Generation of Binary Manipulator Workspaces and Work Envelopes

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

Abstract

A binary manipulator is a discrete manipulator whose actuators have only two states. We present an efficient algorithm for the approximation of workspaces and work envelopes for binary manipulators of highly actuated structure. The approximation describes not only the shape of the workspace, but also its local point density, i.e. the distribution of the number of points per unit area/volume. This characteristic of manipulator workspaces and work envelopes is of great practical importance for binary manipulators for a variety of problems, e.g., inverse kinematics and obstacle avoidance.

The method extends naturally to the continuous range-of-motion case for any manipulator that can be approximated as a binary manipulator with a sufficiently large number of bits. This is particularly useful for manipulators with low resolution, since the point density provides a measure for the local positional accuracy of the end-effector.

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 which 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 [1].

The goal of this paper is to develop an efficient algorithm for the approximation of binary manipulator workspaces and work envelopes for highly actuated structures. The *workspace*, $W \subset I\!R^N$, is the set of all positions reachable by the end-effector. A *work envelope* is the boundary of the smallest simply connected subset of $I\!R^N$ which contains the whole manipulator structure undergoing all configurations of the manipulator.

The importance of manipulator workspace properties for design considerations is well known, e.g., [2, 3], as is the importance of manipulator work envelopes, especially in the context of obstacle avoidance.

In this paper, we explicitly consider binary manipulator workspaces and work envelopes. The methodology can be adjusted to analyze general discrete manipulators, and extends to the continuous range-of-motion manipulators if they can be approximated as binary manipulators with a large number of bits. Note that the approximation describes not only the shape of the workspace, but also its local point density (precise definition to follow). For manipulators with low resolution this is useful, since the point density provides a measure for the local positional accuracy of the end-effector, in this context see [4, 5].

The number of configurations that a binary manipulator can attain is of the form 2^n where n is the number of binary actuators. In order to reach a large number of points n tends to be large for binary manipulators. 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. Therefore an algorithm to determine binary manipulator workspaces and work envelopes for large n becomes necessary.

A schematic of a highly actuated prototype is shown in Figure 1 for two of its almost thirty three thousand (2^{15}) end-effector positions. 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. The actuator lengths here are respectively 3/20 and 5/20. The platform widths are each 1/5. For this protoype it is still possible to calculate all points of the workspace explicitely by enumerating all configurations (Figure 2). This workspace will be used

*Graduate Student
†Assistant Professor, Presidential Faculty Fellow

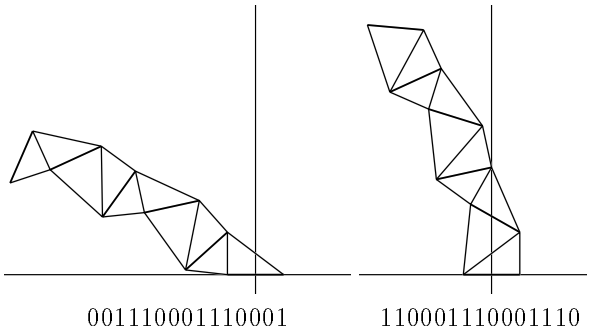


Figure 1: Sample configurations for a manipulator with 5 cascaded platforms

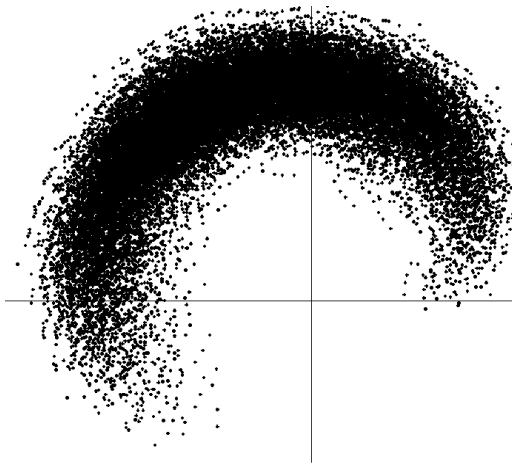


Figure 2: Workspace for a manipulator with 5 cascaded platforms

for a comparison with the output of the algorithm presented in this paper.

