Efficient Workspace Generation for Binary Manipulators with Many Actuators

Imme Ebert-Uphoff and Gregory S. Chirikjian*
Department of Mechanical Engineering
Johns Hopkins University
Baltimore, MD 21218

Accepted February 2, 1995

Binary actuators have only two discrete states, both of which are stable without feedback. As a result, manipulators with binary actuators have a finite number of states. Major benefits of binary actuation are that extensive feedback control is not required, task repeatability can be very high, and two-state actuators are generally very inexpensive, thus resulting in low-cost robotic mechanisms. Determining the workspace of a binary manipulator is of great practical importance for a variety of applications. For instance, a representation of the workspace is essential for trajectory tracking, motion planning, and the optimal design of binary manipulators. Given that the number of configurations attainable by binary manipulators grows exponentially in the number of actuated degrees of freedom, $\mathcal{O}(2^n)$, brute force representation of binary manipulator workspaces is not feasible in the highly actuated case. This article describes an algorithm that performs recursive calculations starting at the end-effector and terminating at the base. The implementation of these recursive calculations is based on the macroscopically serial structure and the discrete nature of the manipulator. As a result, the method is capable of approximating the workspace in linear time, $\mathcal{O}(n)$, where the slope depends on the acceptable error.

バイナリ・アクチュエータは2種類の状態だけを持ち、両方ともフィードバックを行わなくても安定している。その結果、バイナリ・アクチュエータを持ったマニピュレータの状態の数は、有限となる。バイナリ操作の主な利点は、広範囲のフィードバック制御を必要とせず、タスクの繰り返し性が非常に高くでき、また2種類の状態だけを持つアクチュエータは一般的に非常に安価であるために、ローコストのロボット・メカニズムが構築できることである。バイナリ・マニピュレータの作業空間の決定は、様々なアプリケーションに適用する場合に最も重要になる。例えば、作業空間の表現は、軌道追跡、動作計画、マニピュレータの最適設計に重要である。バイナリ・マニピュレータで可能な配置の数は、操作可能な d.o.f.の数 $O(2^n)$によって指数関数的に増加する。バイナリ・マニピュレータ作業空間のブルート力表現は、高度

*To whom all correspondence should be addressed.

Journal of Robotic Systems 12(6), 383–400 (1995)

CCC 0741-2223/95/060383-18

に操作されている場合には不適切である。この発表では、エンドエフェクタから始まりベースで終了する帰納的計算を実行するアルゴリズムについて説明する。これらの帰納的計算の組み込みは、マニピュレータの巨視的な直列構造と分散特性に基づいて行なう。その結果、この方法では直線時間 O(n) における作業空間の近似が可能となる。ただし、傾きは許容できる誤差によって決まる。

1. INTRODUCTION

The traditional assumption in robotics is that mechanisms are actuated with continuous-range-of-motion actuators such as d.c. motors. However, there are many applications of mechanisms and robotic manipulators that require only discrete motion. For these tasks, continuous-range-of-motion machines are overkill.

A binary actuator is one type of discrete actuator that has only two stable states (denoted "0" and "1"). As a result, binary manipulators have a finite number of states. Major benefits of binary actuation are that extensive feedback control is not required, task repeatability can be very high, and two-state actuators are generally very inexpensive (e.g., solenoids, pneumatic cylinders, etc.), thus resulting in low-cost robotic mechanisms.

In principle, an analogy can be made between continuous vs. binary manipulators and analog vs. digital circuits. In the history of electronics and computing, digital devices replaced many of their analog counterparts because of higher reliability and lower cost—exactly the same reasons for developing a binary paradigm for robotics.

One motivation for this work is that standard robotic manipulators (which are often used for pick-and-place tasks) have not been embraced by many industries because of their relatively high cost, low accuracy, and low payload capability as compared to dedicated machine tools. Similarly, many mechanism design problems do not require continuous range-of-motion actuation, but because that is what traditional methods address, those are the only kinds of designs considered.

While robotic hardware based on binary actuation is very inexpensive as compared to continuous range-of-motion robots, there are associated algorithmic and analytical issues in binary manipulator motion planning and design that are currently much more difficult than the corresponding problems in the continuous range-of-motion case. The goal of this article is to develop an efficient algorithm for the approximation of binary manipulator workspaces as a tool to make these algorithmic issues computationally tractable, and thus the cost benefits of binary manipulators attainable.

The number of configurations that a binary manipulator can attain is of the form 2^n where n is the number of binary actuators. It is easy to see that for n large enough (e.g., $n \approx 40$) the explicit computation and storage of all workspace points becomes impractical. Using this number of binary actuators in a given manipulator is not unrealistic. If for instance, one revolute joint is resolved into eight bits, a five-axis manipulator would have 40 bits/binary actuators. Therefore, no matter what aspect of these manipulators is to be studied (e.g., kinematic synthesis of binary manipulators with prescribed workspace characteristics; motion planning of binary manipulators; or man-machine interface) efficient methods for describing binary manipulator workspaces for large n are useful, and in fact necessary.

Figure 1 illustrates all possible configurations of a "3-bit" planar binary platform manipulator. A finite number of points are reachable by the gripper of the manipulator. In this case, 2^3 possible configurations result because there are three actuators. Note that for this design the location of points reachable by the end-effector is a function of the retracted cylinder length, extended cylinder length, and width of the platform. In the general case these kinematic parameters will be divided into joint stop and structural parameters, which for this case are denoted q^{min}, q^{max}, and a, respectively. In Figure 1, $q^{min} = 1$, $a = 1.2$ and $q^{max} = 1.5$.

A schematic of a highly actuated prototype is shown in Figure 2 for 2 of its almost 33,000 (2^{15}) end-effector positions along with its workspace. This particular design is a variable geometry truss manipulator. As currently configured, this manipulator consists of 15 identical prismatic actuators, each with two stable states (completely retracted, 0, or completely extended, 1). In these figures $q_i^{min} = 3/20$ and $q_i^{max} = 5/20$ for each cylinder, while the width of each platform is $a_i = 1/5$ for $i = 1, \ldots, 15$.

The remainder of this article is organized as follows. Section 2 reviews the literature. Section 3 presents the necessary background and definitions needed to formalize our approach. This background is necessary because the workspace description for a binary manipulator is different from that of standard continuous range-of-motion manipulators. Section 4 presents the *workspace mapping algorithm*

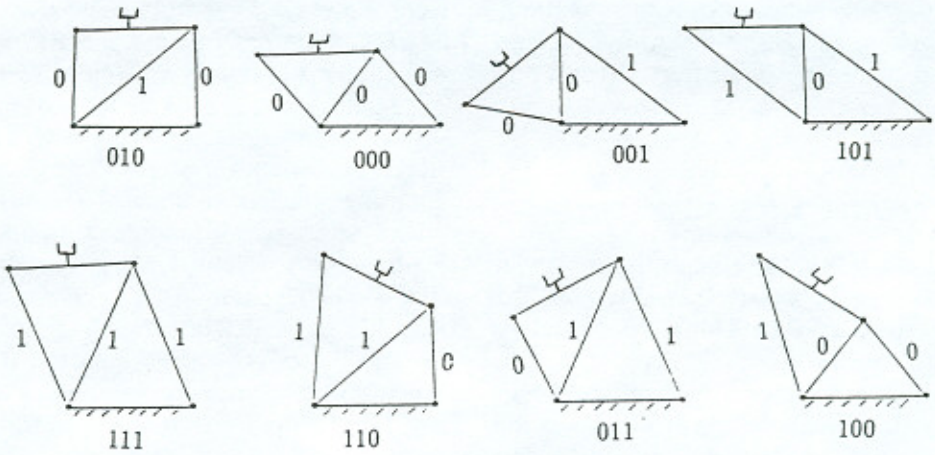
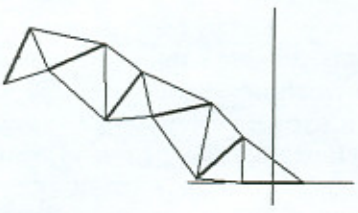


Figure 1. Configurations of a 3-bit platform manipulator.

Configuration 110001110001110



Configuration 001110001110001

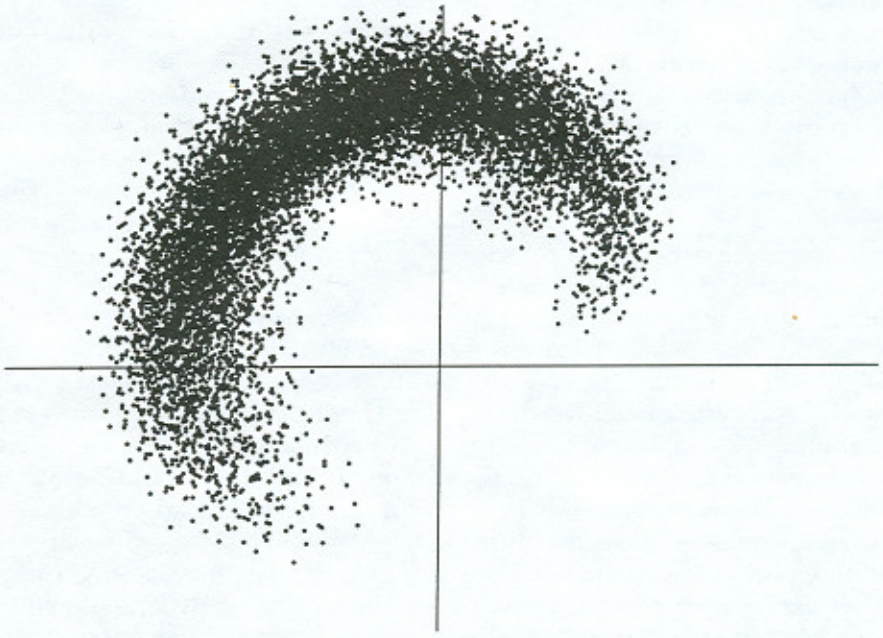


Figure 2. Sample configurations and workspace for a manipulator with 5 cascaded platforms.

