Proceedings of the 1996 IEEE
International Conference on Robotics and Automation
Minneapolis, Minnesota - April 1996

An Efficient Method for Computing the Forward Kinematics of Binary Manipulators

David S. Lees*
Gregory S. Chirikjian†

Department of Mechanical Engineering, Johns Hopkins University, Baltimore, MD 21218

Abstract

Binary actuators have only two discrete states (denoted '0' and '1'), both of which are stable without feedback. As a result, manipulators built with binary actuators have a finite number of states. Compared to a manipulator built with continuous actuators, a binary manipulator provides good performance, and is also relatively inexpensive. However, the number of states of a binary manipulator grows exponentially with the number of actuators. While this makes the calculation of its inverse kinematics quite difficult, the discrete nature of a binary manipulator makes it possible to compute its forward kinematics more efficiently than for a continuously actuated manipulator. By precomputing all possible configurations of each module of a binary manipulator (a finite and usually small number) it is possible to compute the forward kinematics from a set of joint parameters without using any transcendental functions.

1 Introduction

Presently, nearly all robots are powered by continuous actuators. Continuously actuated robots can be built to be precise and to carry large payloads, but they usually have very high price/performance ratios, as evidenced by the high cost of the industrial robots available today. "Hyper-redundant," discretely actuated robots are a promising alternative to traditional robots for certain applications. A hyper-redundant robot can be built by stacking variable geometry trusses on top of each other in a long serial chain (See Figure 1). This approach yields a structure with good stiffness, dexterity and load-bearing capabilities, compared to traditional non-redundant robots. Using discrete, rather than continuous, actuators to power a truss-based robot increases the reliability and lowers the cost of the system. Discrete actuators (e.g. pneumatic cylinders) are less expensive and simpler than their continuous counterparts. Furthermore, since binary actuators have only two distinct stable states, a robot constructed with them does not need feedback control at each actuator, which eliminates the need for a very costly element of continuously actuated robots.

The characteristics of binary VGT manipulators make them well suited to a number of tasks. They could be used for inspection or repair in constricted spaces, where the flexibility and compactness of the VGT structure is a distinct advantage. They are also candidates for use in human service applications, where good performance is needed, along with low cost. Finally, miniature "snake-like" robots are promising as tools for performing minimally invasive medical procedures. For example, they could be used in a laproscope, or as an element of a catheter. For applications on such a small scale, it is much easier to build discrete actuators (e.g. actuators operated by electrostatic forces [1]) than to build continuous actuators.

While robots with binary actuators may be useful in many applications and are very inexpensive compared to continuously actuated robots, there is one problem related to their control that is much more difficult than the corresponding problem for a continuously actuated robot: The number of possible configurations of a binary robot grows exponentially with the number of actuators. For example, a binary robot with 30 actuators has 2^{30} (approximately 10^9) distinct states, which makes the exhaustive enumeration of all of its states impractical. The large number of possible states of a binary manipulator makes it highly desirable to have efficient algorithms for searching through some (potentially large) subset of manipulator configurations that satisfy a particular constraint, so that an "optimal" configuration can be chosen.

*E-mail: lees@polaris.me.jhu.edu
†E-Mail: greg@polaris.me.jhu.edu

This work was supported by a young investigator award and a presidential faculty fellow award to the second author from the National Science Foundation

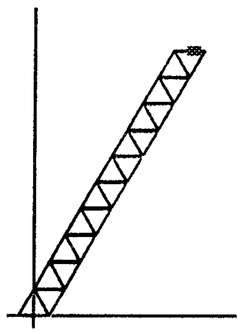
Figure 1: A 10 module binary manipulator built with variable geometry trusses.

This paper discusses an efficient method for computing the forward kinematics of binary manipulators. It takes advantage of the discrete nature of binary actuators to make the calculation of the forward kinematics considerably more efficient than it would be for continuous actuators. Such an algorithm is useful not only for the calculation of forward kinematics, but also as an element of more complex algorithms, such as those for computing the inverse kinematics or finding the workspace of binary robots.

2 Previous Work

Since the high price/performance ratio of most robots makes them impractical for many potential applications, efforts to develop inexpensive, but capable, robots have begun to gain momentum. For example, Canny and Goldberg [2] have proposed a reduced complexity paradigm for robotic manipulation. There have also been several efforts to develop reliable sensorless manipulation [3, 4]. In sensorless manipulation, the geometric constraints of a task are exploited to create a manipulation strategy that is guaranteed to succeed even without feedback, within certain broad limits. For example, Erdmann and Mason [5] have demonstrated an algorithm that can force an "L" shaped bracket into a known orientation by placing it on a tray and moving the tray through a pre-determined sequence of motions.

