Course 5: Mechatronics - Foundations and Applications

Binary Manipulator Motion Planning

Vasily Chernonozhkin

May 26, 2006

Abstract

Continuously actuated robotic manipulators are the most common type of manipulators even though they require sophisticated and expensive control and sensor systems to function with high accuracy and repeatability. Binary hyper-redundant robotic manipulators are potential candidates to be used in applications where high repeatability and reasonable accuracy are required. Such applications include pick-and-place, spot welding and assistants to people with disabilities. Generally, the binary manipulator is relatively inexpensive, lightweight, and has a high payload to arm weight ratio. This paper discusses a concept of binary manipulator and its influencing concepts, most known prototypes of binary manipulators are presented here. Several effective kinematic algorithms for binary manipulators are considered.

Contents

1 Introduction

Most robots available today are powered by continuous actuators such as DC motors or hydraulic cylinders. Continuously actuated robots can be built to be precise and to carry large pay loads, but they usually have high price/performance ratios, as evidenced by the high cost of industrial robots available today. Thus, there is a need for a new paradigm in robotics which will lead to lower cost and higher reliability.

Discrete actuators such as solenoids and pneumatic cylinders are relatively inexpensive and simpler than their continuous counterparts. Discretely actuated robots are a promising alternative to traditional robots for certain applications. Using discrete, rather than continuous, actuators to power a robot increases the reliability and lowers the cost of the system even further. Compared to a manipulator built with continuous actuators, a binary manipulator provides reasonable performance, and it's relatively inexpensive.

In this report we'll discuss binary paradigm for robotic manipulators and recent results in binary manipulator motion planning. The remainder of this paper organized as follows: Section 2 formalizes the concept of binary manipulator. Section 3 reviews the literature. Section 4 introduces the most known prototypes of binary manipulators. Section 5 presents recent results in studying binary robots kinematics. Section 6 introduces author's personal work in computer modeling of binary manipulator.

2 Binary Manipulator Concept

Discretely actuated robots have a finite number of states. So, that kind of robots should have a large number of binary actuators for its capabilities to be comparable with continuously actuated robots' capabilities. Thus, only "hyper-redundant" discretely actuated manipulators, or binary manipulators, can be true competitors to manipulators available today.

The word "redundant" is used in the context of robotic manipulators to indicate that the number of actuated degrees of freedom exceeds the minimal number required to perform a particular task. For instance, a manipulator required to position and orient an object in space needs six actuated degrees of freedom, and so a manipulator with seven or more is redundant with respect to this task. "Hyper-redundant" manipulators are manipulators with a very large degree of redundancy.

Hyper-redundant manipulators can be analogous in morphology and operation to "snakes", "elephant trunks", or "tentacles". Because of their highly articulated structures, these robots are well suited for operation in highly constrained environments, and can be designed to have greater robustness with respect to mechanical failure than manipulators with a low degree of redundancy. Furthermore, the concept of hyper-redundancy can be generalized beyond manipulators to describe novel forms of robotic locomotion analogous to the motion of worms, slugs, and snakes.

Binary manipulators are a particular kind of discrete device in which actuators have two stable states. Therefore, they can reach only a discrete (but possibly large because of its hyper-redundancy) number of locations. Major benefits of binary manipulators are:

- they can be operated without extensive feedback control;
- they are relatively inexpensive (up to an order of magnitude cheaper);
- they are relatively lightweight and have a high payload to arm weight ratio;
- their task repeatability is very high.

The characteristics of binary 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 binary 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 laparoscope, 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.

3 Related Literature

The binary manipulator concept is influenced by several successive concepts, such as sensorless systems, discrete actuation and hyper-redundant structure of robotic manipulators.