The remainder of this paper is organized as follows: Section 2 presents the necessary background and definitions needed to formalize our approach. Section 3 presents the *workspace mapping algorithm* (our approximation algorithm for binary manipulator workspaces). Section 4 presents numerical examples and applies the algorithm to the development of work envelopes. Section 5 is the conclusion.

2 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 and work envelopes.

Intuitively, the approach presented here is to break up the workspace into pixels, and calculate how many end-effector positions in each one are reached. This is done efficiently with an algorithm that superposes the contributions of each section of the manipulator by performing recursive homogeneous transformations starting at the end-effector and terminating at the base. In a sense, the whole workspace is generated by repeatedly performing a discrete convolution product of the workspaces associated with individual segments of the manipulator. The computation of the work envelope is also a superposition of the workspaces of subsections of the manipulator, but these computations start at the base and work upwards.

The computations required by our algorithm to generate the workspace for a manipulator composed of m concatenated modules of a given module design, are of order $\mathcal{O}(m)$, where the slope depends on the design of the module. The work envelope is generated using m runs of the workspace algorithm, and therefore requires $\mathcal{O}(m^2)$ computations.

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 of the workspace.

The following subsections present the necessary background and definitions 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 2.1 presents notation and definitions. Subsection 2.2 discusses the storage of information needed for the mapping algorithm.

2.1 Concepts for Discrete Workspaces

The workspace of a continuous range-of-motion manipulator is often described by its boundary, all points in the interior of which can be reached. For a binary manipulator the situation is quite different since only a finite number of points can be reached. Therefore not only the boundary of the workspace is important, but also the distribution of the points inside this boundary.

From now on we assume that the manipulator workspace W (a subset of $\mathbb{R}^N$) is divided into blocks (pixels) of equal size. The distribution of the points is described as follows: The *point density* ρ assigns each block of $W \subset \mathbb{R}^N$ the number of points within the block that are reachable by the binary manipulator, normalized by the volume of the block:

$$\rho(\text{block}) = \frac{\#\text{ reachable points in block}}{\text{unit volume/area}}$$

The point density serves as a probabilistic measure of the positional accuracy of the end-effector in a certain area of the workspace. The higher the density in the neighborhood of a point, the more accurately we expect to be able to reach the point.

The *density array* is an N-dimensional array of integers ($D(i,j)$ for $N=2$ or $D(i,j,k)$ for $N=3$) in which each element corresponds to one block of the workspace and contains the number of reachable points in this block. The density array provides a discretized version of the

workspace from which point density is trivially calculated (multiplication with a constant). Furthermore, the shape of a workspace is approximated by all blocks for which the corresponding entry in the density array is not zero.

The following definition specifies the type of manipulator for which our algorithm is used. 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. Modules are numbered 1 to B, from the base to the end of the manipulator. Closed loops may exist in each module, but macroscopic loops are not permitted. Note: any module partitions a macroscopically-serial manipulator into distinct segments.

The i^{th} *(upper) intermediate workspace*, W_i^I, of a macroscopically serial manipulator composed of B modules is the workspace of the manipulator segment from module $i+1$ to the end-effector. The i^{th} *(lower) partial workspace*, W_i^P, of the same manipulator is the workspace of the segment from the base to module i. To visualize this, imagine that the manipulator arm is cut between module i and module $i+1$. 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 manipulator segment is the ith intermediate workspace of the whole structure. The lower part (the first i modules) determine the i^{th} partial workspace.

The *workspace mapping algorithm* is based on the sequential calculation of the intermediate workspaces, starting at the end-effector and ending at the base. The fact that these workspaces are the intermediate results of the algorithm is responsible for the naming. The partial workspaces are only used for the generation of work envelopes.

Each module has a frame attached to its top. 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. 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 $\mathbf{b}_j \in I\!R^N$ are translation vectors for $j = 1, ..., 2^{J_i}$. These pairs describe all possible relative orientations and positions of frame i with respect to frame $i-1$.

