$O(\log n)$ Algorithm for Forward Kinematics under Asynchronous Sensory Input

Ryo Wakatabe, Yasuo Kuniyoshi and Gordon Cheng

***Abstract*— This paper presents a new algorithm for forward kinematics, called Asynchronous Forward Kinematics (AFK). The algorithm has the complexity of $O(\log n)$ for updating one joint angle, and $O(\log n)$ for obtaining a homogeneous transformation matrix between links. AFK enables computation for efficient forward kinematics under asynchronous sensory data. Moreover, AFK peovides localise computational resources at sensitive joints to the position of the endpoint (e.g. a fingertip), like a root joint. We provide comparative results including computation time, evaluating AFK against the conventional forward kinematics (CFK). The results showed that the computation time is well adequate for real-time computation. Computation time for 100 links takes less than 20 us for 1 query. Moreover, computation time with over 50000 links takes less than 35 us for 1 query.**

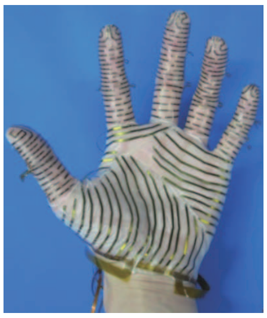
I. Introduction

Numerous domains in robotics require robots with high number of degrees of freedom, most of these robots are of practical use. Here we highlight two examples, the first example is a hyper-redundant robot such as the Buckingham and Graham's snake-arm with 20 degrees of freedom, which is being used to conduct inspection and repair operations within nuclear power plants [3]. Second example is robots covered with highly dense tactile sensors (e.g. [8], [10]). When computing kinematics and dynamics of robots with tactile sensor, degrees of freedom can be enormous due to the kinematics and dynamics algorithms require coordinate frames on every tactile sensor to estimate forces and torques of each tactile sensor [9].

Furthermore, synchronising sensory data of robots with high degrees of freedom at once induces a large overhead. Even when one joint angle or one tactile sensor is updated, conventional algorithms must recalculate all kinematics and dynamics. Moreover, robots with high degrees of freedom tend to have a large amount of data communication between sensors and robot's interface. Conventional algorithms require to wait for all sensory data to be updated prior to subsequent computation, which causes idleness of processors and loss of real-time computation.

Algorithms associated with robots with high number of degrees of freedom are usually demanding, thus, in order to solve these computational problems asynchronous kinematics and dynamics estimation are preferable. In other words, kinematics and dynamics which do not need synchronisation of sensory data are desirable for more efficient computation. Here, we propose a new algorithm for forward kinematics, Asynchronous Forward Kinematics (AFK) – with the aims to support asynchronous dynamics computation and AFK for branched chains.

Ryo Wakatabe and Gordon Cheng are with the Institute for Cognitive Systems, The Technical University of Munich, Karlstrasse 45/II, 80333 Mnche, Germany. {ryo.wakatabe, gordon}@tum.de. Ryo Wakatabe and Yasuo Kuniyoshi are with the Department of Mechano-Informatics, The University of Tokyo, 7-3-1 Hongo, Bunkyo, Tokyo. {wakatabe, kuniyosh}@isi.imi.i.u-tokyo.ac.jp


(a)

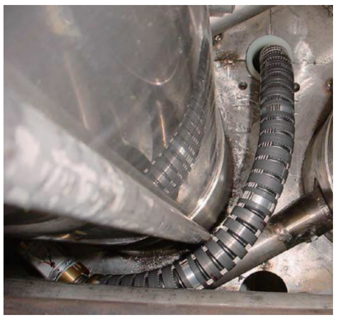

(b)

Fig. 1. Appliations with high degree of freedom. (a) High-density tactile sensing glove with 1000 sensors [13]. (b) Ringhals, Snake-arm robots with 20 DoF [3].

A. Related work

Research in the efficient parallel computation of kinematics and dynamics for synchronous sensory data is well matured. These algorithms include $O(n)$ for serial computation [1] and $O(\log n)$ for parallel computation [14]. Yamane and Nakamura have developed an efficient parallel kinematics and dynamics computation of human figures, which achieves 2.5ms for 48 DOF human figures [14]. However, these algorithms all require parallel computation and cannot support asynchronous input, thus inducing a more complex computational architecture with larger computation overhead.