Particular hyper-redundant designs have previously been referred to as: "highly articulated", "tentacle", "snake-like", "tensor-arm", "elephant trunk", "swan's neck", and "spine" (see [2] for specific references). The word "hyper-redundant" was first used in [3] to capture the essence of these related concepts. To our knowledge, the earliest hyper-redundant robot designs/implementations date to the late 1960s [4]. Hirose and coworkers [5, 6] have implemented a large number of working high-DOF systems. Numerous other authors have suggested hyper-redundant designs or developed hyper-redundant robot mechanisms. Examples include [7, 8, 9]. Many of these designs were driven to some extent by a particular application or operating environment/scenario. Figure 1 exemplifies the three major types of hyper-redundant manipulators: serial (a), continuous (b), and cascaded platforms (c).

The concept of discretely actuated manipulators is quite old in the literature. A planar serial revolute 'digital manipulator' is discussed in Pieper's classic thesis from 1968 [1]. In another classic paper, Roth et al. discuss a three-dimensional (3D) digital manipulator actuated with inflatable airbags [10]. In the mid 1980s, discretely actuated manipulator arms were developed in the former Soviet Union by Koliskor [11]. Closely related to the concept of a discretely actuated manipulator is the idea of sampling a continuous-motion robot at discrete values. This has been done to analyze the error in robotic mechanisms and to generate their workspaces using the Monte Carlo method. See, e.g. [12, 13].

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 [14] have proposed a reduced complexity paradigm for robotic manipulation. There have also been several efforts to develop reliable sensorless manipulation [15, 16]. 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 [17] 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 of sensorless manipulation. Sensorless manipulation reduces the need to sense a robot's environment, while binary actuators allow us to build a

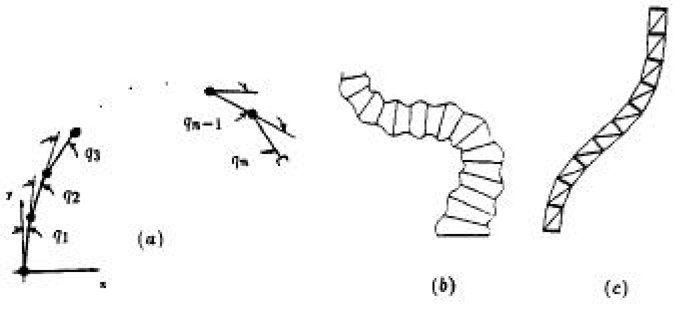


Figure 1: Three major types of hyper-redundant manipulators.

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 [1, 10, 11]. 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 [18]. Also, an effective algorithm has been devised to compute the workspace of hyper-redundant binary robots [19]. Finally, methods have been presented to synthesize a binary manipulator to reach a specific set of points exactly [20], and to make a binary manipulator adhere to a specified curve [21].

4 Examples of Binary Manipulator Realizations

In this section the most known prototypes of binary manipulators are presented.

4.1 Chirikjian et al.

A hyper-redundant robot can be built by stacking variable geometry trusses (VGTs) on top of each other in a long serial chain. This approach yields a structure with good stiffness and load-bearing capabilities at a low cost, compared to traditional non-redundant robots.

The kinematics of hyper-redundant VGT truss manipulators embodies elements of the kinematics of both serial and parallel mechanisms [22, 23, 24, 25]. This is because an individual module of a VGT manipulator is a parallel mechanism, while the complete manipulator, composed of a stack of VGT modules, looks more like a serial structure.

Figure 2 illustrates all possible configurations of a '3 bit' planar binary platform manipulator – one VGT module. A finite number of points are reachable by the manipulator's gripper. 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 are 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 w respectively. In Figure 2, $q_{min} = 1$, $w = 1.2$ and $q_{max} = 1.5$. Thus, when an actuator is in state '1' it is one and a half times its length in state '0'.