Binary robots are a natural extension to sensorless manipulation. Sensorless manipulation reduces the need to sense a robot's environment, while binary actuators allow us to build a robot without joint-level sensing of position and velocity. There have been a number of efforts in the past to build robots with binary actuators [6, 7]. However, at the time these projects were undertaken, effective algorithms for controlling hyper-redundant manipulators had not yet been developed, nor were computers sufficiently powerful to control robots with many degrees of freedom, even if they could have used the control algorithms that are currently available.

More recent efforts to develop binary robots include a project to use silicon micro-machining techniques to build small actuators [1]. Also, an effective algorithm has been devised to compute the workspace of hyper-redundant binary robots [8]. Finally, methods have been presented to synthesize a binary manipulator to reach a specific set of points exactly [9], and to make a binary manipulator adhere to a specified curve [10].

In the present work, and other related efforts [11, 12], we are developing a framework in which to plan well behaved motions of manipulators with discrete (binary) actuation. It is our hope that this will lead to very inexpensive and reliable manipulators. In order for the cost benefits of binary actuation to be realized for the subset of tasks for which binary manipulators are appropriate, the computational requirements for the planning must be reduced to a competitive level with continuous actuation. Providing a means for fast computation of the manipulator forward kinematics is one step in this direction.

3 An Efficient Forward Kinematics Algorithm

The forward kinematics algorithm presented here is designed to compute the position and orientation (relative to the base) of the center of a binary manipulator's end-effector, given the states of all the actuators in the manipulator.

3.1 Definition of Terms

To understand the forward kinematics algorithm, it is important to understand the following definitions, which were originally defined in [8]:

Binary manipulator A *binary manipulator* is a manipulator that is composed of a set of modules with two-state actuators, stacked one atop the other (Fig 1). The modules are numbered from $1, \ldots, B$, starting from the base of the manipulator.

Each module has a frame attached to its top (Figure 2). The frames are numbered such that frame i is on top of module i, and frame 0 is the frame at the manipulator base. J_i denotes the number of independent binary actuators in module i. Therefore, there are 2^{J_i} different combinations of binary actuator states (and corresponding configurations) for the i^{th} module.

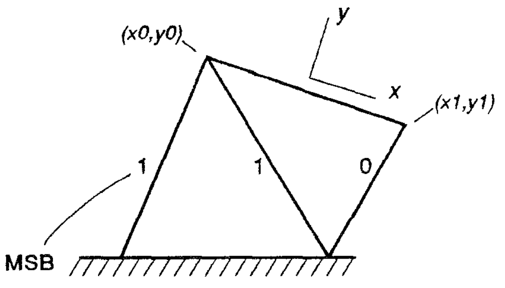

Figure 2: A single binary truss module shown in state 110.

Manipulator state The *state* of a binary manipulator is a binary number, S, whose bits represent the states of each actuator in the manipulator. The state of an individual module in the manipulator is denoted by s_i, which is a J_i bit binary number. Therefore, the total number of bits in S is given by $\mathbf{J} = \sum_{i=1}^{B} J_i$, so the manipulator can achieve $2^{\mathbf{J}}$ different configurations. S is constructed from the individual module states in such a way that the state of the manipulator base module corresponds to the most significant bits of S. I.e., $S = \{s_1 s_2 \ldots s_B\}$. Also, while the actuators in a given module of a truss manipulator have approximately the same "significance" in their effect on end-effector motion, we have chosen to number the bits within an individual truss from left to right (Figure 2), with the MSB being the leftmost actuator in the truss.

Configuration Set The kinematic properties of each module in the manipulator will be represented by a *configuration set*:

$$C_i = \{(\mathbf{R}_1, \mathbf{b}_1), (\mathbf{R}_2, \mathbf{b}_2), \ldots, (\mathbf{R}_{2^{J_i}}, \mathbf{b}_{2^{J_i}})\}, \quad (1)$$

where: $i = 1, \ldots, B$, $\mathbf{R}_j$ are orthonormal rotation matrices and $\mathbf{b}_j \in I\!R^N$ are translation vectors from the bottom center of a module to the top center of a module for $j = 1, \ldots, 2^{J_i}$. These sets describe all possible relative orientations and positions of frame i with respect to frame $i-1$.