A hyper-redundant robot is with a finite degree-of-freedom, and a continuum robot is with infinite degrees of freedom. A forward kinematics algorithm especially for hyper-redundant robots and continuum robots has been developed. Rucker *et al.* developed a forward kinematics algorithm for continuum robots using a rod theory [12], which can compute the position of fingertip when a distributed force is applied on a continuum rod. Chirikjian developed a forward and inverse kinematics algorithm for hyper-redundant robots by using a continuum approach [4]. However, these algorithms cannot provide an exact solution, as continuum approaches can only provide approximated solutions.

B. Contribution

The contributions of this work are as follows.

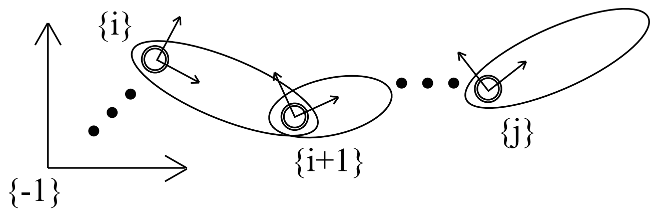


Fig. 2. Coordinates of serial links.

- The proposed algorithm is more efficient than CFK under asynchronous sensory input. CFK requires $O(n)$ time complexity to update even only one joint angle. Whereas, AFK can update the query only with $O(\log n)$ cost. This is preferable for hardware systems, less processors idleness and real-time computation of hyper-redundant robots.
- AFK allow the scheduling of computational resources. AFK enables us to localise computational resources to the sensitive joints for a task. For example, the position and orientation of an end effector is usually important for the task. The joint angle of the root joint is more sensitive than the others because its moment arm is large. This algorithm can easily schedule computational resources by increasing update frequency of the root angle.
- This algorithm can efficiently compute more continuous forward kinematics than CFK. CFK is less continuous because it needs to wait for synchronisation of sensory data by timer interruption. Whereas, AFK can compute an exact forward kinematics at any time by exploiting the previously computed result.

C. Organization

This paper is organised as follows. First, a review of conventional forward kinematics and the algorithm of AFK is presented. This is followed by examples of the functioning of AFK. Then, the experiment to compare computation time between CFK and AFK discussed. And finally, a summary of the comparison of CFK and AFK is presented.

II. Methodology

In this section, we present outlines of CFK and AFK. Then, we compare their time complexity. Note, in this paper we assume a serial link with no branched structure exists[1].

A. Conventional Forward Kinematics (CFK)

CFK has been widely used to solve forward kinematics problems [5]. Here, let us recap the basic concept of CFK.

Let $\{X\}$ denote coordinate frame X, ${}^X_Y R$ denote a 3×3 rotation matrix from $\{X\}$ to $\{Y\}$, and ${}^X P_Y$ denote a translation vector, which indicates the origin position of $\{Y\}$ from $\{X\}$. A 4×4 homogeneous transformation matrix ${}^X_Y H$ can be defined by

$$ {}^X_Y H = \begin{pmatrix} {}^X_Y R & {}^X P_Y \\ \mathbf{0}^\mathbf{T} & 1 \end{pmatrix} \tag{1} $$

[1]AFK for branched chains will be addressed in a follow up paper.