A scheme of a highly actuated prototype is shown in Figure 3 for two of its almost thirty three thousand (2^{15}) configurations: 110001110001110 and 001110001110001. 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

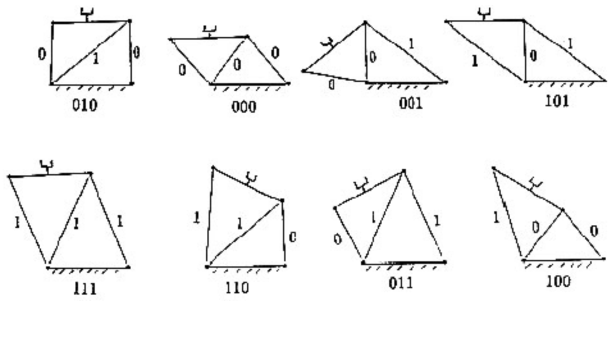


Figure 2: All possible configurations of VGT module.

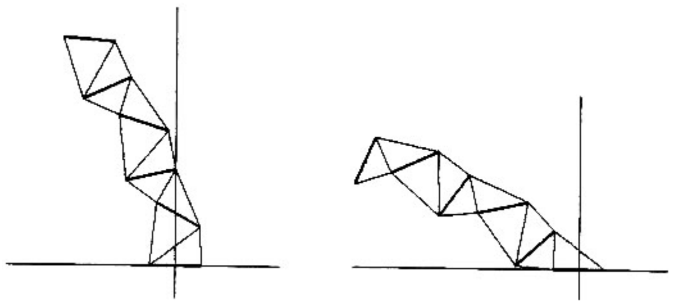


Figure 3: Two configurations of 15-DOF planar VGT manipulator.

'0' or completely extended '1'). In these figures each cylinder has $q_{min} = 3/20$ and $q_{max} = 5/20$ with the width of each platform $w = 1/5$.

Actuators are numbered from left to right in each 'bay' of the truss, and from base to tip. Writing these l's and 0's from left to right, the most significant bit corresponds to the actuator on the left side of the base, and the least significant bit corresponds to the actuator at the right side of the distal end of the manipulator. It is interesting to note that the configurations shown in Figure 3 are the l's compliment of each other.

Since 1994, Chirikjian and his co-workers implemented several binary manipulators based on VGT structure, developed in the Robot and Protein Kinematics Lab at Johns Hopkins University.

4.1.1 Planar Binary Robot Manipulator

This manipulator consists of five 3-bit planar VGT modules (see Fig. 4). Pneumatic cylinders are used as actuators because of their low cost, lightweight, and sufficient force. This manipulator is designed to manipulate objects in two dimensions only. One end of the manipulator is attached to a base, while the end-effector has a two-state gripper. The total number of actuators (bits) is 3×5 (=15), which provides 2^{15} (=32768) reachable positions in 2-dimensional space. The manipulator is controlled by the user who inputs a binary number (0 or 1) for each individual actuator.

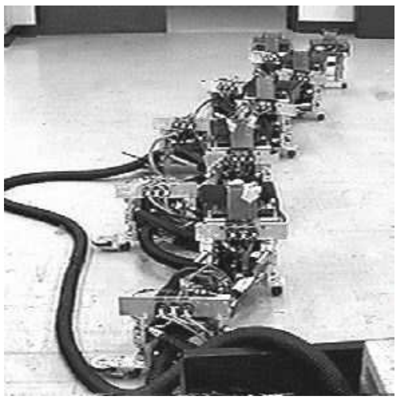


Figure 4: Planar 30-DOF binary robot manipulator by Chirikjian and Burdick.

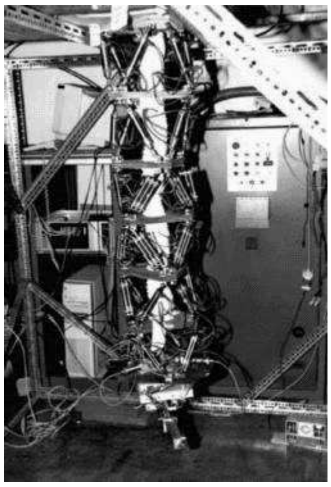


Figure 5: Ebert-Uphoff's binary robot manipulator.