(our approximate representation of binary manipulator workspaces) in detail. Section 5 presents results generated using this algorithm. Section 6 is the conclusion.

2. REVIEW OF RELATED LITERATURE

There are two distinct bodies of literature reviewed in the two following subsections. Subsection 2.1 discusses the history and development of binary mechanisms and "minimalist" robots. Subsection 2.2 reviews analysis methods for continuous range-of-motion manipulator workspaces presented in the literature. These two very different bodies of literature are both relevant to the current work.

2.1. The History of Binary Mechanisms

Due to the high cost/performance ratio of sophisticated robotic systems, a recent trend in "minimalist" robotics has begun to gain momentum. For instance, there have been recent efforts to develop new paradigms in robotics that parallel the development of reduced instruction set computers (RISC).[1] Related efforts include the investigation of *sensorless* robots.[2,3] In these efforts the mechanics of contact and pushing are used to formulate planning algorithms that are guaranteed to work provided particular physical constraints are observed. Thus, objects can be manipulated in a precise manner without the need for force feedback.

This trend runs counter to ideas that robots equipped with sophisticated tactile feedback systems provide the solution to all industrial robotics problems. However, the minimalist approach has not included robotic manipulators completely devoid of joint-level feedback (including position and velocity) until now.

Nonetheless, if one reviews the literature, sporadic efforts in binary actuation can be found.[4–6] Such efforts resulted when computers were first available to control robotic manipulators. However, despite the seemingly natural parallel between discretely actuated mechanical systems and the development of the computer, these efforts were abandoned for lack of a framework in which to design and plan well-behaved motions of such systems.

Of course, a natural question that one might raise is how different binary manipulators are from current systems that use stepper motors, or pick-and-place machines used in circuit board fabrication, or even flexible automation systems in which technicians set joint stops. The answer is that, just as in electronics, the true benefit of binary robotic mechanisms is not so much a function of their discrete nature as it is the reliability of having only two states. Moreover, simple robots with only a few binary actuators cannot perform complicated tasks such as obstacle avoidance. Therefore, the true benefit of a binary paradigm for robotics can only be exploited if a relatively large number of actuated degrees of freedom (DOF) are considered. In this way, the current work combines the trend towards minimalist (or sensorless) robots with issues in the kinematics and motion planning of high-degree-of-freedom ("hyper-redundant") manipulators.[7] Other recent works explore the design of binary manipulators to reach a small number of specified points.[8,9]

2.2. Determining Characteristics of Manipulator Workspaces

We review various methods for determining manipulator workspaces in this subsection. The impact of workspace characteristics on the choice of a robot for any particular application is substantial.[10]

Sen and Mruthyunjaya[11] use manipulators with only discrete joint states to model continuous-range-of-motion manipulators with limited joint resolution. The results are used to examine the positional accuracy of continuous-range-of-motion robots. In addition, an algorithm is developed to use these results to improve positional accuracy in motion planning processes for manipulators with only few degrees of freedom. Kumar and Waldron[12] describe a model that relates the joint accuracy to the accuracy of the end effector.

Blackmore and Leu examine the volume swept by a continuous-range-of-motion manipulator changing its configuration.[13] That work also develops the mathematical framework to analyze the swept volume of a manipulator arm.

The workspaces of manipulators with simple geometric structure can be derived using classical geometric approaches. These methods cover a wide range of special manipulator structures, but they are not applicable for manipulators of general structure. In particular there exist algorithms for the generation of workspaces for manipulators with only revolute joints.[14,15] These approaches address n-link manipulators, where n can be large.

Another approach solves the problem for manipulators with any kind of joints using the Monte-Carlo method. This approach generates a large number of random actuator values (joint angles),

and calculates the corresponding end effector positions.[16] An approximation of the workspace boundaries is found by tracking the border of this point set. In principle the Monte Carlo method is not restricted to manipulators with few degrees of freedom. However, the results lose reliability with increasing degrees of freedom.

Finally, Rastegar and Deravi consider workspace generation for a manipulator with general structure and n joints.[17] Those authors use an approach similar to the one presented in our article—dividing the manipulator into parts, for which subspaces are calculated. Other works by those authors include the effect of joint motion constraints on the workspace.[18]

3. BACKGROUND CONCEPTS AND DEFINITIONS

The goal of this section is to provide background needed to develop an efficient algorithm for the approximation of binary manipulator workspaces.

Intuitively, the approach presented here is to break up the workspace into pixels/voxels in the planar/spatial case, and calculate how many end-effector positions in each one are reached. This is done efficiently with an algorithm that adds the contributions of each section of the manipulator by performing recursive calculations starting at the end-effector and terminating at the base. The quantity calculated by the algorithm is called the *point density* of the workspace and will be represented by something called a *density array* (precise definitions to follow). The latter is a computer representation of the number of end-effector points for each pixel/voxel of the workspace.

The following subsections present the necessary background and definitions needed to formalize our approach. This background is necessary because the workspace description for a binary manipulator is different from that of standard continuous range-of-motion manipulators. Subsection 3.1 presents notation and definitions. Subsection 3.2 reviews module kinematics and discusses methods for storing the minimal amount of information needed for the mapping algorithm. Subsection 3.3 discusses the calculations associated with efficient representation of binary manipulator workspaces. In section 4, an efficient algorithm based on these ideas is developed to determine the point density of the workspace.

3.1. Concepts for Discrete Workspaces

In this subsection we motivate the goals stated earlier and provide background needed for the workspace mapping algorithm for binary manipulators. From now on we assume that the manipulator workspace W (a subset of $\mathbb{R}^N$) is divided into blocks (pixels or voxels) of equal size. The following definitions are useful:

Definition: The *point density* ρ assigns each block of $W \subset \mathbb{R}^N$ the number of binary manipulator states resulting in an end-effector position within the block, normalized by the volume of the block:

$$\rho(block) = \frac{\#\text{ binary manipulator states resulting in ee-position within block}}{\text{volume/area of workspace block}}$$

Since each binary manipulator state corresponds to exactly one configuration and a resulting end effector position, the density can also be defined as:

$$\rho(block) = \frac{\#\text{ of reachable points within block}}{\text{volume/area of block}},$$

where points are multiply counted when they are reachable by multiple binary manipulator configurations. The point density is important for binary manipulators because it is a measure of the positional accuracy of the end-effector, i.e., the higher the density is in the neighborhood of a point, the more accurately that point can be reached.

Definition: The *point density array*, or *density array* for short, is an N-dimensional array of integers ($D(i, j)$ for $N = 2$ or $D(i, j, k)$ for $N = 3$) in which each field/element corresponds to one block of the workspace and contains the number of binary manipulator states causing the end-effector to be in this block.

The density array provides a discretized version of the workspace from which point density is trivially calculated. Furthermore, the shape of a workspace is approximated by all blocks for which the corresponding entry in the density array is not zero. The borders of the shape can be approximated by curves from this information, but no efforts in this direction are made in the present work. For most applications only the discretized representation of the shape is needed. Other characteristics follow in a straightforward way: the size of a workspace, also as understood intuitively, is given by the area/volume of the chosen block size multiplied by the number of non-zero entries in the density array. The

calculation of moments of the workspace area/volume (e.g., center of mass, second moments, etc.) follow in a similar way. However, in this case entries of the density array are used as weights associated with the coordinates of all non-empty blocks.

The density array could be computed directly by calculating the end-effector positions for all possible binary manipulator configurations and counting the points reached in each block. This is a reasonable approach for binary manipulators with up to 20 actuators using currently available personal computer technology, but this method is not efficient for robots with many more binary degrees of freedom (e.g., 30–40).

To explain for what kinds of manipulators our approach is useful, and how it works, the following definitions are essential:

Definition: A *module* is a connected segment of a manipulator that changes its configuration (and thus that of the manipulator) by changing the states of actuators contained in itself, and is completely fixed in shape and size for given actuator states.

Definition: A *minimal module* is the smallest kinematically independent module in a manipulator. For example, the collection of two or more adjacent minimal modules is a module, but the resulting module is not minimal. Note that the manner in which a manipulator is decomposed into modules is not unique, whereas the decomposition into minimal modules is unique.