Algorithm 1 CFK

```
isPending = false
function UPDATE($i$)                ▷ Change joint angle of {i}
    $\theta_i \leftarrow new\theta_i$
    isPending = true
end function
function ASK($i, j$)        ▷ Get trans. matrix from {i} to {j}
    if isPending = true then
        for $i$ in $[0,n)$ do
            ${}^{-1}_{i}H \leftarrow {}^{-1}_{i-1}H {}^{i-1}_{i}H$
        end for
        isPending = false
    end if
    return $({}^{-1}_{i}H^{-1}) {}^{-1}_{j}H$
end function
```

where $\mathbf{0}^\mathbf{T}$ is a row vector of 3 zeros. Then, the kinematic chain law can be viewed as the matrix multiplication

$$ {}^X_Y H {}^Y_Z H = {}^X_Z H \tag{2} $$

A homogeneous transformation matrix is always invertible. The inverse matrix is efficiently computed using the fact that an inverse matrix of a rotation matrix is equal to a transposed matrix

$$ {}^X_Y H^{-1} = {}^Y_X H = \begin{pmatrix} {}^X_Y R^T & -{}^X_Y R^T \, {}^X P_Y \\ \mathbf{0}^\mathbf{T} & 1 \end{pmatrix} \tag{3} $$

Let us consider that a serial link with n links. A link i of the robot has coordinate frame $\{i\}$ and joint i. θ_i denotes an angle of joint i. The coordinate frame of the base link is $\{-1\}$.

The homogeneous transformation matrix ${}^{i-1}_{i}H$ is dependent on θ_i. Hence ${}^{i-1}_{i}H$ can be written as

$$ {}^{i-1}_{i}H = {}^{i-1}_{i}H(\theta_i) \tag{4} $$

where ${}^{i-1}_{i}H(\theta_i)$ is easily determined by a structure of a robot using Denavit and Hartenberg (D-H) [6].

The purpose of CFK is to calculate ${}^i_j H$ for each i and j. To solve this problem, CFK compute them in two steps. First, CFK constructs ${}^{-1}_{i}H$ using equation (2) for each j.

$$ {}^{-1}_{i}H = {}^{-1}_{i-1}H {}^{i-1}_{i}H \tag{5} $$

Second, using equation (3), ${}^i_j H$ is computed by

$$ {}^i_j H = ({}^{-1}_{i}H^{-1}) {}^{-1}_{j}H \tag{6} $$

The time complexity of CFK is $O(n)$ for construction of ${}^{-1}_{i}H$ given by Equation (5), and O(1) for calculating ${}^i_j H$ with Equation (6). The space complexity of CFK is $O(n)$ to store ${}^{-1}_{i}H$ for each link i.

In a normal operation, CFK constructs ${}^{-1}_{i}H$ for every control cycles. Consider the CFK Algorithm 1, *UPDATE* and *ASK* must be combined at each cycle, which would make the reconstruction inefficient.

Algorithm 2 AFK for update query

```
function UPDATE($i$)
    $v \leftarrow$ node of ${}^{i-1}_{i}H$
    while $v$ is not root do
        $v \leftarrow$ parent of $v$
        $v \leftarrow$ left child of $v$ multiplied by right child of $v$
    end while
end function
```

B. Asynchronous Forward Kinematics (AFK)

The core idea of AFK is based on separating *UPDATE* (Algorithm 2) and *ASK* (Algorithm 3) queries, and storing previously calculated kinematics. AFK have $2n-1$ homogeneous transformation matrices, which can always derive an exact solution of the forward kinematics. Soon after an update query is given, AFK updates matrices only dependant on the update query. A similar idea was also used for a binary robot [7], although the kinematics of binary manipulators is completely predefined.

In order to realise AFK, *segment trees* are used. A segment tree is a data structure which is originally used in computational geometry to solve Klee's rectangle problems [2]. This algorithm has wide applications because segment trees can make *UPDATE* and *ASK* operations in logarithmical time for any associative operations [11].

C. Segment trees

The root node of a segment tree will hold the information for the interval $[i,j]$ with n leaf nodes. If $i \neq j$, the left and right sons will hold the information for the intervals $[i,(i+j)/2]$ and $[(i+j)/2+1,j]$. The maximum depth of the segment tree is $O(\log n)$. To update one leaf node, the algorithm updates all nodes dependent on the updated nodes. This will be done by updating all nodes in which the path between the root node and the updated node. The number of nodes to be updated cannot exceed $O(\log n)$ because the depth of the tree is bounded. To report the result of ask query, the tree finds the intervals that can cover $[i,j]$ with the minimum number of nodes. The minimum number of nodes cannot exceed $O(\log n)$ because a number of nodes covering the given query is at most 2 for all node in a specific depth. Hence, the overall interval costs is only $O(\log n)$ time complexity.