4.1.2 Ebert-Uphoff Binary Robot Manipulator

This is the 3-dimensional binary robot manipulator (see Fig. 5) influenced by the Stewart/Gough platform actuated with pneumatic cylinders. The manipulator consists of 6 modules, and one end is vertically attached to the structure from the top (ceiling-liked). A 3-D gripper (X, Y, and theta) is attached at the end of end-effector. Each module consists of 6 binary actuators. Thus, the end-effector can reach a total of $2^{6\times6}$ ($\sim$ 68.7 billion) different positions in 3-D space.

4.1.3 Suthakorn Discretely-Actuated Hyper-Redundant Robotic Manipulator

The design of this robotic manipulator uses 3-bit binary VGT modules stacked on top of each other with a discretely actuated rotating joint between each module (see Fig. 6). As a result the manipulator has the ability to reach many points and covers a full 3-dimensional sphere around the manipulator itself. The prototype consists of three modules of 3-bit binary VGTs, and each rotating joint between each module has 16 steps. This configuration makes the prototype have $2^{3\times3} \times 16^3$ ($\sim$ 2.1 million) discrete states.

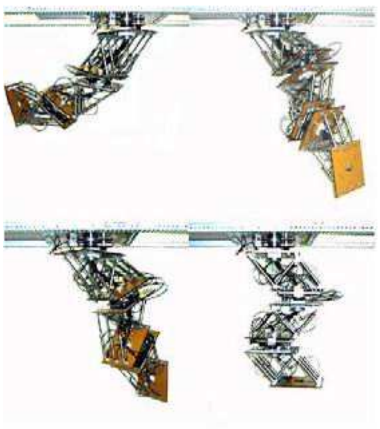


Figure 6: Suthakorn's discretely-actuated hyper-redundant robotic manipulator.

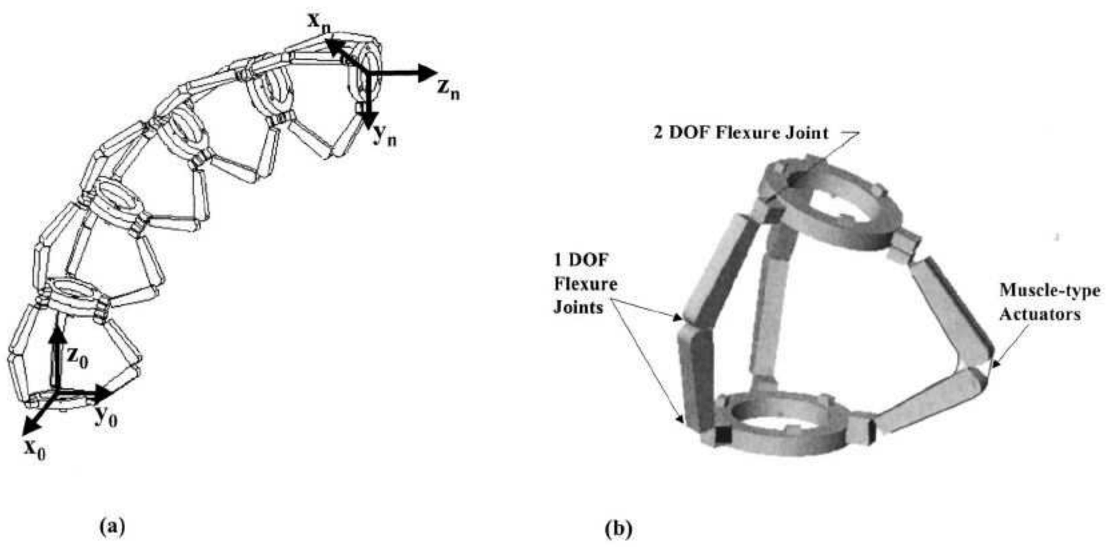


Figure 7: BRAID's i^{th} parallel link stage.

4.2 Dubowsky et al.

In 2002 Dubowsky and co-workers presented device, called a Binary Robotic Articulated Intelligent Device (BRAID) [26, 27]. It consists of compliant mechanisms with large numbers of embedded actuators and is a step toward practical implementation of binary devices for space robotic systems.

