-scalable.

[Kasahara85] H. Kasahara & S. Narrita: Parallel processing of robot-arm control computation on a multimicroprocessor system. IEEE Journal of Robotics and Automation, Vol. 1, No. 2, pp. 104-113, 1985.
Keywords: forward kinematics, MIMD, static load balancing, scalable.

[Khosla87] P. K. Khosla: Choosing sampling rates for robot control. Proc. 1987 IEEE Int. Conf. on Robotics and Automation, Vol. 1, pp. 169-174, 1987.
Keywords: General.

[Kokusho95] Y. Kokusho: Parallel computation for manipulator control using the double-layered load-distribution mechanism on multi-transputer networks. Intelligent Autonomy Systems, IOS Press, 1995.
Keywords: forward and reverse Jacobian, computation chain level, MIMD, scheduling, scalable.

[Lee87] C. S. G. Lee & P. R. Chang: A maximum pipelined CORDIC architecture for inverse kinematic position computation. IEEE Journal of Robotics and Automation, Vol. 3, No. 5, October, pp. 445-458, 1987.
Keywords: reverse kinematics, computation chain level, pipeline, CORDIC, static load balancing, non-scalable.

[Lee89] C. S. G. Lee: Introduction, Robot manipulators: algorithms and architectures. IEEE Trans. on Robotics and Automation, Vol. 5, No. 5, October, pp. 541-542, 1989.
Keywords: Overview.

[Lee91] C. S. G. Lee: On the parallel algorithms for robotics computation. Sensor-based Robots: Algorithms and Architectures. (Eds. C. S. G. Lee) Springer-Verlag, New York, pp. 239-280, 1991.
Keywords: Overview.

[Leung87] S. S. Leung & M. A. Shanblatt: Real-time DKS on a single chip. IEEE Journal of Robotics and Automation, Vol. 3, No. 4, August, pp. 281-290, 1987.
Keywords: forward kinematics, operation level, VLSI, non-scalable.

[Lin91] C.-T. Lin & C. S. G. Lee: Fault-tolerant reconfigurable architecture for robot kinematics and dynamics computations. IEEE Trans. on Systems, Man, and Cybernetics, Vol.21, No.5, September/October , 1991.
Keywords: kinematics and Jacobian, joint level, generalized cube network, SIMD, fault-tolerant.

[Luh82] J. Y. S. Luh & C. S. Lin: Scheduling of parallel computation for a computer-controlled mechanical manipulator. IEEE Trans. on Systems, Man, and Cybernetics, Vol. 12, No. 2, March/April, pp. 214-234, 1982.
Keywords: forward Jacobian, joint level, MIMD, scheduling.

[Maciejewksi94] A. A. Maciejewski & J. M. Reagin: A parallel algorithm and architecture for the control of kinematically redundant manipulators. IEEE Trans. on Robotics and Automation, Vol. 10, No. 4, August, pp. 405-414, 1994.
Keywords: reverse Jacobian, joint level, linear array, MIMD, non-scalable.

[Nabhan95] T. M. Nabhan & A. Y. Zomaya: Application of parallel processing to robotic computational tasks. The Int. Journal of Robotics Research, Vol. 14, No. 1, February, pp.76-86, 1995.
Keywords: forward Jacobian, computation chain level, MIMD, scalable.

[Orin84] D. E. Orin & W. W. Schrader: Efficient computation of the Jacobian for robot manipulators. Int. Journal of Robotics Research, Vol. 3, No. 4, Winter, pp. 66-75, 1984.
Keywords: forward Jacobian.

[Orin87] D. E. Orin & K. W. Olson & H. H. Chao: Systolic architectures for computation of the Jacobian for robot manipulators. in Computer Architectures for Robotics and Automation (Eds. James H. Graham), Gordon and Breach Science Publishers, pp. 39-67, 1987.
Keywords: forward Jacobian, joint level, hypercube, systolic, scalable.