If you choose any k that $2^k \geq n$, a segment tree with 2^k nodes will be a complete binary tree. That enables us to efficiently find a parent and two sons of a node using only bitwise operations.

The space complexity of a segment tree is $O(n)$. The number of deepest nodes is n, the second deepest nodes is $n/2$ and so on. So the total number of nodes is $n+n/2+n/4+...+1=2n-1$.

Thus, we can determine the worst-case time complexity of AFK to be $O(\log n)$. This means that computation time of AFK is guarantee to be stable. Hence, an essential quality for any real-time systems, like a robot, is achieved.

Algorithm 3 AFK for ask query

```
function ASK($i, j$)
    return REC(root, $i$, $j$)
end function
function REC($v, i, j$)
    if $[i,j]$ includes $v$ then                    ▷ Include
        return $v$
    else if $[i,j]$ does not cross $v$ then         ▷ Unrelated
        return Identity Matrix
    else                                            ▷ Cross
        leftMat = REC(left child of $v$, $i$, $j$)
        rightMat = REC(right child of $v$, $i$, $j$)
        return leftMat multiplied by rightMat
    end if
end function
```

D. Detail of AFK

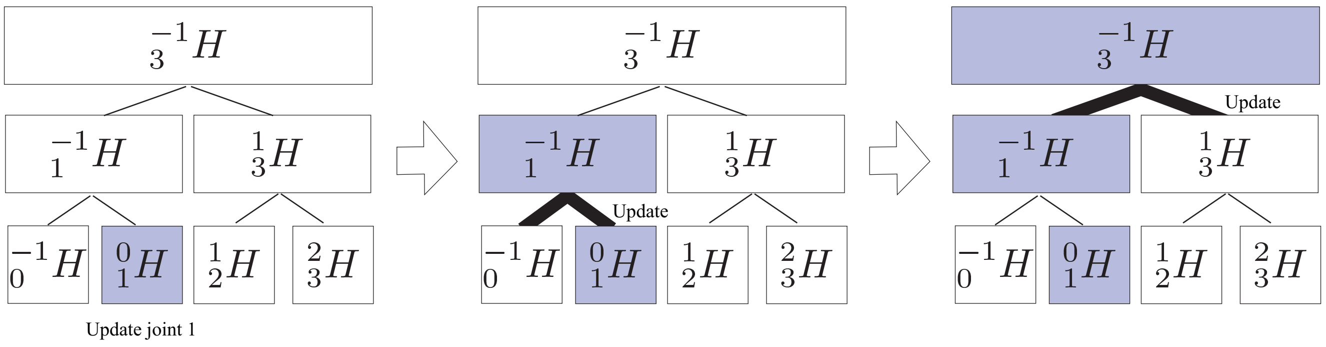
We stored ${}^{i-1}_{i}H$ on leaf nodes of a segment tree. The stored information on nodes is a matrix chain multiplication of homogeneous transformation matrices, which is ${}^{j}_{k}H$ of Equation (2). How to update leaf nodes is completely determined by a robot structure. For each update loop, nodes dependent on the updated leaf node, which can be updated by $O(\log n)$ times simple 4×4 matrix multiplication. Combining intervals of ask queries is also done with $O(\log n)$ time complexity.

Let us consider an example of AFK for a robot with 4 links (Fig. 3). Consider the angle of joint 1 is updated. Then ${}^{0}_{1}H$, ${}^{-1}_{1}H$ and ${}^{-1}_{3}H$ have to be updated. The other nodes do not have to be updated because they are not dependent on θ_1. Therefore, the number of updated nodes is only 3, which is equal to the depth of the segment tree. Also, consider if you wish to ask ${}^{-1}_{2}H$. AFK algorithm searches the set of intervals with the minimum number of nodes from the root node. AFK achieves this by multiplying nodes when the query includes them and splitting search intervals when the query and the search interval crosses. To calculate ${}^{-1}_{2}H$ requires only 1 multiplication of ${}^{-1}_{1}H$ and ${}^{1}_{2}H$.