Definition: A *macroscopically serial manipulator* is a manipulator that is serial on a large scale, i.e., it can be represented by a serial collection of modules (where each module is mounted on top of the previous one). Closed loops may exist in each module, but macroscopic loops are not permitted. Any module partitions a macroscopically serial manipulator into distinct segments.

Definition: The i^{th} *intermediate workspace* of a macroscopically serial manipulator composed of B modules is the workspace of the partial manipulator from module $i + 1$ to the end-effector. (Modules are numbered 1 to B, from the base to the end of the manipulator.)

End-effector position vectors in the i^{th} intermediate workspace are written in a coordinate frame attached to the top of module i. To visualize this, imagine that the manipulator arm is cut between module i and module $i + 1$. The lower part (the first i modules) are ignored. The upper part (the most distal $B - i$ modules) are considered as a manipulator on its own, with its base in the separating plane. The workspace generated by this partial manipulator is the ith intermediate workspace of the whole structure.

Definition: An *affine transformation* in $\mathbb{R}^N$ is a transformation of the form $\mathbf{y} = \mathbf{Ax} + \mathbf{b}$, where $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{b}$ are vectors in $\mathbb{R}^N$, and $\mathbf{A}$ is an arbitrary matrix in $\mathbb{R}^{N\times N}$. A *homogenous transformation* is a special case of an affine transformation: $\mathbf{y} = \mathbf{Rx} + \mathbf{b}$, where $\mathbf{R}$ is a special orthogonal matrix, i.e., an orthogonal matrix with determinant 1. The set of all $(N \times N)$ special orthogonal matrices is denoted as $\mathbf{SO}(N)$ (which is a group under matrix multiplication). Members of this set are also called *rotation matrices*. Note that the set of all affine transformations in $\mathbb{R}^N$ is closed under the operation of composition, and that the set of all homogeneous transformations is a closed subset under the same operation. Furthermore, this subset is a group under the operation of composition denoted as $\mathbf{SE}(N)$.

Modules are numbered 1 to B, from the base of the manipulator to the end-effector. Each module has a frame attached to its top (according to the Danavit-Hartenberg convention in the serial case). The frames are numbered such that frame i is on top of module i, and frame 0 is the frame at the manipulator base. The number of independent binary actuators in module i is denoted J_i. Therefore there exist 2^{J_i} different combinations of binary actuator states (and corresponding configurations) for the i^{th} module.

Definition: Each module i with J_i binary joint angles, for $i = 1, \ldots, B$ will be represented by the *configuration set:*

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

where $\mathbf{R}_j \in \mathbf{SO}(N)$ are rotation matrices and $\mathbf{b}_j \in \mathbb{R}^N$ are translation vectors for $j = 1, \ldots, 2^{J_i}$. These pairs describe all possible relative orientations and positions of frame i with respect to frame $i - 1$.

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. Efficient Representation of Workspaces

For our algorithm to work, a computational tool is needed to efficiently store intermediate workspaces

for future use. Efficient representation is critical because intermediate workspaces may contain many points. Intermediate workspaces generated by modules composed of binary Stewart platforms as minimal modules can easily have millions of points.

We first formalize requirements for potential workspace representations before we propose one approach that satisfies all of them:

1. The amount of data stored at any time must be far less than the explicit storage of an intermediate workspace, which would require 2^k N-dimensional vectors for $k \leq n$.
2. The positional error caused by the representation of the workspace has to be small. In the ideal case it must stay below a given bound.
3. It is crucial that the workspace representation used supports efficient computation of affine transformations.
4. It is desirable to be able to quickly test whether a particular vector lies in an intermediate workspace.

While the first three conditions are necessary requirements, the last one is not needed for all applications. However it is a nice feature since it opens up a lot of new applications involving binary manipulator motion planning, e.g., obstacle avoidance.

We decided to use the point density array defined in subsection 3.1 to store all intermediate workspaces. It satisfies all four conditions. To restore or generate a workspace from a given density array some additional information, e.g., size and volume of each block, is needed. For this purpose, we define the following:

Definition: A *density set* is a computational structure containing the following information:

- A reference point $\mathbf{x}_0 \in \mathbb{R}^n$ that defines a point of the workspace in real coordinates. Here $\mathbf{x}_0$ is chosen to represent the middle point of a workspace. That is, each component of $\mathbf{x}_0$ is the middle of the interval bounded by minimal and maximal coordinate values of the workspace.
- The resolution of the discretization, i.e. block dimensions given by $\Delta\mathbf{x} = [\Delta x, \Delta y, \Delta z]^T$,
- The dimensions/length of the array in each direction, either in real (workspace) coordinates, $\mathbf{x}_L = [x_L, y_L, z_L]^T$, or as integers, i_L, j_L, k_L, giving the numbers of pixels/voxels for the particular resolution,
- The *density array*, D, of the workspace, which is an N-dimensional array of integers representing the point density of the workspace multiplied by block volume.

We denote a *density set* as

$$\mathcal{D} = \{D, \mathbf{x}_0, \Delta\mathbf{x}, \mathbf{x}_L\}.$$

An example of the description of a workspace as a density set is given in Figure 3 for the planar case. Note that the orientation of the end-effector is not stored because we only discretize in the workspace translational coordinates.

For the next section, in which the workspace mapping algorithm is presented, we need to know how to access the density array D. In other words, the following question must be answered: For given workspace coordinates $\mathbf{x} = (x, y, z)^T$, what are the corresponding array indices (i, j, k), and vice versa?

First (i_0, j_0, k_0) are chosen to be the indices corresponding to the middle point $\mathbf{x}_0$ of the workspace, such that the range of possible indices of the array is simply

$$i = 0, 1, \ldots, i_0, i_0 + 1, \ldots, 2i_0,$$

$$j = 0, 1, \ldots, j_0, j_0 + 1, \ldots, 2j_0,$$

$$k = 0, 1, \ldots, k_0, k_0 + 1, \ldots, 2k_0.$$

Note that with this definition the number of indices in each dimension is always odd. The rule to calculate workspace coordinates from array indices is:

$$\begin{aligned} x(i, j, k) &= x(i) = x_0 + \Delta x(i - i_0), \\ y(i, j, k) &= y(j) = y_0 + \Delta y(j - j_0), \\ z(i, j, k) &= z(k) = z_0 + \Delta z(k - k_0). \end{aligned} \tag{1}$$

The inverse problem is similar; however, the results must be rounded to the nearest integer. We use the notation of the *floor* operation $\lfloor . \rfloor$ to describe the rounding procedure. Round(x) denotes the nearest integer to x, while $[x]$ is defined to be the largest integer that is smaller or equal to x. The following relationship holds:

$$\text{Round}(x) = \lfloor x + 0.5 \rfloor,$$

so that the inverse problem is solved as follows:

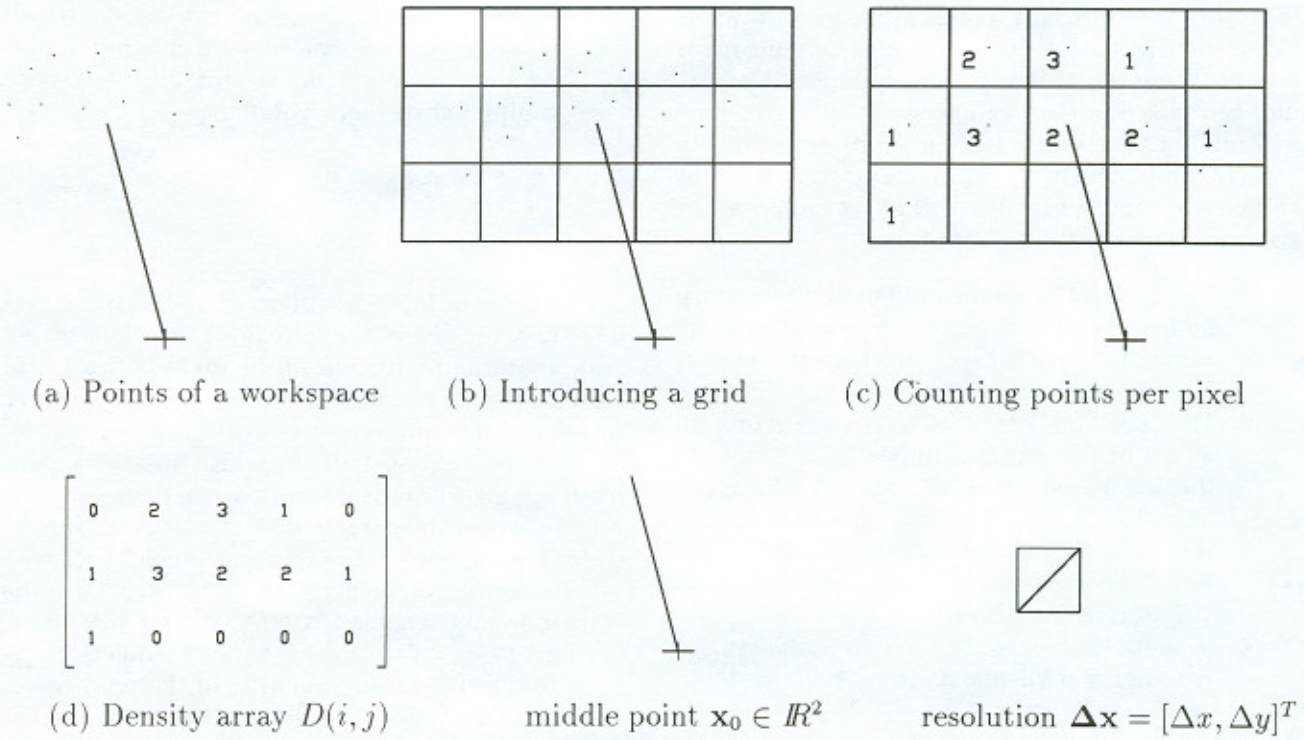


Figure 3. Representation of a workspace as a density set.