The configuration set can be represented more efficiently for two-dimensional, planar manipulators, as:

$$C_i = \{(c_1, s_1, \mathbf{b}_1), (c_2, s_2, \mathbf{b}_2), \ldots, (c_{2^{J_i}}, s_{2^{J_i}}, \mathbf{b}_{2^{J_i}})\} \quad (2)$$

where: $i = 1 \ldots B$, θ_j denotes the angle of the top of module i relative to the bottom for state j (where: $j = 1 \ldots 2^{J_i}$), and s_j and c_j represent $\sin(\theta_j)$ and $\cos(\theta_j)$ respectively.

For the general case (Eq. 1), the information can either be stored in explicit form, or the rotation matrix can be represented using Euler angles, unit quaternions, etc. The choice is a trade-off between computational time spent on arithmetic operations and storage space.

3.2 Algorithm Implementation

The forward kinematics algorithm is implemented in two stages, a pre-computation stage (executed once for any set of kinematic parameters), which generates the configuration sets for the entire manipulator, followed by a stage (which may be executed many times) which computes the position of the manipulator's end effector for a particular state, S. A two-dimensional VGT based manipulator and a two-dimensional serial-revolute manipulator are used as examples in this section. The differences in the algorithm for the two types of manipulator are noted when appropriate. Note that the choice of manipulator type affects only the kinematic equations used in the pre-calculation stage. The main body of the algorithm is independent of manipulator type, except for the choice of a two (Eq. 2) or three (Eq. 1) dimensional configuration set. Even this dependence can be avoided if the configuration-set is always computed in the form of Equation 1, regardless of whether the manipulator is planar or three-dimensional.

3.2.1 Pre-Calculation

The purpose of the pre-calculation phase is to compute and store the configuration set of each module in the manipulator.

Inputs to the Algorithm:

1. q_j^{min}, q_j^{max}, the joint limits for each actuator in the manipulator in states 0 and 1 respectively, for $j = 1, \ldots, \mathbf{J}$.
2. w_i, the width of the top of module i, for a truss-type manipulator, or l_i, the length of link i in a serial-revolute manipulator, for $i = 1, \ldots, B$.

Module Kinematics:

For a serial-revolute manipulator (Figure 3) the kinematics of an individual module in a particular state are described by:

$$\mathbf{b}_i = (l_i \cos(\theta_i),\ l_i \sin(\theta_i)) \quad (3)$$

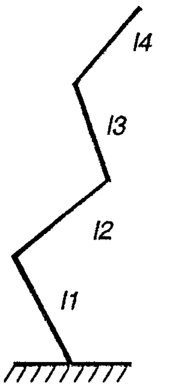

Figure 3: A four link serial-revolute manipulator.

$$\theta_i = \begin{cases} q_i^{min} & \text{if } s_i = 0 \\ q_i^{max} & \text{if } s_i = 1 \end{cases} \tag{4}$$

The forward kinematics for a planar VGT based manipulator are described in detail in [13]. The kinematics for a single truss are presented here for convenience. The forward kinematics obey the following geometric constraints (Refer to Figure 2):

$$\begin{aligned} x_1^2 + y_1^2 &= q_1^2 & (5)\\ (x_1 - w)^2 + y_1^2 &= q_2^2 & (6)\\ x_0^2 + y_0^2 &= q_0^2 & (7)\\ (x_0 - x_1)^2 + (y_0 - y_1)^2 &= w^2 & (8) \end{aligned}$$

These equations are solved simultaneously for the coordinates of the top plate of the truss:

$$\begin{aligned} x_1 &= -\frac{q_2^2 - q_1^2 - w^2}{2w} & (9)\\ y_1 &= \sqrt{\left(q_1^2 - \left(\frac{q_2^2 - q_1^2 - w^2}{2w}\right)^2\right)} & (10) \end{aligned}$$

and

$$\begin{aligned} x_0 &= \frac{-b - \sqrt{(b^2 - 4ac)}}{2a} & (11)\\ y_0 &= \sqrt{(q_0^2 - x_0^2)} & (12) \end{aligned}$$

where:

$$\begin{aligned} a &= 4x_1^2 + 4y_1^2 & (13)\\ b &= 4(w^2 x_1 - q_0^2 x_1 - q_1^2 x_1) & (14)\\ c &= q_1^4 + q_0^4 + 2q_0^2 q_1^2 - 2q_0^2 w^2 - & (15)\\ & \quad 2q_1^2 w^2 + w^4 - 4y_1^2 q_0^2 \end{aligned}$$