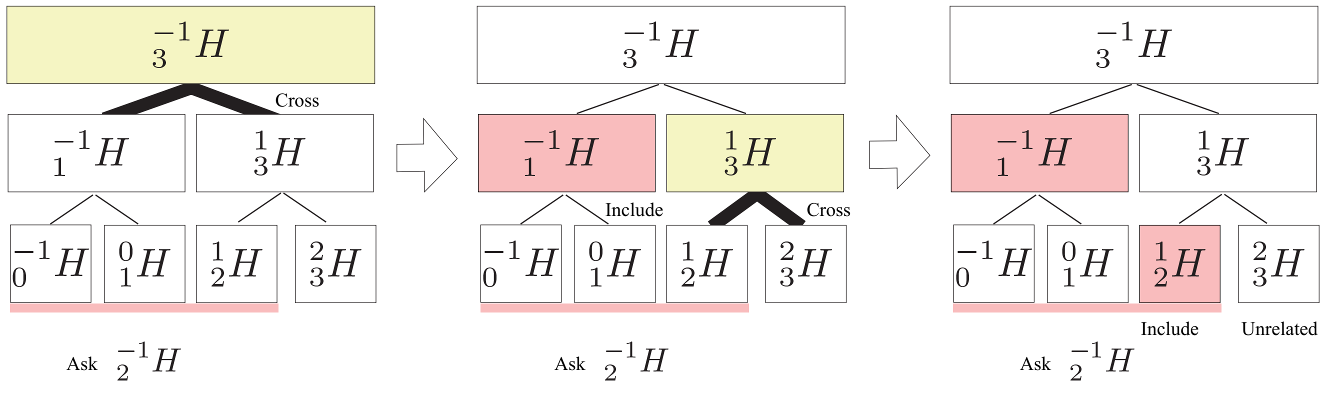
This algorithm can be used when n is not a power of 2. There are two methods. The first method is to construct the segment tree as an unbalanced binary tree. If a node has only one child node, the node has to return the same value as the child node. The second method is to construct a segment tree of a larger complete binary tree. In this case, ${}^{n-1}_{n}H, {}^{n}_{n+1}H$ and so on, which should be an identity matrix due to the fact that there are no corresponding joints. For example, when $n=11$, we can construct a segmented tree with $2^4=16$ leaf nodes. In the 16 homogeneous matrices, ${}^{11}_{12}H, {}^{12}_{13}H, ..., {}^{15}_{16}H$ are always identity matrices.

E. Comparison of Computational Complexity

The time complexity of CFK and AFK is summarised in Table I. CFK has time complexity of $O(n)$ for an update query and $O(1)$ for an ask query. AFK has time complexity with $O(\log n)$ for an update query and $O(\log n)$ for an ask query.



(a) Update query



(b) Ask query

Fig. 3. An example of how AFK algorithm works. (a) UPDATE query. Blue nodes indicate nodes which is already updated. (b) ASK query. Yellow nodes indicate crossing nodes. Red nodes indicate the final nodes to be multiplied.

TABLE I

COMPARISON OF TIME COMPLEXITY FOR UPDATE AND ASK QUERIES.

	update	ask
CFK	$O(n)$	$O(1)$
AFK	$O(\log n)$	$O(\log n)$

TABLE II

COMPARISON OF TOTAL TIME COMPLEXITY OF CFK AND AFK.

	Synchronous	Random	Alternative
CFK	$O(qn)$	$O(qn^2)$	$O(qn^2)$
AFK	$O(qn\log n)$	$O(qn\log n)$	$O(qn\log n)$

The time complexity for query series of CFK and AFK is summarised in Table II. Synchronous query series include n update queries and some ask queries alternatively. Asynchronous query series are defined as query series which is not synchronous query series. Let Q be the number of queries and q be Q/n. For asynchronous query series, AFK is more efficient as $O(qn\log n)$ than CFK's $O(qn^2)$. Whereas, for synchronous query series, AFK is less efficient as $O(qn\log n)$ than CFK's $O(qn)$.