$$i(x, y, z) = i(x) = \left\lfloor \frac{(x - x_0)}{\Delta x} + 0.5 \right\rfloor + i_0,$$

$$j(x, y, z) = j(y) = \left\lfloor \frac{(y - y_0)}{\Delta y} + 0.5 \right\rfloor + j_0,$$

$$k(x, y, z) = k(z) = \left\lfloor \frac{(z - z_0)}{\Delta z} + 0.5 \right\rfloor + k_0. \quad (2)$$

The following section presents the mapping algorithm.

4. THE WORKSPACE MAPPING ALGORITHM

In this section, the workspace mapping algorithm is presented and analyzed. Subsection 4.1 presents an overview of the workspace mapping algorithm. Subsection 4.2 describes the implementation of one iteration of the algorithm in detail. Subsection 4.3 presents an error analysis. Because the workspace mapping algorithm is an approximation method, error analysis is important so that appropriate parameters can be chosen for the algorithm to yield acceptable results, e.g., determining what pixel dimensions yield reasonable accuracy. Subsection 4.4 discusses computational complexity and establishes a trade-off between accuracy and computation time.

4.1. An Overview of the Algorithm

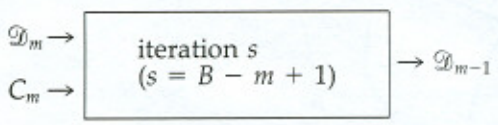
The goal of this subsection is to explain the workspace mapping algorithm on two different levels. The first part describes how the algorithm uses a recursive process to calculate the workspace of a manipulator. The second part draws a basic picture of one iteration. Subsection 4.2 describes the implementation of one iteration in greater detail.

As throughout this article, B denotes the number of modules of the manipulator under consideration. In addition the following indices are used throughout this section:

- Index s denotes the sth iteration of the mapping algorithm, ($s = 1, 2, \ldots, B$),
- Index m denotes the mth module considered in the sth step, ($m = B, B - 1, \ldots, 1$).

Recall that W_m is the intermediate workspace from the top of module m and W_{m-1} is the intermediate

workspace from the bottom of module m. These two workspaces are related to each other through the set, C_m, of all possible configurations of module m: $C_m = \{(\mathbf{R}_1^{(m)}, \mathbf{b}_1^{(m)}), (\mathbf{R}_2^{(m)}, \mathbf{b}_2^{(m)}), \ldots, (\mathbf{R}_{2^{l_m}}^{(m)}, \mathbf{b}_{2^{l_m}}^{(m)})\}$. One iteration of workspace mapping determines the density set $\mathcal{D}_{m-1}$ (representing the point density of workspace W_{m-1}) from given point density $\mathcal{D}_m$ and configuration set C_m. This situation is described graphically below:



A schematic of this procedure is given in Figure 4. The iterations of the algorithm are counted from 1 to B. Because the algorithm starts with the last module and propagates backwards the module number m considered at step s is $m(s) = B - s + 1$, for $s = 1, 2, \ldots, B$. Workspace W_B contains only one point because there are no actuators above the top of the most distal module. Thus the density array D_B consists of a single element/field, containing the value 1.

The algorithm can therefore be summarized as follows: It starts with the trivial density set $\mathcal{D}_B$. The first iteration determines $\mathcal{D}_{B-1}$, the second determines $\mathcal{D}_{B-2}$, etc. After B iterations the algorithm ends providing the density set $\mathcal{D}_0$ of the complete manipulator arm.

The following describes iteration s of the algorithm, which deals with module $m = B - s + 1$, in more detail:

1. Estimate size and location of intermediate workspace W_{m-1} (details to follow). Based on this information:
   (a) Choose the dimensions of a block in the new density array: $(\Delta x^{(m-1)}, \Delta y^{(m-1)}, \Delta z^{(m-1)})$.
   (b) Based on these dimensions determine the number of fields of the density array in each direction: $(i_L^{(m-1)}, j_L^{(m-1)}, k_L^{(m-1)})$.
   (c) Allocate sufficient memory for this density array and initialize it with zeros.
   (d) Determine the coordinates of the middle point of the new workspace: $(x_0^{(m-1)}, y_0^{(m-1)}, z_0^{(m-1)})$.
   (e) Determine the array indices, $(i_0^{(m-1)}, j_0^{(m-1)}, k_0^{(m-1)})$, of the middle point of the new array.
2. For all configurations $(\mathbf{R}_l^{(m)}, \mathbf{b}_l^{(m)}) \in C_m$, $(l = 1, \ldots, 2^{l_m})$, apply the corresponding homogeneous transformation to the density array D_m:

   For all indices (i, j, k) for which the entry $D_m(i, j, k)$ of the density array D_m is not zero, the following steps are applied:
   (a) Calculate the vector $\mathbf{x} = (x(i), y(j), z(k))^T$ from the array indices (i, j, k) according to Eq. (1).
   (b) Calculate the coordinate vector $\mathbf{x}' = \mathbf{R}_l^{(m)}\mathbf{x} + \mathbf{b}_l^{(m)} \in W_{m-1}$.
   (c) Find the array indices (i', j', k') of $\mathbf{x}'$ in the new array according to Eq. (2).
   (d) Increment entries in the block of the new array by the corresponding entry of the old array:

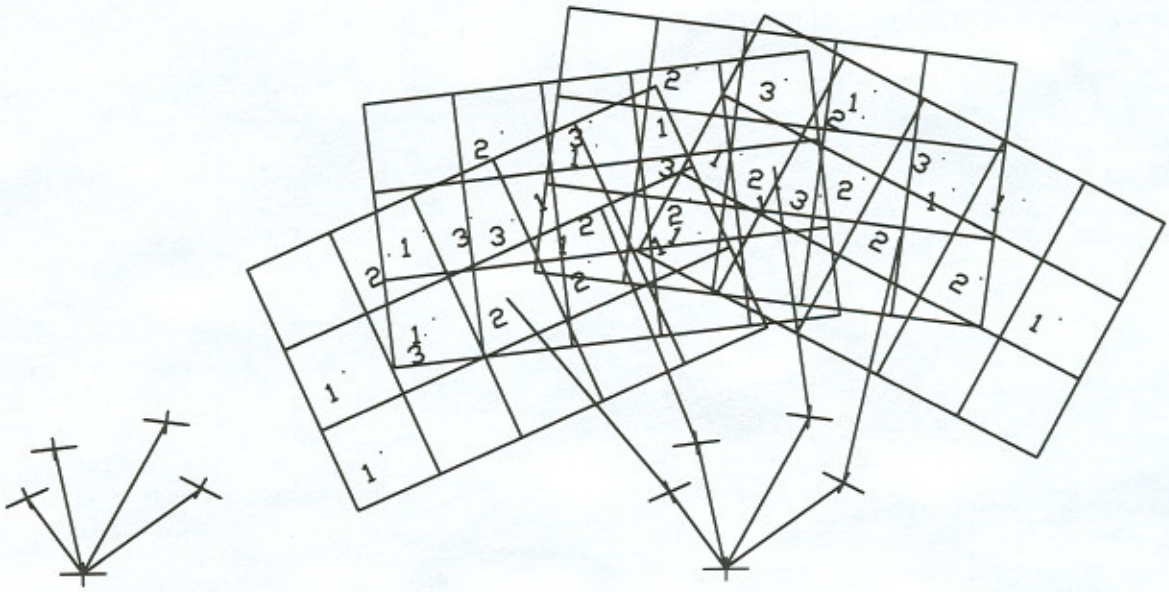
$$D_{m-1}(i', j', k') \leftarrow (D_{m-1}(i', j', k') + D_m(i, j, k)) \quad (3)$$

To estimate size and location of workspace W_{m-1}, all 2^{l_m} homogeneous transforms are applied to the eight corners of the density array D_m (four for the planar case). The resulting maximal and minimal values in each coordinate axis are taken as the boundaries of the next density array, D_{m-1}. Because the actual workspace W_{m-1} is smaller than this estimate, a memory overhead is produced that would decrease the efficiency of the next iteration of the algorithm. For this reason an additional reduction procedure is implemented, which detects and cuts out the smallest part of the density array that contains the whole workspace. The computational complexity for both of these parts, the estimate and the reduction procedure, are not significant compared to the mapping process itself.