The BRAID element is made of a serial chain of parallel stages (see Fig. 7(a)). Each three DOF stage has three flexure-based legs, each with muscle type binary actuators. In the experimental system these were shape memory alloy (SMA) actuators, but more promising polymer actuators were then being implemented. Muscle actuation allows binary operation of each leg. The flexures are simple and light weight. The experimental BRAID built consists of five parallel stages, yielding 15 binary degrees of freedom. Thus it has 2^{15} (32768) discrete configurations.

Figure 7(b) shows one stage of the BRAID element. Each parallel link stage has three legs. Each leg has three flexure joints - two one DOF joints and one three DOF joint. These results in five axes per leg: three in parallel, the fourth orthogonal to the first three and the fifth orthogonal to the fourth. Coupling the three legs together (symmetrically 120° apart) gives the parallel link stage three DOF mobility (vertical translation, pitch, and yaw) controlled by actuators placed on each leg. However, in the physical implementation of the design (see Fig. 7(b)) the fifth DOF in each leg was removed, since this motion is small and can be accommodated by elastic deflections.

Actuation of each leg is accomplished using a muscle-type two-state (or binary) actuator such as an SMA. If each leg has only one actuator (an SMA wire in tension) the restoration force to change the binary states is provided by the elastic flexure joints. On the other hand, if a pair of antagonistic actuators is used, the elastic restoration force is not needed. Detents help lock each binary leg into a discrete state (see Fig. 8) and provide more accurate and repeatable positioning [28, 29]. They also eliminate the need for power while the BRAID is stationary.

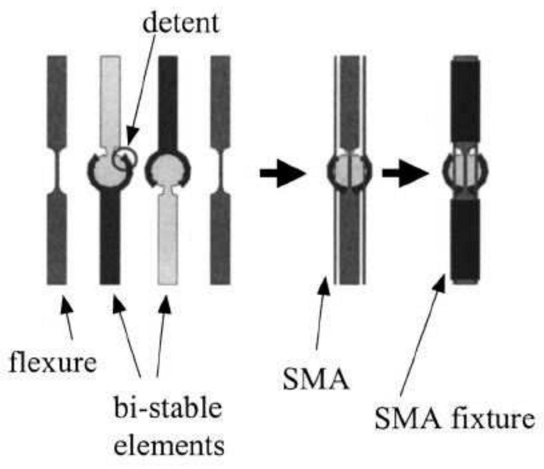


Figure 8: Detent based binary joint.

5 Efforts in Binary Manipulator Motion Planning

While the hardware costs of a binary manipulator are lower than of a continuously actuated manipulator, there is a tradeoff in the complexity of the trajectory planning software. 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.

Since the mid 1990s Chirikjian and coworkers have developed a variety of efficient algorithms for highly actuated discrete-state robots and mechanisms. These include approaches to the kinematic synthesis of such mechanisms [20, 22, 24, 30], the generation of workspaces [19, 31] and inverse kinematics [21, 32, 33, 34, 35, 36, 37].In this work we present some of this algorithms.

5.1 An Efficient Algorithm for Computing the Forward Kinematics

Now we'll discuss an efficient method for computing the forward kinematics of binary manipulators, that allows 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. 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.

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.

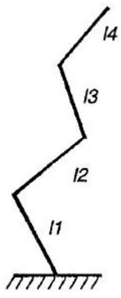


Figure 9: Serial-revolute manipulator.

5.1.1 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 $j = 1, \dots, J$.
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 $i = 1, \dots, B$.