The space complexity of CFK and AFK is $O(n)$. CFK stores n homogeneous transformation matrices $^{-1}_{i}H$. And AFK stores $2n-1$ homogeneous transformation matrices.

III. EXPERIMENTAL SETUP

In this experiment, we computed forward kinematics for each update queries and ask queries using CFK and AFK. We compared these computation time for n serial link robot with 3 types of query series (*Synchronous, Random and Alternative*).

A. Queries

We prepared 2 types of queries: *update* and *ask*.

1) update query: set θ_i

2) ask query: get $^{i}_{j}H$

For our assessment, we measured computation time of 1 update and ask query. Because computation time of 1 query is quite small, we computed 100000 queries then took the average time.

B. Query series

To know how the arrangement of query will affect the computation time, we prepared 3 types of query series: *Synchronous*, *Random* and *Alternative* query series. Each query series have $2nq$ queries.

1) Synchronous: repeat (n update $\rightarrow$ n ask query), q times.

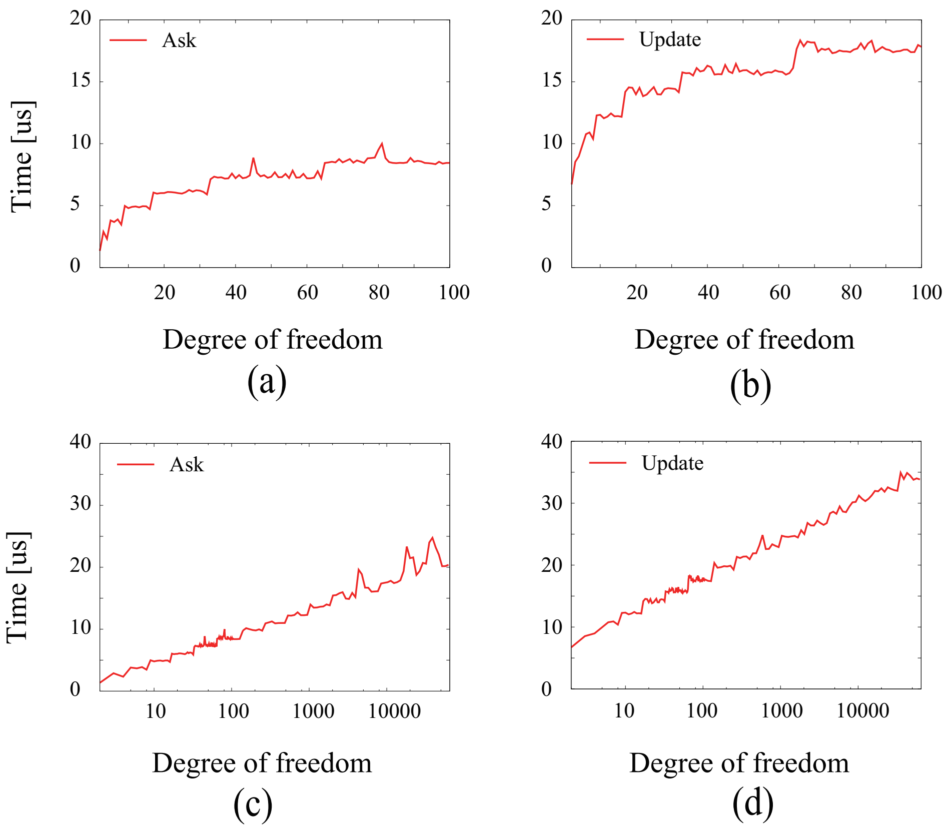


Fig. 4. Computation time of 1 update and ask query of AFK. (a) Computation time of ask query with small number of degree of freedom. (b) Computation time of update query with small number of degree of freedom. (c) Computation time of ask query with large number of degree of freedom, with abscissas denoting DoF of logarithmic scale. (d) Computation time of update query with large number of degree of freedom, with abscissas denoting DoF of logarithmic scale.