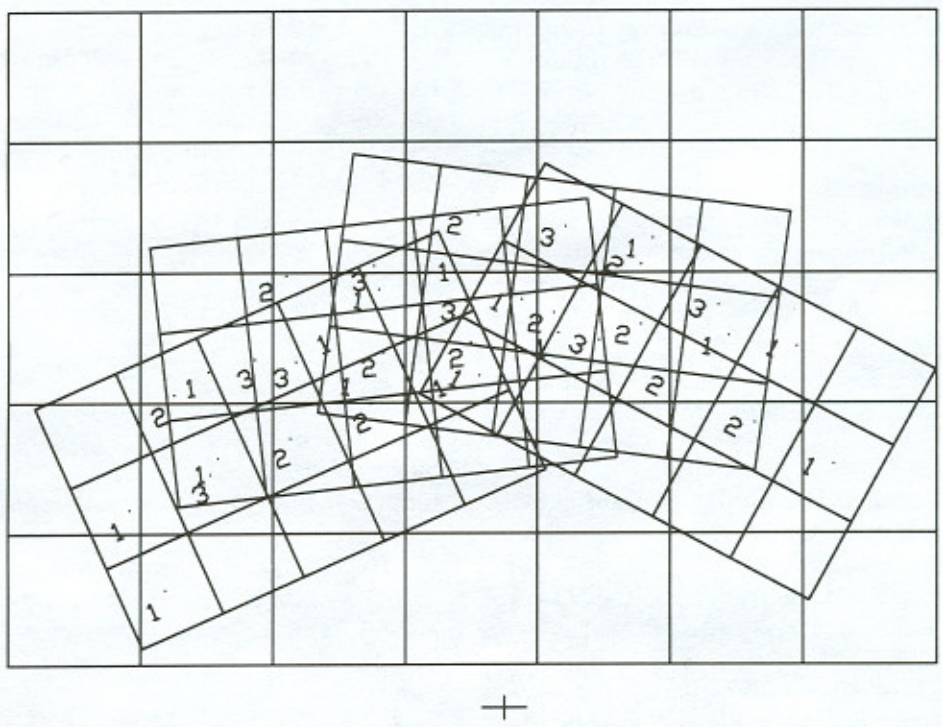
The next section shows the implementation of these steps. In particular the operations described in 2(a)-(c), which build the core of each iteration, are simplified so that the effort is reduced.

4.2. Implementation of One Iteration

The core part of iteration s is the transformation of the indices of density array D_m to the indices of density array D_{m-1}, where $m = B - s + 1$. This is the most important part, because it has to be performed for each configuration and for all indices corresponding to non-zero entries in D_m. Hence it is the main contribution of the computational complexity of the algorithm. The emphasis of this section is on



(a) Configurations of one module and resulting overlay of density sets



(b) Summing the points in the new grid leads to the next density set

Figure 4. Schematic of one recursion of the workspace mapping algorithm.

the simplification and implementation of this part of the calculation.

To eliminate indices that complicate the presentation of one step of the algorithm, the following notation is used: All variables belonging to workspace W_m or density set $\mathcal{D}_m$ have names without superscript, while variables belonging to W_{m-1} or $\mathcal{D}_{m-1}$ have a superscript ('). For one particular configuration $(\mathbf{R}, \mathbf{b}) = (\mathbf{R}_i^{(m)}, \mathbf{b}_j^{(m)}) \in C_m$, and one particular set of indices (i, j, k), the three steps to calculate

the indices (i', j', k') from (i, j, k) as described in section 4.1, 2 (a)-(c), are illustrated.

The first part is described by Eq. (1) and can be written as an affine transformation:

$$\mathbf{x} = \begin{pmatrix} \Delta x & 0 & 0 \\ 0 & \Delta y & 0 \\ 0 & 0 & \Delta z \end{pmatrix} \begin{pmatrix} i \\ j \\ k \end{pmatrix} + \begin{pmatrix} x_0 - i_0 \Delta x \\ y_0 - j_0 \Delta y \\ z_0 - k_0 \Delta z \end{pmatrix}.$$

The second part is a homogeneous transformation (a special type of affine transformation):

$$\mathbf{x}' = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \mathbf{R}\mathbf{x} + \mathbf{b}$$

The third and last part has the form

$$\begin{pmatrix} i'(x') \\ j'(y') \\ k'(z') \end{pmatrix} = \begin{pmatrix} \left\lfloor \frac{(x' - x_0')}{\Delta x'} + 0.5 + i_0' \right\rfloor \\ \left\lfloor \frac{(y' - y_0')}{\Delta y'} + 0.5 + j_0' \right\rfloor \\ \left\lfloor \frac{(z' - z_0')}{\Delta z'} + 0.5 + k_0' \right\rfloor \end{pmatrix},$$

according to Eq. (2), which is also an affine transformation, followed by a *floor* operation $\lfloor . \rfloor$ in each coordinate.

The composition of affine transformations is always an affine transformation itself. Hence we can find constants $a_1, a_2, a_3, b_1, \ldots, d_3$, such that the general transformation from (i, j, k) to (i', j', k') has the form:

$$\begin{aligned} i' &= \lfloor a_1 i + b_1 j + c_1 k + d_1 \rfloor, \\ j' &= \lfloor a_2 i + b_2 j + c_2 k + d_2 \rfloor, \\ k' &= \lfloor a_3 i + b_3 j + c_3 k + d_3 \rfloor. \end{aligned} \tag{4}$$

As an example of how to calculate the constants $a_1, a_2, \ldots, d_3$ in Eq. (4), the constants for the first of these relationships, a_1, b_1, c_1, and d_1 are determined. In this context $(\mathbf{R})_{uv}$ denotes the coefficient of the matrix $\mathbf{R}$ in row u and column v, and $(\mathbf{b})_u$ denotes the uth coefficient of vector $\mathbf{b}$. Substitution of the equations above into the relationship

$$i' = \left\lfloor \frac{(x' - x_0')}{\Delta x'} + 0.5 + i_0' \right\rfloor = \lfloor a_1 i + b_1 j + c_1 k + d_1 . \rfloor$$

leads to the first four constants (see equation 5).

Result for iteration $s = B - m + 1$: For each configuration $(\mathbf{R}, \mathbf{b}) = (\mathbf{R}_l^{(m)}, \mathbf{b}_l^{(m)})$, the constants $a_1, \ldots, d_3$ are calculated according to relationships as given by Eq. (5) for the first four constants. Second, Eq. (4) is applied to all relevant indices (i, j, k) using these constants. The resulting indices (i', j', k') are used to increment the entries $D_{m-1}(i', j', k')$ according to assignment (3). Note, that the entries of the same density array D_{m-1} are changed while handling all $l = 1, \ldots, 2^{l_m}$ configurations, without reinitializing. This implements the superposition of all parts of workspace W_{m-1}.

4.3. Error Analysis

The previous subsections developed an efficient method to approximate the point density of the workspace of binary manipulators with macroscopi-

---

$$\begin{aligned} a_1 &= (\mathbf{R})_{11} \frac{\Delta x}{\Delta x'}, \\ b_1 &= (\mathbf{R})_{12} \frac{\Delta y}{\Delta x'}, \\ c_1 &= (\mathbf{R})_{13} \frac{\Delta z}{\Delta x'}, \\ d_1 &= \frac{(\mathbf{R})_{11}(x_0 - i_0 \Delta x) + (\mathbf{R})_{12}(y_0 - j_0 \Delta y) + (\mathbf{R})_{13}(z_0 - k_0 \Delta z) + (\mathbf{b})_1 - x_0'}{\Delta x'} \\ &\quad + 0.5 + i_0'. \end{aligned} \tag{5}$$

cally serial structure. In this subsection an error analysis is performed. Results from this analysis are used in the next subsection, where a trade-off between complexity and error is developed.

The following tracks the error occurring at the recursive steps of the algorithm in the intermediate workspaces, caused by representing the workspace as a density set. An upper bound on the maximal error at each step of the algorithm is derived.

Recall that B denotes the number of modules in the manipulator, which means that the algorithm has B steps and there are $(B + 1)$ intermediate workspaces. The intermediate workspaces are denoted by W_m as in previous sections. Again, the relationship $m(s) = B - s + 1$ describes the number m of the module that is treated in iteration s of the algorithm. For the following let

- $\mathbf{x}_m$ denote any exact end-effector position in workspace W_m ($\mathbf{x}_m \in W_m$),
- $\tilde{\mathbf{x}}_m$ denote the corresponding vector calculated by the mapping algorithm before discretization at step $s = B - m + 1$, and
- $\hat{\mathbf{x}}_m$ denote the vector calculated by the mapping algorithm after this discretization.

Definition: The *maximal error* E_m of the workspace mapping algorithm in workspace W_m is an upper bound of the distance between vector $\mathbf{x}_m \in W_m$ and the corresponding approximation $\hat{\mathbf{x}}_m$ calculated by the algorithm. Using the Euclidean norm as a measure of the length of a vector, $\left\| \begin{pmatrix} x \\ y \\ z \end{pmatrix} \right\|_2 = \sqrt{x^2 + y^2 + z^2}$, the maximal error is defined as $E_m = \|\mathbf{x}_m - \hat{\mathbf{x}}_m\|_2$.