We can now solve for the position and orientation of the center of the top plate of the truss, for a particular module, i:

$$\begin{aligned} \mathbf{b}_i &= \left(\frac{(x_{0,i} + x_{1,i})}{2}, \frac{(y_{0,i} + y_{1,i})}{2}\right) & (16)\\ \theta_i &= \text{Atan2}(y_{1,i} - y_{0,i}, x_{1,i} - x_{0,i}) & (17) \end{aligned}$$

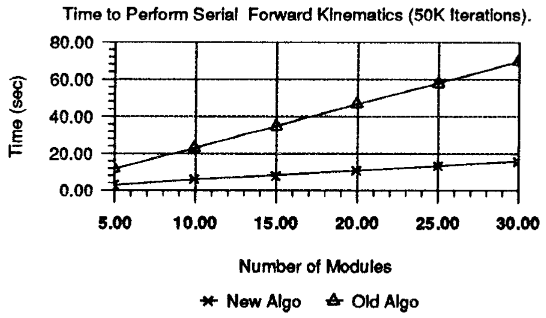

Figure 4: Comparison of "standard" kinematics algorithm with the algorithm in this paper for serial revolute robots with various numbers of modules. All tests were performed on a 33MHz NeXTStation Turbo Color system.

Pre-Computation Algorithm:
Once we know the forward kinematics of an individual module in the manipulator we can compute the configuration sets of the modules as follows:

```
for $i = 1$ to $B$
    for $j = 1$ to $2^{J_i}$
        Use the kinematic param-
        eters of module $i$ to com-
        pute $C_i$ for state $j$.
    end
end
```

3.2.2 Computation of End-Effector Position

Inputs to the Algorithm:

1. S, a **J** bit binary number representing the current state of the manipulator.

2. C_i, for $i = 1, \ldots, B$, the configuration sets for the modules in the manipulator.

Outputs from the Algorithm:

1. $\mathbf{EE}_{pos}$, the position of the manipulator's end effector.

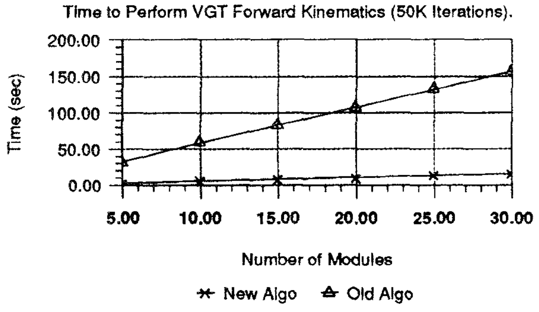

Figure 5: Comparison of "standard" kinematics algorithm with the algorithm in this paper for VGT robots with various numbers of modules. All tests were performed on a 33MHz NeXTStation Turbo Color system.

2. s_{ee}, c_{ee}, the sin and cos of the end-effector orientation angle for the two-dimensional case, or $\mathbf{R}_{ee}$, the rotation matrix describing end-effector orientation, in the three dimensional case.

Main Algorithm:
After completing the pre-calculation phase, as described in the last section, the position of the center of the manipulator's end-effector and its orientation can be computed as follows:

```
$\mathbf{EE}_{pos} = \mathbf{0}$
$\mathbf{R}_{ee} = \mathbf{I}$, the identity matrix.
for $i = 1$ to $B$
    select $C_i$, the rotations and position vectors for module $i$.
        $\mathbf{EE}_{pos} = \mathbf{EE}_{pos} + \mathbf{R}_{s_i}\mathbf{b}_{s_i}$
        $\mathbf{R}_{ee} = \mathbf{R}_{s_i}\mathbf{R}_{ee}$
    end
end
```

For a two-dimensional manipulator the sin and cos of the end-effector orientation can be used directly instead of a rotation matrix, to streamline the calculation. For either the two or three dimensional case, the algorithm requires $O(B)$ storage, while a "traditional" algorithm requires only constant storage. The main body of the algorithm requires $O(B)$ time to compute. This is the same order as an algorithm that computes the forward kinematics in the "obvious" way, by solving the kinematics of each module's structure whenever the end-effector position is needed. Nevertheless, the absence of transcendental functions

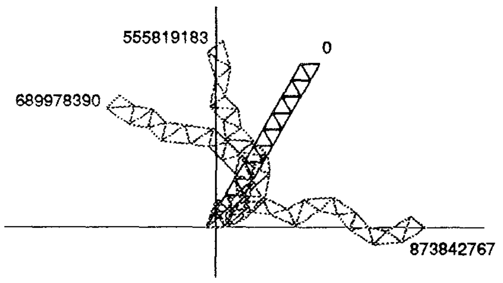

Figure 6: Forward kinematics calculations for several states of a VGT robot.