2) Random: repeat (randomly selected query), $2nq$ times

3) Alternative: repeat (1 update $\rightarrow$ 1 ask query), nq times

Random and alternative query series simulate asynchronous sensory data. The performance of AFK is expected to be high because CFK requires reconstruction with O(n) time complexity for each update queries. Alternative query series has the worst query series for CFK. Whereas, High performance of CFK is expected because CFK assumes synchronous sensory data.

C. *Detail of query series*

We compared computation time of CFK and AFK for 3 query series. The computation time of CFK and AFK with q and n was compared. q was fixed at 1000. We selected n from 5 to 90 in multiples of 5.

We randomly selected i of update queries, j of ask queries and an angle to update.

D. *Implementation*

The algorithms were implemented with the C++ language. Computation time is measured on an Intel Core i7-4790 CPU @ 3.60GHz. We used the Eigen library for the implementation of matrix manipulations.

IV. Results

A. *Computation time of* 1 *query*

Computation time of $n(n < 100)$ links is fast enough to achieve real-time computation(Fig. 4(a), (b)). For all n up to 100, ask query takes only 10 us and update query only takes 20 us. Computation time of $n(n < 65000)$ links is also examined (Fig. 4(c), (d)). Even when n is very large like $n > 50000$, ask query takes only 25 us and update query only takes 35 us. Thus, the computation time scales well for large range of n.

B. *Computation time of query series*

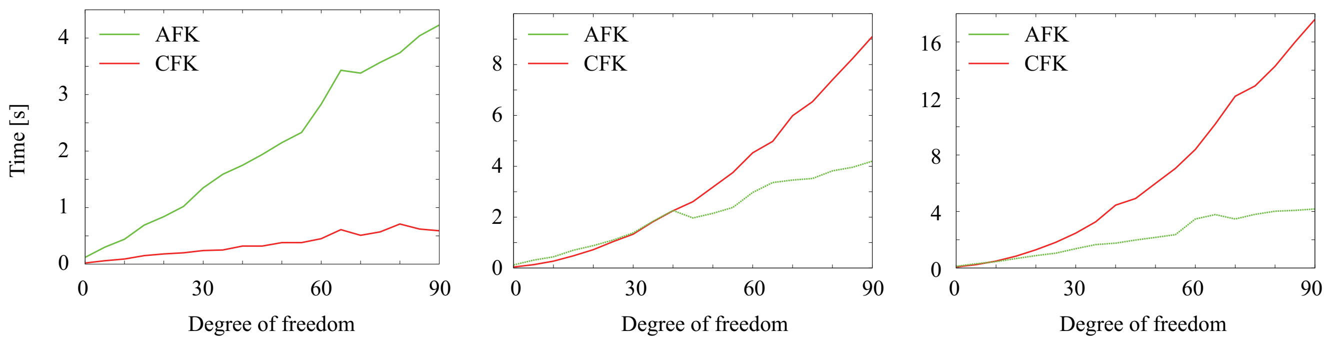
AFK is more efficient than CFK for random query series when $n \leq 50$ and for alternative query series when $n \leq 20$ (Fig. 5(b, c)). Whereas, AFK is less efficient than CFK for synchronous query series for all n (Fig. 5(a)). The theoretical time complexity is confirmed as shown in Table II.

The comparison indicates:

1 AFK is more efficient than CFK for asynchronous sensory data and asking queries, when degrees of freedom is large.

2 AFK is less efficient for synchronous sensory data, like data acquired by time interrupt.

3 AFK is less efficient when degrees of freedom is small.



(a) Synchronous query series (b) Random query series (c) Alternative query series

Fig. 5. Total computation time of CFK and AFK for synchronous, random and alternative query series. The number of queries is 2000n, where n denotes degrees of freedom. Red: conventional forward kinematics. Green: asynchronous forward kinematics.

V. Conclusion and Discussion

In this paper, we presented an algorithm for efficient computation of forward kinematics under asynchronous sensory data, called Asynchronous Forward Kinematics. Our algorithm has $O(\log n)$ time complexity for each updating and asking queries, and $O(n)$ space complexity. Computation time of the algorithm is compared with one of the conventional forward kinematics algorithm.