2.2 Efficient Representation of Workspaces

Our algortihm is based on determining intermediate workspaces in sequence. Therefore a conceptual tool is needed to efficiently store intermediate workspaces for future use. Efficient representation is critical because intermediate workspaces may contain many points, e.g., intermediate workspaces generated by modules composed of binary Stewart platforms can easily have millions of points.

We use the point density array defined in Subsection 2.1 to store all intermediate workspaces. 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: The *density set* associated with an intermediate or partial manipulator is a computational structure containing the following information:

- A reference point $\mathbf{x}_0 \in I\!R^n$ which 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 $\mathbf{\Delta x} = [\Delta x, \Delta y, \Delta z]^T$,
- 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 for the particular resolution,
- The *density array*, $D(i, j, k)$, of the workspace, which is a N-dimensional array of integers representing the point density of the workspace multiplied by block volume.

We denote a *density set* as

$$\mathcal{D} = \{D, \mathbf{x_0}, \mathbf{\Delta x}, \mathbf{x_L}\}.$$

Note that the orientation of the end effector is not stored since we only discretize in the workspace translational coordinates.

We denote the density set of the i^{th} intermediate workspace as $\mathcal{D}_i^I$, and the density set of the i^{th} partial workspace as $\mathcal{D}_i^P$. In the special case when the manipulator is composed of identical modules, $\mathcal{D}_i^I = \mathcal{D}_i^P$, but generally this will not be the case.

3 The Workspace Mapping Algorithm

The generation of manipulator workspaces and work envelopes is primarily a matter of generating the partial and intermediate workspaces and convolving them in the appropriate way. In this section, the workspace mapping algorithm, which is an efficient way of generating these intermediate and partial workspaces, is presented. Subsection 3.1 presents an overview of the workspace mapping algorithm. Subsection 3.2 describes the implementation

of one iteration of the algorithm in detail. Further details of the implementation, an analysis of the computational complexity of the algorithm in terms of time and memory and the error resulting from the discretization can be found in [6].

3.1 An Overview of the Algorithm

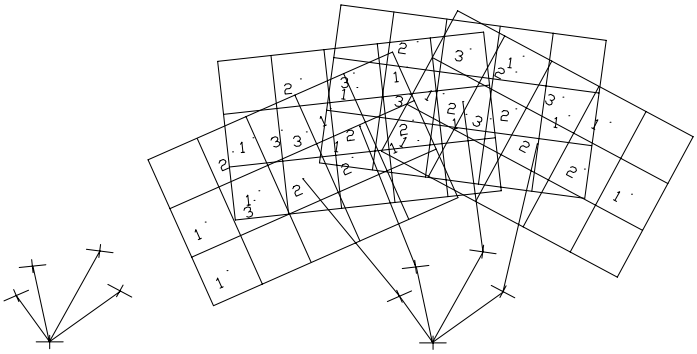
The workspace mapping algorithm determines intermediate workspaces starting at the end-effector and ending at the base. At each step we climb down one module, maintaining an approximation of the intermediate workspace corresponding to the segment of all modules above the current one. In this subsection we explain how the intermediate workspace at a given level has to be transformed to yield the next intermediate workspace including one more module.

Throughout this section B denotes the number of modules of the manipulator under consideration. 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. The algorithm starts with the last module and propagates backwards. Hence the module number m considered at step s of the algorithm is $m(s) = B - s + 1, \quad s = 1, 2, \ldots, B$, i.e. m is decreasing, while s is increasing.