Platform	Old Algo	New Algo	Old/New
HP	5.64 sec.	0.79 sec.	7.09
PC	11.10 sec.	2.12 sec.	5.25
NeXT	61.44 sec.	5.86 sec.	10.48

Table 1: A comparison of the forward kinematics algorithm described in this paper with a "standard" algorithm, where the position and orientation are computed from the geometric parameters of the truss modules every time (As in Section 3.2.1). The 10 module VGT manipulator in Figure 1 was used for this test. All times are in seconds. All computer platforms were running NeXTStep Version 3.2. The difference in performance ratios is most likely caused by differences in the designs of the floating point units in the various processors. I.e, some architectures compute transcendental functions more efficiently. The HP is a Model 720/80 workstation, the PC is a 66Mhz Intel Pentium system, and the NeXT is a 33Mhz, NeXTStation Turbo.

in the algorithm presented here give it a very important practical advantage. For one computer architecture (see Section 3.3), it executed ten times faster than the standard approach.

3.3 Example of the Algorithm

Figure 6 shows the results of using the algorithm presented here to compute the forward kinematics of a VGT robot for several different states. Table 1 shows the time required to execute the forward kinematics algorithm on various computer architectures. The forward kinematics were evaluated 50,000 times for randomly selected configurations of the ten module manipulator shown in Figure 1. The top and bottom links of each module in the manipulator were 0.08 units wide, and the upper and lower actuator travel limits were 0.12 and 0.08 units repsectively for all actuated links. Figures 5 and 4 show the performances of the forward kinematics algorithm on one computer architecture for truss and serial robots of several different sizes. Even on a fast computer, using our algorithm provides a significant time savings.

4 Conclusion

Binary actuation offers us a way to create manipulators that are simultaneously rigid, accurate and inexpensive. The biggest obstacle to the use of binary manipulators has been the challenge of planning the motions of such highly redundant systems effectively. In this paper, we presented an efficient algorithm for computing the forward kinematics of manipulators with binary actuators. This work is useful by itself, but more importantly, this algorithm can vastly improve the performance of other algorithms for binary manipulators, such as those for inverse kinematics, obstacle avoidance, and trajectory following, that often require the frequent evaluation of forward kinematic equations.

References

[1] P. L. Bergstrom, T. Tamagawa, and D.L. Polla, "Design and fabrication of micromechanical logic elements", in *Proceedings of the IEEE Micro Electro Mechanical Systems Workshop*, Napa, CA, February 1990, pp. 15–20.

[2] J. Canny and K. Goldberg, "A risc paradigm for industrial robotics", Tech. Rep. ESRC 93-4/RAMP 93-2, Engineering Systems Research Center, University of California at Berkeley, 1993.

[3] M.T. Mason, "Kicking the sensing habit", *AI Magazine*, Spring 1993.

[4] K. Goldberg, "Orienting polygonal parts without sensors", *Algorithmica*, 1992, Special robotics issue.

[5] M. A. Erdmann and M. T. Mason, "Exploration of sensorless manipulation", *IEEE Journal of Robotics and Automation*, vol. 4, pp. 369–379, August 1988.

[6] D.L. Pieper, *The Kinematics of Manipulators under Computer Control*, PhD thesis, Stanford University, Stanford, CA, 1968.

[7] B Roth, J. Rastegar, and V. Scheinman, "On the design of computer controlled manipulators", *First CISM-IFTMM Symp. on Theory and Practice of Robots and Manipulators*, pp. 93–113, 1973.

[8] I. Ebert-Uphoff and G. S. Chirikjian, "Efficient workspace generation for binary manipulators with many actuators", *Journal of Robotic Systems*, June 1995.

[9] G.S. Chirikjian, "Kinematic synthesis of mechanisms and robotic manipulators with binary actuators", in *1994 ASME Mechanisms Conference*, Minneapolis, MN, Sept 1994.

[10] G. S. Chirikjian and D. Lees, "Inverse kinematics of binary manipulators with applications to service robotics", in *IROS 95*, August 1995.

[11] D. S. Lees and G. S. Chirikjian, "A combinatorial approach to trajectory planning for binary manipulators", in *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, 1996.

[12] I. Ebert-Uphoff and G. S. Chirikjian, "Inverse kinematics of discretely actuated hyper-redundant manipulators using workspace densities", in *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, 1996.

[13] G. S. Chirikjian, "A binary paradigm for robotic manipulators", in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, 1994, pp. 3063–3069.