Definition: The *block radius* r_m is defined as half the diagonal length of a block of density set $\mathcal{D}_m$, which is:

$$r_m = \frac{1}{2}\sqrt{\Delta x_m^2 + \Delta y_m^2 + \Delta z_m^2} = \frac{1}{2}\|\Delta \mathbf{x}_m\|_2,$$

where $\Delta \mathbf{x}_m$ is the vector of block dimensions of the density set $\mathcal{D}_m$.

Consider a vector $\tilde{\mathbf{x}}_m$ before discretization, as defined above. After the discretization $\tilde{\mathbf{x}}_m$ is represented by the middle point, $\hat{\mathbf{x}}_m$, of the block of the density set $\mathcal{D}_m$ in which $\tilde{\mathbf{x}}_m$ lies. Therefore the maximal possible distance between $\tilde{\mathbf{x}}_m$ and $\hat{\mathbf{x}}_m$ is just the block radius of density set $\mathcal{D}_m$, i.e., $\|\hat{\mathbf{x}}_m - \tilde{\mathbf{x}}_m\|_2 \leq r_m$.

As a result, the following relationships hold:

$E_m = \|\mathbf{x}_m - \hat{\mathbf{x}}_m\|_2$
maximal error in workspace m

$\mathbf{x}_{m-1} = \mathbf{R}_m \mathbf{x}_m + \mathbf{b}_m$

$\tilde{\mathbf{x}}_{m-1} = \mathbf{R}_m \hat{\mathbf{x}}_m + \mathbf{b}_m$

$\|\hat{\mathbf{x}}_m - \tilde{\mathbf{x}}_m\|_2 \leq r_m$
discretization error at step $s = B - m + 1$

and therefore

$$\begin{aligned} E_{m-1} &= \|\mathbf{x}_{m-1} - \hat{\mathbf{x}}_{m-1}\|_2 \\ &\leq \|\mathbf{x}_{m-1} - \tilde{\mathbf{x}}_{m-1}\|_2 + \|\tilde{\mathbf{x}}_{m-1} - \hat{\mathbf{x}}_{m-1}\|_2 \\ &\leq \|(\mathbf{R}_m \mathbf{x}_m + \mathbf{b}_m) - (\mathbf{R}_m \hat{\mathbf{x}}_m + \mathbf{b}_m)\|_2 + r_{m-1} \\ &= \|\mathbf{R}_m (\mathbf{x}_m - \hat{\mathbf{x}}_m)\|_2 + r_{m-1} \\ &= \|\mathbf{x}_m - \hat{\mathbf{x}}_m\|_2 + r_{m-1} \\ &= E_m + r_{m-1}. \end{aligned}$$

The mapping algorithm begins with

$$\mathbf{x}_B = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = \hat{\mathbf{x}}_B = \tilde{\mathbf{x}}_B \Rightarrow E_B = 0$$

Therefore:

$$\begin{aligned} E_{B-1} &\leq E_B + r_{B-1} = r_{B-1} \\ E_{B-2} &\leq E_{B-1} + r_{B-2} \leq r_{B-1} + r_{B-2} \\ &\vdots \leq \vdots \\ E_0 &\leq E_1 + r_0 \leq r_{B-1} + r_{B-2} + \cdots + r_0 \end{aligned}$$

such that

$$E_m = \sum_{b=m}^{B-1} r_b$$

is an upper bound on the error in workspace W_m. In particular

$$E_0 = \sum_{b=0}^{B-1} r_b$$

is an upper bound on the maximal error of the algorithm.

4.4. Computational Complexity

This subsection discusses the trade-off between memory, time, and accuracy for the algorithm. It consists of three parts: Subsection 4.4.1 discusses how the composition of minimal modules to larger units can decrease the error of the algorithm in exchange for some additional calculations. Subsection 4.4.2 addresses the computational complexity of the core of the mapping algorithm. Subsection 4.4.3 combines the results of the first two, and suggests a strategy to adjust the algorithm to given requirements of accuracy, time, and memory.

In this context we denote:

- $U_1, U_2, \ldots, U_B$, the number of configurations of each module. $U_m = 2^{l_m}$ for $m = 1, 2, \ldots, B$.
- $W_B, W_{B-1}, \ldots, W_0$, the intermediate workspace (in the order calculated by the algorithm). Note, that the algorithm starts with a special workspace at the end-effector, denoted W_B, which contains only the origin (0, 0, 0).
- $\mathcal{D}_m$, $(m = B, B-1, \ldots, 0)$, the density set corresponding to workspace W_m. D_m denotes the density array of density set $\mathcal{D}_m$. $\mathcal{D}_B$ consists of only one block, with entry 1, representing the origin.
- C_m, $(m = B, B-1, \ldots, 0)$, the configuration set of module m.
- V_m, $(m = B, B-1, \ldots, 0)$, the volume of the smallest box enclosing workspace W_m that is aligned with the axes of the mth frame (the area of the smallest enclosing rectangle for the planar case).
- P_m, $(m = B, B-1, \ldots, 0)$, the number of blocks, used to store workspace W_m in a density set $\mathcal{D}_m$. Special case: $P_B = 1$.

4.4.1. Error Reduction by Choosing Module Size

This subsection describes how the size of the modules can reduce the error of the workspace mapping algorithm. Starting the algorithm with modules composed of a number of minimal modules realizes a trade-off between *direct calculation* of all end-effector positions (no error, with large computational and memory requirements) and the *pure mapping algorithm* (not exact, but much smaller computational and memory requirements).

Accordingly two opposing factors are to be balanced: Choosing the modules very large results in few recursions in the mapping algorithm. On the other hand, the effort to calculate all configurations of each module in advance grows exponentially with its number of binary actuators. As a general rule, the above calculations are not allowed to exceed the computational order of the mapping algorithm (whose complexity is determined in 4.4.2).

Consider a module m, which consists of p minimal modules with $u_1, u_2, \ldots, u_p$ configurations, respectively. All $U = u_1 u_2 \ldots u_p$ configurations of the composed module can be calculated by combining the configuration sets of the minimal modules. For our approach the effort to calculate all configurations of module m is approximately proportional to the number of configurations to be determined, $c_m = \lambda_C U_m$, so that the total effort for these calculations is

$$C_I = \sum_{m=1}^{B} c_m = \lambda_C \sum_{m=1}^{B} U_m.$$

The memory is specified by the number of configurations to be stored $(U_1 + U_2 + \cdots + U_B)$ times the amount of memory needed to store one configuration:

$$M_I = \lambda_M \sum_{m=1}^{B} U_m \text{ floats.}$$

There is no error arising from this calculation except that which is caused by floating point arithmetic. Therefore, $E_I = 0$.

4.4.2. Complexity of the Main Algorithm

Let us consider one step of the mapping algorithm $(s = 1, 2, \ldots, B)$: Intermediate workspace W_m is represented by a density set $\mathcal{D}_m$, where $m = m(s) = B - s + 1$. The number of elements/fields in D_m is P_m. The next module to be considered is module m, which has $U_m = 2^{l_m}$ different configurations, i.e., U_m affine transformations are applied. For each transformation the constants $a_1, a_2, a_3, b_1, \ldots, d_3$ have to be determined. This takes only 18 floating point multiplications and 12 additions, and is negligible compared to the effort for the affine transformation applied to *all* the pixels.

The essence of the mapping process for each

configuration is to transform the indices of all blocks of D_m to indices of D_{m-1}, i.e.

$$i' = \lfloor a_1 i + b_1 j + c_1 k + d_1 \rfloor,$$

$$j' = \lfloor a_2 i + b_2 j + c_2 k + d_2 \rfloor,$$

$$k' = \lfloor a_3 i + b_3 j + c_3 k + d_3 \rfloor,$$

must be evaluated for all possible i, j, k within the dimensions of the density array. This can be done in three nested loops, extracting all constants that are invariant. This way the computational complexity for one affine transformation of all fields of an array D_m is approximately $3P_m T_{FM}$, where T_{FM} is the average time needed to perform one floating point multiplication and addition. This number, multiplied by the number of affine transformations to be performed, (i.e., the number of configurations U_m) gives the total effort for step s, ($s = B - m + 1$):

$$c^{(s)} = 3P_m U_m T_{FM} = 3P_{B-s+1} U_{B-s+1} T_{FM}.$$

Summing over all steps, ($s = 1, \ldots, B$), results in

$$C_{II} = 3 \sum_{s=1}^{B} (P_{B-s+1} U_{B-s+1}) T_{FM} = 3 \sum_{m=B}^{1} (P_m U_m) T_{FM},$$

$$= 3 \sum_{m=1}^{B} (P_m U_m) T_{FM},$$

where $P_B = 1$. (In the special case of two dimensions the factor in front is 2 instead of 3.)