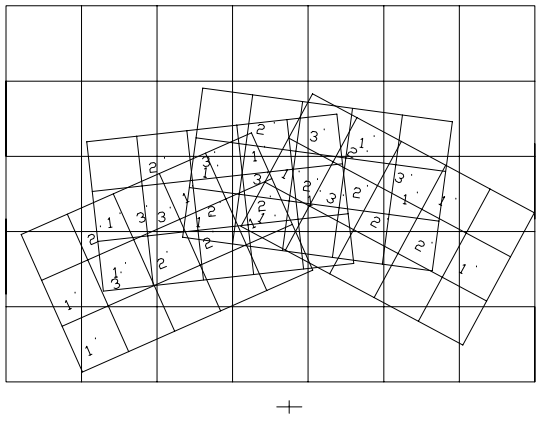
By our definition W_m^I denotes the intermediate workspace from the top of module m and W_{m-1}^I is the intermediate workspace from the bottom of module m. These two workspaces are related to each other through the set C_m of all possible configurations of module m: $C_m = \{(\mathbf{R}_1^{(m)}, \mathbf{b}_1^{(m)}), (\mathbf{R}_2^{(m)}, \mathbf{b}_2^{(m)}), \ldots, (\mathbf{R}_{2^{J_m}}^{(m)}, \mathbf{b}_{2^{J_m}}^{(m)})\}$. One iteration of workspace mapping determines the density set $\mathcal{D}_{m-1}^I$ (representing the point density of workspace W_{m-1}^I) from given point density $\mathcal{D}_m^I$ and configuration set C_m.

A schematic of this procedure is shown in Figure 3. To simplify the graphical representation a manipulator with only four states per module is considered. Figure 3(a) shows at the left the four configurations (C_m) of the module under consideration in iteration s in terms of the translation and rotation of a frame. The second input to the procedure is the density set $\mathcal{D}_m^I$. It is represented by one rectangle that is divided into blocks of equal size with an integer associated to each of them. An additional reference point represents the position of the center of the rectangle with respect to the origin. The way the density sets (rectangles) are superposed depends on the configurations of the module, as shown on the right of Figure 3(a). An introduction of a wider grid on top of the superposed density sets is shown in Figure 3(b). It only remains to add all entries in each pixel of the new grid to get the output of this iteration: density set $\mathcal{D}_{m-1}^I$.

As can be seen in the picture one iteration only consists of a set of homogenous transformations and the superposition of the results. On a more abstract level this can be seen as a discrete convolution determined by the configurations C_m applied to the input $\mathcal{D}_m^I$.



(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 3: Schematic of one recursion of the workspace mapping algorithm

3.2 Implementation

We start the discussion of the details with a summary of the algorithm: The algorithm starts with the density set $\mathcal{D}_B^I$. The first iteration determines $\mathcal{D}_{B-1}^I$, the second determines $\mathcal{D}_{B-2}^I$, etc. After B iterations the algorithm terminates providing the point density $\mathcal{D}_0^I$ of the complete manipulator arm.

The first implementation detail is the question of the intermediate workspace to start with: Workspace W_B^I is the first workspace to be considered. It describes the location of the end-effector relative to the top of the most distal module. In general this is simply one point, typically the location of the center of the end-effector. However, for the purpose of work envelope generation we

1. Estimate size and location of intermediate workspace W_{m-1}^I. Based on this information:
   (a) Choose the dimensions of a block in the new density array: $\left(\Delta x^{(m-1)}, \Delta y^{(m-1)}, \Delta z^{(m-1)}\right)$.
   (b) Based on these dimensions determine the number of fields of the density array in each direction: $\left(i_L^{(m-1)}, j_L^{(m-1)}, k_L^{(m-1)}\right)$.
   (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: $\left(x_0^{(m-1)}, y_0^{(m-1)}, z_0^{(m-1)}\right)$.
   (e) Determine the array indices, $\left(i_0^{(m-1)}, j_0^{(m-1)}, k_0^{(m-1)}\right)$, of the middle point of the new array.

2. For all configurations $(\mathbf{R}_l^{(m)}, \mathbf{b}_l^{(m)}) \in C_m$, $(l = 1, \ldots, 2^{J_m},)$ apply the corresponding homogeneous transformation to the density array D_m^I:

   For all indices (i, j, k) for which the entry $D_m^I(i, j, k)$ of the density array D_m^I is not zero:

   (a) Calculate the vector $\mathbf{x} = (x(i), y(j), z(k))^T$ from the array indices (i, j, k).
   (b) Calculate the coordinate vector $\mathbf{x}' = \mathbf{R}_l^{(m)}\mathbf{x} + \mathbf{b}_l^{(m)} \in W_{m-1}^I$.
   (c) Find the array indices $\left(i', j', k'\right)$ of $\mathbf{x}'$ in the new array.
   (d) Increment the entry of the block of the new array by the corresponding entry of the old array:

$$D_{m-1}^I(i', j', k') \quad \leftarrow \quad \left(D_{m-1}^I(i', j', k') + D_m^I(i, j, k)\right)$$

Figure 4: Implementation of one recursion of the workspace mapping algorithm

choose W_B^I to be the discretized shape of the end-effector including the upper part of the last module. In any case density set $\mathcal{D}_B^I$ can easily be determined from this information.

A schematic of one iteration is given in Figure 3 and was explained in the previous subsection. Figure 4 lists the major steps of the implementation of one iteration and is further explained below.

The first block in Figure 4 describes the administrational part of an iteration: an estimate of size and location of the intermediate workspace to be calcaluted is needed, the resolution has to be chosen and memory has to be allocated accordingly. For details we again refer the reader to [6].

To estimate size and location of workspace W_{m-1}^I, all 2^{J_m} homogeneous transforms are applied to the eight corners of the density array D_m^I (four for the planar case). The resulting maximal and minimal values in each coordinate axis provide a conservative estimate for the boundaries of the next density array, D_{m-1}^I. An additional reduction procedure is implemented to reduce a resulting memory overhead after the workspace is calculated. The computational complexity for the estimate and the reduction procedure are not significant compared to the essential mapping process described in the second block.

The second block in Figure 4 describes the implementation of the homogenous transformations. They are implemented by steps (a) - (d), in the interior of the lower block, which are performed for each configuration and for all indices corresponding to non-zero entries in D_m^I. Hence it is the main contribution to the computational complexity of the algorithm. It is therefore worthwhile to discuss these steps in more detail:

We consider only one iteration. To eliminate indices that complicate the presentation the following notation is used. All variables belonging to workspace W_m^I or density set $\mathcal{D}_m^I$ have names without superscript, while variables belonging to W_{m-1}^I or $\mathcal{D}_{m-1}^I$ have a superscript $'$. Furthermore we restrict the description to one particular configuration $(\mathbf{R}, \mathbf{b}) = (\mathbf{R}_l^{(m)}, \mathbf{b}_l^{(m)}) \in C_m$.

Now we consider one particular block of the input density set $\mathcal{D}_m^I$ specified by the indices (i, j, k). The corresponding entry of the density array is denoted $d = D_m^I(i, j, k)$. The operations to be performed are:
(a) The middle point of block with index (i, j, k) is taken as the refernce for the block. Its coordinates in the workspace are:

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

(b) The homogenous transformation corresponding to configuration $(\mathbf{R}, \mathbf{b})$ is applied to this reference point:

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

(c) The indices (i', j', k') of $\mathbf{x}'$ for the output density set $\mathcal{D}^I_{m-1}$ are:

$$\begin{pmatrix} i'(x') \\ j'(y') \\ k'(z') \end{pmatrix} = \begin{pmatrix} \text{round}\left(i'_0 + (x' - x'_0)/\Delta x'\right) \\ \text{round}\left(j'_0 + (y' - y'_0)/\Delta y'\right) \\ \text{round}\left(k'_0 + (z' - z'_0)/\Delta z'\right) \end{pmatrix}$$

(d) Finally the entry of block (i', j', k') in density set $\mathcal{D}^I_{m-1}$ is incremented by $d = D^I_m(i, j, k)$.

Note, that the entries of the same density array D^I_{m-1} are changed while handling all $l = 1, \ldots, 2^{J_m}$ configurations, without reinitializing. This implements the superposition (or convolution) of all parts of workspace W^I_{m-1}.

Steps (a)-(c) are affine transformations, the last one followed by a rounding procedure. This can be represented as one composite affine transformation followed by rounding which considerably improves the performance of the operation. The described procedure is performed in constant time for each block. Choosing a fixed number of pixels for all intermediate workspaces, has shown good performance. As a result the error grows quadratically in the number of modules B with a very small factor. The amount of required memory is linear in the number of modules as is the time requirement, see [6].