We conclude this paper by emphasising the following points:

- An efficient algorithm for forward kinematics under asynchronous sensory data was developed and implemented. Its worst-case time complexity is $O(\log n)$ to update a joint angle and ask a homogeneous transformation matrix.
- Simulation results showed that the computation time is effective enough for real-time computation. Computation time with 100 links takes less than 20 us for 1 query. Moreover, computation time with over 50000 links takes less than 35 us for 1 query.
- The algorithm does not require recalculation of all kinematics when only one joint is updated.
- The algorithm provides with a mechanism to enables us to effectively schedule computational resources for sensitive joints to a task.
- This algorithm does not assume a fixed control cycle, which can enable a more reflective and continuous forward kinematics.

In our future work, we will focus on the computation of AFK of branched chains and asynchronous inverse dynamics. In particular, AFK of branched chains can be achieved through: (1) decomposing branched chains as several serial chains; and (2) then by combining all results of AFK of serial chains. Furthermore, we will examine various ways of decomposition of trees to enables us to efficiently compute AFK.

Acknowledgments

This first author is supported by the European Commission under grant agreements PITN-GA-2012-317488-CONTEST. This first author is supported by GCL Program of The Univ. of Tokyo. We'd like to thank Yoshiyuki Ohmura is with the Department of Mechano-Informatics, The University of Tokyo for his valuable help regarding discussing a range of applications of this work.

References

[1] D. Baraff, "Linear-time dynamics using lagrange multipliers," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 1996, pp. 137–146.

[2] J. L. Bentley, "Solutions to klee ' s rectangle problems," Technical report, Carnegie-Mellon Univ., Pittsburgh, PA, Tech. Rep., 1977.

[3] R. Buckingham and A. Graham, "Nuclear snake-arm robots," *Industrial Robot: An International Journal*, vol. 39, no. 1, pp. 6–11, 2012.

[4] G. S. Chirikjian and J. W. Burdick, "A modal approach to hyper-redundant manipulator kinematics," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 3, pp. 343–354, 1994.

[5] J. J. Craig, *Introduction to robotics: mechanics and control*. Pearson Prentice Hall Upper Saddle River, 2005, vol. 3.

[6] J. Denavit, "A kinematic notation for lower-pair mechanisms based on matrices." *Trans. of the ASME. Journal of Applied Mechanics*, vol. 22, pp. 215–221, 1955.

[7] D. S. Lees and G. S. Chirikjian, "An efficient method for computing the forward kinematics of binary manipulators," in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 2. IEEE, 1996, pp. 1012–1017.

[8] P. Mittendorfer, E. Yoshida, and G. Cheng, "Realizing whole-body tactile interactions with a self-organizing, multi-modal artificial skin on a humanoid robot," *Advanced Robotics*, vol. 29, no. 1, pp. 51–67, 2015.

[9] Y. Nakamura and K. Yamane, "Dynamics computation of structure-varying kinematic chains and its application to human figures," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 2, pp. 124–134, 2000.

[10] F. Nori, S. Traversaro, J. Eljaik, F. Romano, A. Del Prete, and D. Pucci, "icub whole-body control through force regulation on rigid non-coplanar contacts," *Frontiers in Robotics and AI*, vol. 2, p. 6, 2015.

[11] F. P. Preparata and M. Shamos, *Computational geometry: an introduction*. Springer Science & Business Media, 2012.

[12] D. C. Rucker, B. A. Jones, and R. J. Webster III, "A geometrically exact model for externally loaded concentric-tube continuum robots," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 769–780, 2010.

[13] T. Sagisaka, Y. Ohmura, A. Nagakubo, K. Ozaki, and Y. Kuniyoshi, *Development and Applications of High-Density Tactile Sensing Glove*. Springer Berlin Heidelberg, 2012.

[14] K. Yamane and Y. Nakamura, "Efficient parallel dynamics computation of human figures," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 1. IEEE, 2002, pp. 530–537.