The memory needed for the algorithm depends on the application: for motion planning tasks the information from all intermediate workspaces is needed. On the other hand, to generate the final workspace (for instance, to see how changes in kinematic parameters change the workspace), intermediate workspaces need not be retained after their information has been used, i.e., at most two density sets, $\mathfrak{D}_m$ and $\mathfrak{D}_{m-1}$, have to be stored at a time. In this case the memory needed can be estimated as:

$$M_{II} = 2 \max_{i=0}^{B} P_i \text{ integers}$$

The error caused by the mapping algorithm was already determined in subsection 4.3.

4.4.3. Total Complexity, Memory, and Error

Here the results of the two previous parts are added together to yield the final complexity, memory requirements, and error of the algorithm:

$$C = C_I + C_{II} = \lambda_C \sum_{m=1}^{B} U_m + 3 \sum_{m=1}^{B} (P_m U_m) T_{FM},$$

$$M = M_I + M_{II} = \lambda_M \sum_{m=1}^{B} U_m \text{ floats} + 2 \max_{i=0}^{B} P_i \text{ ints},$$

$$E = E_I + E_{II} = 0 + \sum_{m=0}^{B-1} r_m,$$

where r_m is half the diagonal of a block in density array D_m as defined in subsection 4.3.

In general vectors are rotated in all directions, which means that the errors are transferred from one coordinate axis to the next one. A high resolution in only one coordinate direction does not help to keep the error in this direction small. For this reason we choose the resolution to be the same in each coordinate axis: $\Delta x_m = \Delta y_m = \Delta z_m$. (In two dimensions Δz_m is ignored.) The expression for the block radius r_m, as defined in subsection 4.3, simplifies to:

$$r_m = \frac{1}{2} \sqrt{\Delta x_m^2 + \Delta y_m^2 + \Delta z_m^2}$$

$$= \frac{1}{2} \sqrt{3 \Delta x_m^2} = \frac{\sqrt{3}}{2} \Delta x_m,$$

and the total error E takes the form:

$$E = E_0 = \sum_{m=0}^{B-1} r_m = \frac{\sqrt{3}}{2} \sum_{m=0}^{B-1} \Delta x_m.$$

To adjust the algorithm to given requirements we can choose either the resolutions Δx_m or the number of blocks P_m for each step. If one of them is given, the other one results from the relationship:

$$V_m = P_m \Delta x_m \Delta y_m \Delta z_m = P_m \Delta x_m^3,$$

$$\text{i.e. } P_m = \frac{V_m}{\Delta x_m^3} \text{ or } \Delta x_m = \sqrt[3]{\frac{V_m}{P_m}}.$$

In the planar case the cubic root is replaced by a square root, and volume is replaced by area. V_m is given by the morphology of the robot and its kinematic parameters. Note that the intermediate workspaces are growing as the algorithm propagates down the manipulator, so that: $V_B < V_{B-1} < \cdots < V_0$, and in particular $V_B << V_0$.

The total computational complexity C, the memory M, and the total error E can now be formulated in terms of the same variables, $P_0, P_1, \ldots, P_B$, $(P_B = 1)$:

$$E = \frac{\sqrt{3}}{2} \sum_{m=0}^{B-1} \sqrt[3]{\frac{V_m}{P_m}}$$

$$= \frac{\sqrt{3}}{2} \left(\sqrt[3]{\frac{V_0}{P_0}} + \cdots + \sqrt[3]{\frac{V_{B-1}}{P_{B-1}}} \right)$$

$$C = \lambda_C (U_1 + U_2 + \cdots + U_B) + 3(U_B + P_{B-1}U_{B-1} + \cdots + P_1U_1)T_{FM}$$

$$M = \lambda_M(U_1 + U_2 + \cdots + U_B) \text{ floats} + 2 \max_{i=1}^{B} P_i \text{ ints}$$

The goal is to keep Δx_i small for high accuracy and not to increase computational complexity and memory too much. Because memory tends to be the critical factor (as compared to time) we decided to base our strategy on the direct control of the number of pixels used in the approximations. P_i is chosen to be constant for all workspaces,

$$P \stackrel{def}{=} P_0 = P_1 = \cdots = P_{B-1}.$$

This implies the following complexity of the algorithm in terms of the free parameter P:

$$E = \frac{\sqrt{3}}{2\sqrt[3]{P}} \left(\sqrt[3]{V_0} + \cdots + \sqrt[3]{V_{B-1}} \right)$$

$$C = \lambda_C (U_1 + U_2 + \cdots + U_B) + 3P(U_1 + \cdots + U_{B-1})T_{FM} + 3U_BT_{FM}$$

$$M = \lambda_M (U_1 + U_2 + \cdots + U_B) \text{ floats} + 2P \text{ ints}$$

We now draw a rough picture of how these terms increase in the number of modules B, and their dependence on the free parameter P based on the following two assumptions: (1) the volume V_s grows with the order of three in the number of modules, $V_s \approx \lambda_V (m(s))^3 = \lambda_V(B - s + 1)^3$, (in the planar case $V_s \approx \lambda_V(B - s + 1)^2$), and (2) the module size is kept small. Then the error takes the form

$$E = \frac{\sqrt{3}}{2\sqrt[3]{P}} (\sqrt[3]{\lambda_V(B-1)^3} + \sqrt[3]{\lambda_V(B-2)^3} + \cdots + \sqrt[3]{\lambda_V 0^3})$$

$$= \frac{\sqrt{3}\sqrt[3]{\lambda_V}}{2\sqrt[3]{P}} (1 + \cdots + (B-1))$$

$$= \frac{1}{\sqrt[3]{P}} \frac{\sqrt{3}\sqrt[3]{\lambda_V}}{2} \left(\frac{(B-1)B}{2} \right),$$

and the algorithm has the following behavior:

- The error E grows quadratically in the number of modules B with a very small factor. Dependence on chosen parameter P is as factor $1/\sqrt[3]{P}$.
- The memory M is approximately linear in B and P.
- The time C is approximately linear in B and P.

To get a smaller error, P is increased.

Breaking the manipulator up into optimal sized modules is a more complicated task. The reason is that if the number of minimal modules per module is changed, this not only effects the numbers U_m and B, but it also changes the geometrical data $V_1, V_2, \ldots$ of the workspaces.

Due to this complexity we have not found a perfect solution for this problem yet. However, choosing U_B as large as possible, because U_B is the only parameter not multiplied by P in the expression for time C, $(U_B >> U_{B-1})$, and $U_{B-1} \approx U_{B-2} \approx \cdots \approx U_1$ gives good results. An example for the choice of B and $U_1, U_2, \ldots, U_B$ for a binary truss manipulator and the results for different numbers of modules are given in the next section together with running time and resulting error.

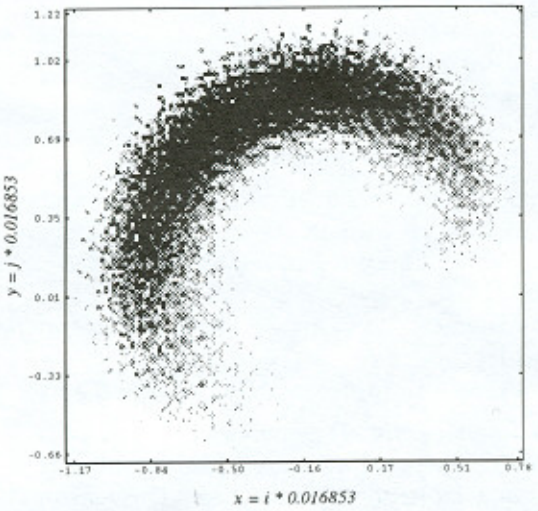
5. NUMERICAL RESULTS

This section presents numerical results for the workspace mapping algorithm. It was implemented for the planar case on a SUN 4/40 SPARC station IPC in the C programming language. Figures were made using Mathematica version 2.2.

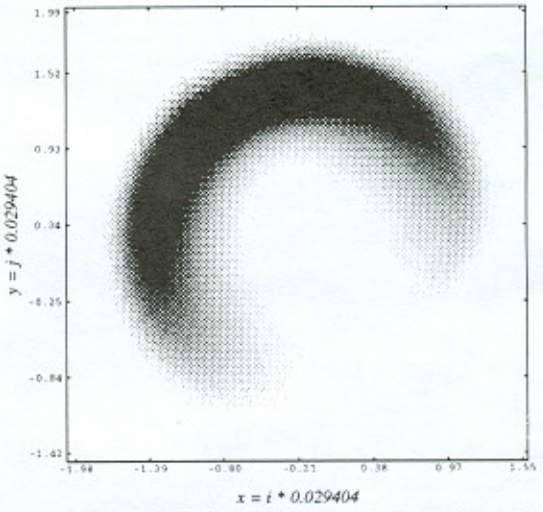
The algorithm is applied to the binary truss manipulator described in section 1 and illustrated in Figure 2. The joint values were chosen as in Figure 2, to allow the comparison of the workspace approximation for a 5-platform manipulator with the exact workspace, shown in Figure 2 ($q_i^{min} = \frac{3}{20}$, $q_i^{max} = \frac{5}{20}$, for $i = 1, \ldots, 15$).