4 Applications and Examples

In this section we present examples of workspaces generated using the workspace mapping algorithm, and show how the algorithm has to be altered to generate work envelopes. In these examples the algorithm is applied to a binary truss manipulator. Each module has the same structure as the modules of the manipulator described in Section 1 and illustrated in Figure 1 and 2. Only the number of modules varies.

Figure 5 (a)-(c) show the results for a binary truss manipulator with 5,8 and 14 platforms respectively, each with actuator strokes stated earlier. Figure 5 (a), which presents the workspace of the manipulator with 5 platforms, can be compared to the exact results shown in Figure 2. Note, that the length scales of the figures differ for different numbers of platforms.

In order to generate the work envelope, we modify the algorithm by choosing a different start workspace than before. Previously a single point was used representing the center of the end-effector. Now we choose a contour that represents the shape of the end-effector. Here we simply use a line representing the upper base of the most distal module. Application of the algorithm sweeps this line (contour) through the plane in discrete steps corresponding to all possible configurations. The algorithm can also be modified to allow smaller steps of the sweeping procedure than given by the discrete configurations. This way a continuous sweeping contour is generated. A

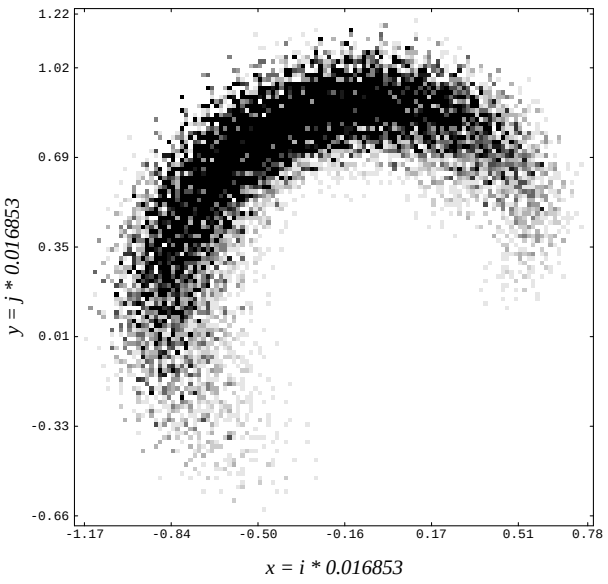


Figure 5(a): Density of a manipulator with 5 modules (15 Bits), $r_5 = 0.018858$

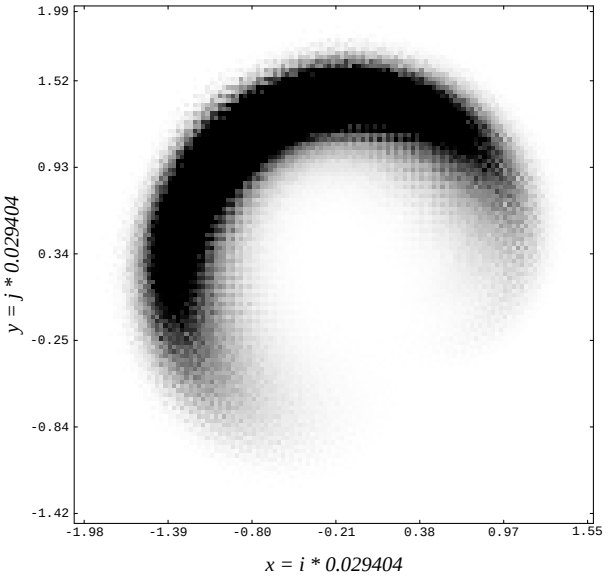


Figure 5(b): Density of a manipulator with 8 modules (24 Bits), $r_8 = 0.041790$

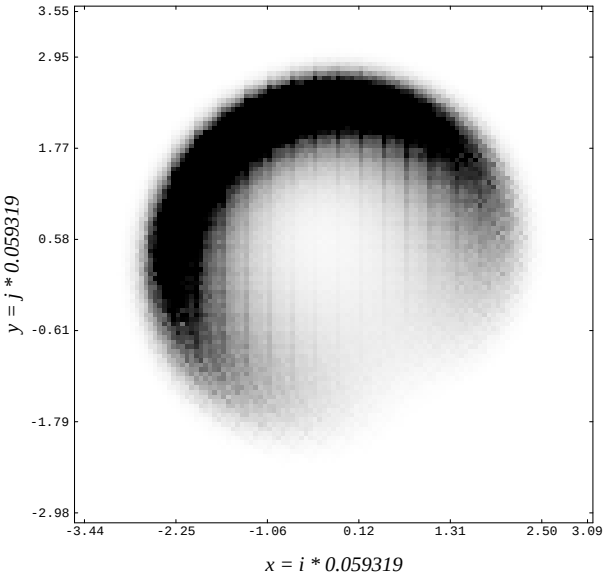


Figure 5(c): Density of a manipulator with 14 modules (42 Bits), $r_{14} = 0.146833$

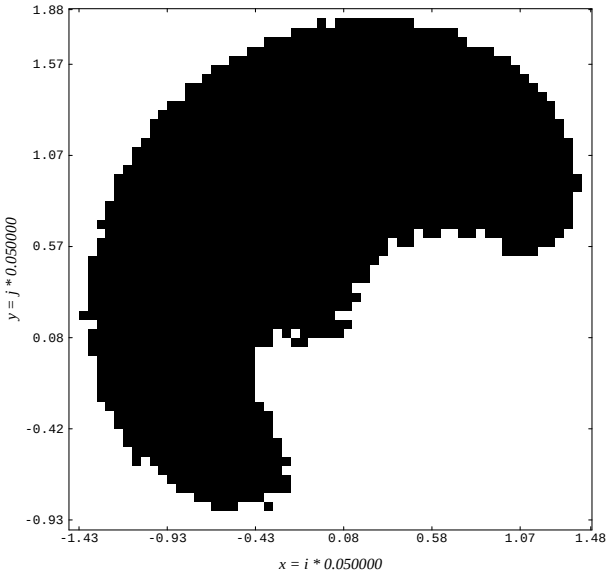


Figure 6: Work envelope of a manipulator with 9 modules (27 Bits)