Module kinematics:

For a serial-revolute manipulator (Fig. 9) the kinematics of an individual module in a particular state are described by:

$$b_i = (l_i \cos\theta_i, l_i \sin\theta_i)\,;\ \theta_i = \begin{cases} q_i^{\min}\ if\ s_i = 0 \\ q_i^{\max}\ if\ s_i = 1 \end{cases}. \tag{1}$$

The forward kinematics for a single VGT module obey the following geometric constraints (refer to Fig. 10):

$$x_1^2 + y_1^2 = q_1^2; \tag{2}$$

$$(x_1 - w)^2 + y_1^2 = q_2^2; \tag{3}$$

$$x_0^2 + y_0^2 = q_0^2; \tag{4}$$

$$(x_0 - x_1)^2 + (y_0 - y_1)^2 = w^2. \tag{5}$$

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

$$x_1 = -\frac{q_2^2 - q_1^2 - w^2}{2w};\ y_1 = \sqrt{q_1^2 - \left(\frac{q_2^2 - q_1^2 - w^2}{2w}\right)^2}, \tag{6}$$

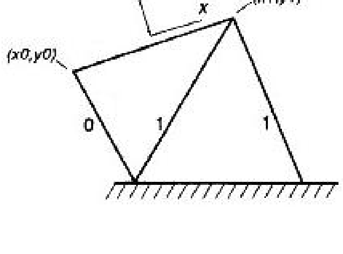


Figure 10: VGT module.

and

$$x_0 = \frac{-b - \sqrt{b^2 - 4ac}}{2a};\ y_0 = \sqrt{q_0^2 - x_0^2}, \tag{7}$$

where:

$$a = 4x_1^2 + 4y_1^2; \tag{8}$$

$$b = 4(w^2 x_1 - q_0^2 x_1 - q_1^2 x_1); \tag{9}$$

$$c = q_1^4 + q_0^4 + 2q_0^2 q_1^2 - 2q_0^2 w^2 - 2q_1^2 w^2 + w^4 - 4y_1^2 q_0^2. \tag{10}$$

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

$$b_i = \left(\frac{(x_{0i} + x_{1i})}{2}, \frac{(y_{0i} + y_{1i})}{2}\right);\ \theta_i = arctg\left(\frac{y_{1i} - y_{0i}}{x_{1i} - x_{0i}}\right). \tag{11}$$

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 $J_i$
        Use the kinematic parameters of module $i$ to compute $C_i$ for state $j$.
    end
end
```

5.1.2 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. EE_{pos}, the position of the manipulator's end effector.
2. s_{ee}, c_{ee}, the sin and cos of the end-effector orientation angle for the two-dimensional case, or R_{ee}, the rotation matrix describing end-effector orientation, in the three-dimensional case.

Main algorithm:

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

```
$EE_{pos} = 0$
$R_{ee} = I$, the identity matrix.
for $i = 1$ to $B$
    select $C_i$, the rotations and position vectors for module $i$.
            $EE_{pos} = EE_{pos} + R_{s_i} b_{s_i}$
            $R_{ee} = R_{s_i} R_{ee}$
    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 $\mathcal{O}(B)$ storage, while a "traditional" algorithm requires only constant storage. The main body of the algorithm requires $\mathcal{O}(B)$ 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 in the algorithm presented here give it a very important practical advantage. For one computer architecture it executed ten times faster than the standard approach.

5.2 A Combinatorial Method for Computing the Inverse Kinematics

Secondly we'll consider an efficient combinatorial method for computing the inverse kinematics and planning the trajectory of binary manipulators. It searches for a solution by changing only a small number of the manipulator's actuators at any given time. This approach reduces the size of the search space considerably, and because only a small number of actuators change state, it produces very smooth robot motions.

The idea behind this inverse kinematics algorithm is to find a set, or sets, of actuator states that cause the manipulator to reach a certain location in space, and also optimize the distance between the end-effector and a desired location.

To avoid exponential growth in the search space as the number of actuators grows, the inverse kinematics is solved incrementally by changing only a small number of actuators at a time. For example, in the 10-module truss with 30 DOF, we might try to minimize the error between the end-effector and the goal, by changing only three of the actuators at one time. In this case, we need only search $\begin{pmatrix} 30 \\ 3 \end{pmatrix}$, or 4060, possible solutions, instead of the 2^{30} we would have had to explore if we searched all possible configurations of the manipulator.