The minimal modules for a truss manipulator are the platforms. Each platform consists of three binary joints, such that it has $2^3 = 8$ different configurations.

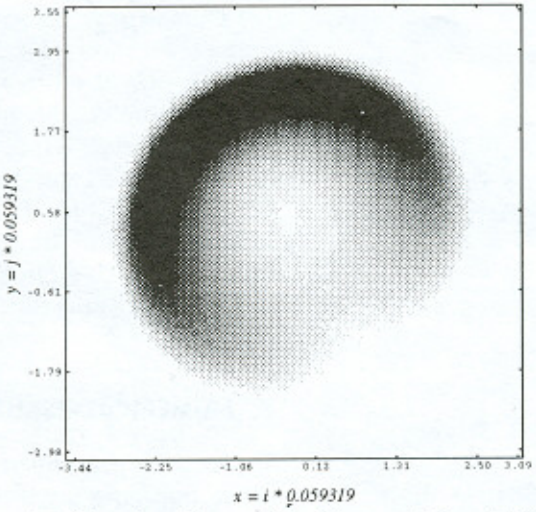
The maximal number of pixels, P, in the algorithm is chosen as $P = 20{,}000$. The first step ($s = 1$) of the algorithm treats the most distal 4 platforms as one module. The remaining steps ($s = 2, \ldots, B$) treat 2 platforms as a single module. If the total number of platforms is odd, the last iteration ($s = B$) considers only one platform as a module. The following summarizes the resulting parameters of the algorithm for a manipulator with an even number of platforms. Given parameters are:



(a) Manipulator with 5 modules (15 Bits), $E_5 = 0.018858$



(b) Manipulator with 8 modules (24 Bits), $E_8 = 0.041790$



(c) Manipulator with 14 modules (42 Bits), $E_{14} = 0.146833$

Figure 5. Point density of a manipulator workspace for 5, 8, and 14 modules.

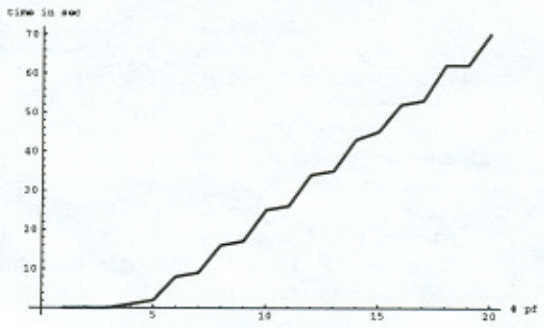
- M, the number of platforms. M is assumed to be even for simplicity of the explanation.
- $P = 20{,}000$, the maximal number of blocks in the representation of a workspace.

Resulting parameters:

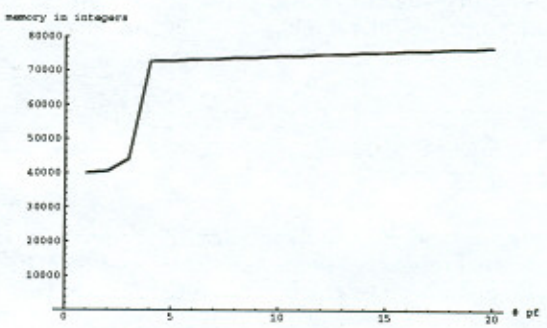
- the number of modules considered by the algorithm: $B = M/2 - 1$.
- the numbers of joints per module: $J_B = 4 \cdot 3 = 12$, $J_1 = \cdots = J_{B-1} = 2 \cdot 3 = 6$,
- the numbers of configurations per module: $U_B = 2^{J_B} = 2^{12} = 4096$, $U_1 = \cdots = U_{B-1} = 2^6 = 64$,
- the numbers of blocks in the representation of a workspace W_m: special case $P_B = 1$, $P_m \approx P = 20000$, for $m = 0, \ldots, B - 1$.

Figures 5a–5c show the results for a binary truss manipulator with 5, 8, and 14 platforms, respectively, each with actuator strokes stated earlier. E_m denotes the maximal error for points in each approximation. Figure 5a, which presents the workspace of the manipulator with 5 platforms, can be compared to the direct results, shown in Figure 2. Note that the length scales of the figures differ for different numbers of platforms.

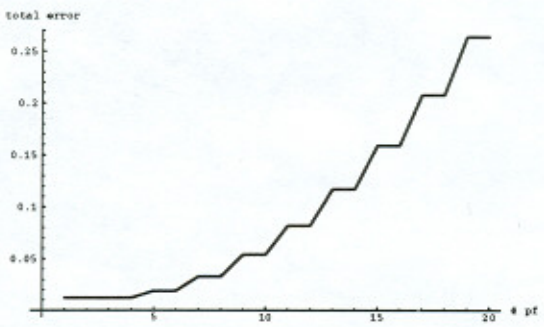
Time, error, and memory were tracked during the calculations. Figure 6a shows the effective user time C, including all calculations except the time for printing and storing the data in a file. While the general trend is linear, discrete jumps in this graph are a result of how the platforms are grouped as larger modules. Figure 6b shows the memory M needed for the calculations. The amount of memory needed is given in terms of a number of integers. Memory used in the form of double variables is converted into integers using a factor 2, because on a SPARC station an integer is represented by 4 bits, a double variable by 8 bits. The memory is approximately constant because the storage of intermediate workspaces is the main factor, and the number P_m of blocks is chosen to be constant. Figure 6c shows the



(a) Time C



(b) Memory M



(c) Error E

Figure 6. Time C, memory M, and error E of the algorithm for the example of section 5.

maximal error E in the approximation of the workspaces. Points in the approximation may differ from the exact positions by at most the distance E. The jumps again follow from the grouping of platforms.

6. CONCLUSIONS

This article has presented an efficient algorithm for generating an approximation to the workspace of robotic manipulators with binary (two-state) actuators. The error associated with this method was analyzed, and a trade-off between accuracy and computation time/memory was derived, and illustrated by an example.

This article is one step in the development of a binary paradigm for robotics. This paradigm has tremendous potential as an alternative to standard continuous range-of-motion actuation. However, challenging algorithmic issues remain in the design and motion planning of these devices.

This work was made possible by the NSF National Young Investigator Award IRI-9357738 from the Robotics and Machine Intelligence Program, and a Presidential Faculty Fellows Award.

REFERENCES

1. 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.
2. K. Goldberg, "Orienting polygonal parts without sensors," *Algorithmica* (special robotics issue), 1992.
3. M. Mason, "Kicking the sensing habit," *AI Mag.*, Spring 1993.
4. V. Anderson and R. Horn, "Tensor arm manipulator design," *Trans. ASME*, **DE-57**, 1967.
5. D. Pieper, "The kinematics of manipulators under computer control," Ph.D. dissertation, Stanford University, Oct. 1968.
6. B. Roth, J. Rastegar, and V. Scheinman, "On the design of computer controlled manipulators," *Proc. CISM-IFTMM Symp. Theory and Practice Rob. Manip.*, 1973, pp. 93–113. (see pp. 106–108).
7. G. Chirikjian and J. Burdick, "A modal approach to hyper-redundant manipulator kinematics," *IEEE Trans. Rob. Autom.* June 1994.
8. G. Chirikjian, "Kinematic synthesis of mechanisms and robotic manipulators with binary actuators," *Proc. ASME Mech. Conf.*, Minneapolis, Sept. 1994.
9. G. Chirikjian, "A binary paradigm for robotic manipulators," *Proc. IEEE Int. Conf. Rob. Autom.*, San Diego, CA, May 1994.
10. Y. Koren, *Robotics for Engineers*, McGraw-Hill, New York, 1985.
11. D. Sen and T. S. Mruthyunjaya, "A discrete state perspective of manipulator workspaces," *Mech. Mach. Theory*, **29**(4), 591–605, 1994.
12. A. Kumar and K. J. Waldron, "Numerical plotting of surfaces of positioning accuracy of manipulators," *Mech. Mach. Theory*, **16** (4), 361–368, 1980.
13. D. Blackmore and M. Leu, "Analysis of swept volume via Lie groups and differential equations," *Int. J. Rob. Res.* **11**(6), 516–537, 1992.
14. J. U. Korein, *A Geometric Investigation of Reach*, MIT Press, Cambridge, MA, 1985.
15. S.-J. Kwon, Y. Youm, and K. C. Chung, "General algorithm for automatic generation of the workspace for n-link planar redundant manipulators," *Trans. ASME* **116**, 967–969, 1994.
16. D. G. Alciatore and C.-C. D. Ng, "Determining manipulator workspace boundaries using the Monte Carlo method and least squares segmentation," *ASME Rob.: Kinematics Dyn. Controls*, **DE-72**, 141–146, 1994.
17. J. Rastegar and P. Deravi, "Methods to determine workspaces, it subspaces with different numbers of configurations and all the possible configurations of a manipulator, *Mech. Mach. Theory*, **22**(4), 343–350, 1987.
18. J. Rastegar and P. Deravi, "The effect of joint motion constraints of the workspace and number of configurations of manipulators," *Mech. Mach. Theory*, **22**(5), 401–409, 1987.