further simplification is that the number of points for each pixel is not stored, but instead whether or not it is empty (0 or 1).

To calculate the work envelope of a manipulator consisting of m modules, this modified version of the mapping algorithm is applied to the partial manipulators composed of m, $m-1$, ..., 1 modules and the resulting partial workspace are superposed. The result will be a fairly reliable representation of the work envelope. Furthermore, if the modules are all the same, the algorithm has to be applied only once, since $D_m^I = D_m^P$.

Figure 6 shows the work envelope for the case when the joint stops are closer together than in the previous examples, (3/20,4/20), and the width of each platform is chosen as before (4/20).

5 Conclusions

This paper has presented an efficient algorithm for generating approximate workspaces and work envelopes of robotic manipulators with binary (two-state) actuators. Examples were provided for the case of a planar binary variable geometry truss with up to 20 modules. While the focus of this paper was binary manipulators, the method is applicable to general manipulators if their joint range is discretized and represented using an appropriate number of bits.

References

[1] G.S. Chirikjian. A binary paradigm for robotic manipulators. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, San Diego, CA, May 1994.

[2] 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.

[3] R. G. Selfridge. The reachable workarea of a manipulator. *Mech. Mach. Theory*, 18(2):131 – 137, 1983.

[4] Alok Kumar and Kenneth J. Waldron. Numerical plotting of surfaces of positioning accuracy of manipulators. *Mech. Mach. Theory*, 16(4):361 – 368, 1980.

[5] Dibakar Sen and T. S. Mruthyunjaya. A discrete state perspective of manipulator workspaces. *Mech. Mach. Theory*, 29(4):591 – 605, 1994.

[6] I. Ebert-Uphoff and G.S. Chirikjian. Efficient workspace generation for binary manipulators with many actuators. Journal of Robotic Systems, June 1995 (in press).