[Orin91] D. E. Orin & P. Sadayappan, Y. L. C. Ling & K. W. Olson: Robotics Vector Processor architecture for real-time control. Sensor-based Robots: Algorithms and Architectures. (Eds. C. S. G. Lee) Springer-Verlag, New York, pp. 203-238, 1991.
Keywords: forward Jacobian and forward kinematics, operation and computation chain level, MIMD,SIMD, VLSI.

[Pedicone87] J. T. Pedicone & T. L. Johnson: Robot motion control with CPAC. in Computer Architectures for Robotics and Automation (Eds. J. Graham) New York, Gordon and Breach Science Publishers, March, pp. 214-234, 1987.
Keywords: forward kinematics, joint level, MIMD, non-scalable.

[Sadayappan89] P. Sadayappan & Y.-L. C. Ling & K. W. Olson & D. E. Orin: A restructable VLSI robotics vector processor architecture for real-time control. IEEE Trans. on Robotics and Automation, Vol. 5, No. 5, October , pp. 583-598, 1989.
Keywords: forward kinematics and forward Jacobian, operation and computation chain level, MIMD, SIMD, VLSI.

[Tsai85] L. W. Tsai & A. P. Morgan: Solving the kinematics of the most general six- and five-DOF manipulators by continuation methods. Trans. ASME, J. Mechanism, Transmissions, Automation in Design, 107, June, pp. 189-200, 1985.
Keywords: reverse kinematics, iterative form.

[Walker93] I. D. Walker: Parallel VLSI architectures for real-time kinematics of redundant robots. IEEE Int. Conf. on Robotics and Automation, pp. 870-877, 1993.
Keywords: forward kinematics, reverse kinematics forward and reverse Jacobian, task and operation level, systolic array, VLSI, CORDIC.

[Wang87] Y. Wang & S. E. Butner: A new architecture for robot control. Proc. 1987 IEEE Conf. on Robotics and Automation, , pp. 664- 670, 1987.
Keywords: reverse kinematics, task and operation level, MIMD, CORDIC, non-scalable.

[Yeung88] T. B. Yeung & C. S. G. Lee: Efficient parallel algorithms and VLSI architectures for manipulator Jacobian computation. Proc. 1988 IEEE Conf. on Robotics and Automation, pp. 1158-1163, 1988.
Keywords: forward Jacobian, operation level, SIMD, systolic, VLSI, scalable.

[Yeung89] T. B. Yeung & C. S. G. Lee: Efficient parallel algorithms and VLSI architectures for manipulator Jacobian computation. IEEE Trans. on Systems, Man, and Cybernetics, Vol. 19, No. 5, September/October, pp. 1154-1165, 1989.
Keywords: forward Jacobian, operation level, SIMD, systolic, VLSI, scalable.

[Zhang90] H. Zhang & R. P. Paul: A parallel inverse kinematics solution for robot manipulators based on multiprocessing and linear extrapolation. IEEE Trans. on Robotics and Automation, pp. 468-474, 1990.
Keywords: reverse kinematics, joint level, MIMD, non-scalable.

[Zhang91] H. Zhang & R. P. Paul: A parallel inverse kinematics solution for robot manipulators based on multiprocessing and linear extrapolation. IEEE Trans. on Robotics and Automation, Vol. 7, No. 5, October, pp. 660-669, 1991.
Keywords: reverse kinematics, joint level, MIMD, non-scalable.

[Zomaya92] A. Y. Zomaya: Modelling and simulation of robot manipulators -A parallel processing approach. World Scientific Publishing Co. Pte. Ltd., 1992.
Keywords: forward Jacobian, reverse Jacobian, joint and computation chain level, tree, MIMD, non-scalable.

[Zomaya96] A. Y. Zomaya: Parallel and distributed computing handbook. McGraw-Hill, pp. 1149-1159, 1996.
Keywords: Overview.