5.2.1 Searching for Robot Configurations

Consider the standard definition of the binomial theorem [38]:

$$(x+y)^n = \sum_{i=0}^{n} \begin{pmatrix} n \\ i \end{pmatrix} x^i y^{n-i}. \tag{12}$$

If we let $x = 1$ and let $y = 1$, we get the following result:

$$(1+1)^n = \sum_{i=0}^{n} \begin{pmatrix} n \\ i \end{pmatrix} 1^i 1^{n-i}; \tag{13}$$

$$2^n = \sum_{i=0}^{n} \begin{pmatrix} n \\ i \end{pmatrix}. \tag{14}$$

Therefore, if we have a 10-module truss robot, for example, we can search its entire state set by taking its current state and looking at all zero bit changes from that state, then all one bit changes, etc., until we have considered all 2^n states. Obviously, if we take this approach to its logical conclusion, it is no better than searching all 2^n states in numerical order. However, if we are willing to move the robot toward its goal by searching for changes in only a small number of bits at any given time, we can obtain a substantial performance gain – to the point where we have a practical algorithm, even for robots with many degrees of freedom.

5.2.2 Complexity of the Algorithm

While a brute-force search of a binary manipulator's workspace requires computational effort of $\mathcal{O}(2^n)$, the combinatorial algorithm can be executed in polynomial time, if we fix the number of bit changes that we search, regardless of the number of DOF in the robot. Consider a VGT robot with J actuators, which we move toward its target location by changing no more than k of its actuators at a time. To do this we must search through:

$$\begin{pmatrix} J \\ 0 \end{pmatrix} + \begin{pmatrix} J \\ 1 \end{pmatrix} + \begin{pmatrix} J \\ 2 \end{pmatrix} + ... + \begin{pmatrix} J \\ k \end{pmatrix} \tag{15}$$

candidate states to find the one that best moves the robot toward its target position. Note that the operation $\begin{pmatrix} n \\ k \end{pmatrix}$ is defined as follows:

$$\begin{pmatrix} n \\ k \end{pmatrix} = \frac{n!}{k!(n-k)!} = \frac{1}{k!}\frac{n!}{(n-k)!}. \tag{16}$$

We can expand this equation into:

$$\begin{pmatrix} n \\ k \end{pmatrix} = \frac{1}{k!}\frac{n(n-1)...(n-k)...1}{(n-k)(n-k-1)...1} = \frac{1}{k!}n(n-1)(n-2)...(n-k+1). \tag{17}$$

This equation has k terms involving n. Therefore, $\mathcal{O}(n^k)$ time is required to enumerate all the combinations in $\begin{pmatrix} n \\ k \end{pmatrix}$, for all n, with k fixed.

An interesting implication of the final equation is that a search for a state change of a single actuator (i.e. one bit) in a VGT truss robot can be accomplished in linear time. Changing the position of only a single actuator is unlikely to make the robot reach its goal position, but iterating the search several times can make the end-effector approach its target with more and more accuracy.

5.2.3 Smoothness of Motion

It can be difficult to make a binary manipulator follow a smooth trajectory, because the path that it follows between any two discrete states cannot be specified precisely. The combinatorial algorithm addresses this problem in two ways. First, because it explicitly controls the number of bits that can change at one time, we can limit changes to only a small number of bits, which reduces the overall change in the manipulator's configuration. Second, we can try to minimize our position error by examining changes to the least significant bits of the manipulator's state first (i.e. the top of the manipulator), which gives preference to relatively small motions of the manipulator. To implement this behavior we must generate the state combinations in lexicographic order, from least to most significant. The lexicographic ordering algorithm is described in [39].

5.2.4 Algorithm Description

Inputs to the algorithm:

1. EE_{des}, the desired position of the manipulator's end-effector.
2. EE_{now}, the current position of the manipulator's end-effector.
3. p_{now}, the manipulator's current state vector.
4. n_{max}, the maximum number of bits allowed to change in the manipulator's state vector.
5. B, the number of degrees of freedom (same as number of bits) of the manipulator.

6. The geometry of the manipulator modules for computing the forward kinematics (using, for example, the method describer in 5.1).

Implementation:

```
/* $d_{min}$ is the distance from the old to new location. */
$d_{min} = cost(EE_{des}, EE_{now})$
/* $p_{min}$ is the closest state vector to the desired position. */
$p_{min} = p_{now}$
/* $b_{min}$ is the number of bits we changed to get to $p_{min}$. */
$b_{min} = 0$
for $i = 1$ to $n_{max}$
    for $j = 1$ to $\binom{B}{i}$
        /* Get combo $j$ in lexicographic order. */
        $c = combo(B, i, j)$
        $p_{test} = c \oplus p_{now}$
        /* $fwdKin(x)$: foward kinematics for state $x$. */
        $d_{test} = cost(EE_{des}, fwdKin(p_{test}))$
        if $d_{test} < d_{min}$ then
            $d_{test} = d_{min}$
            $p_{test} = p_{min}$
            $b_{min} = i$
        end
    end
end
return
```

5.3 Efficient Workspace Generation

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 subsection 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.

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 represented by something called a density array. The latter is a computer representation of the number of end-effector points for each pixel/voxel of the workspace.

5.3.1 Concepts for Discrete Workspaces

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 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:

$$\rho(block) = \frac{\#\text{ binary manipulator states resulting in ee} - \text{position within block}}{\text{volume/area of workspace block}}. \tag{18}$$

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

$$\rho(block) = \frac{\#\text{ of reachable points within block}}{\text{volume/area of block}}, \tag{19}$$

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.

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 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.

An affine transformation in $\mathbb{R}^N$ is a transformation of the form $\mathbf{y} = \mathbf{A}\mathbf{x}+\mathbf{b}$, 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: $\mathbf{y} = \mathbf{R}\mathbf{x} + \mathbf{b}$, where $\mathbf{R}$ is a special orthogonal matrix, i.e., an orthogonal matrix with determinant 1.

5.3.2 Efficient Representation of Workspaces

For this 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.

There are several requirements for potential workspace representations:

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 $k \leqslant n$.
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.

It is decided to use the point density array defined in subsection 5.3.1 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:

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.

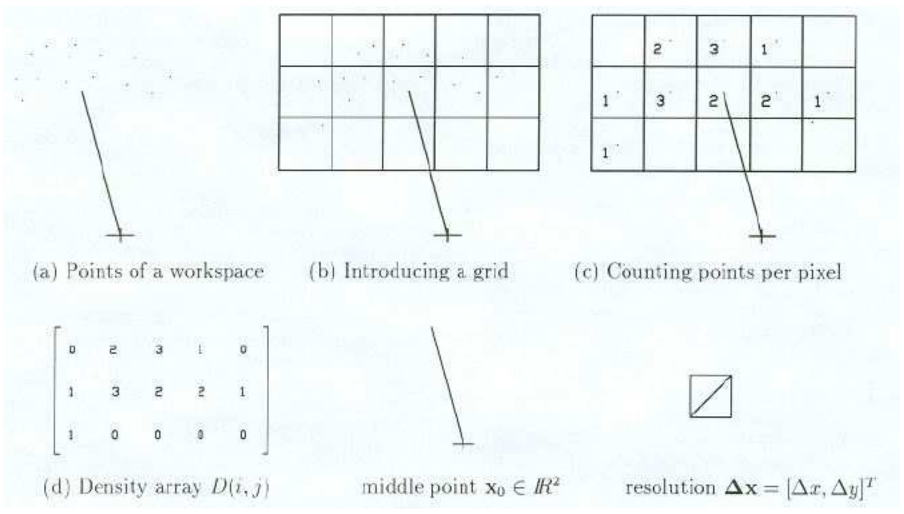

(a) Points of a workspace
(b) Introducing a grid
(c) Counting points per pixel
(d) Density array $D(i,j)$
middle point $\mathbf{x_0} \in I\!R^2$
resolution $\mathbf{\Delta x} = [\Delta x, \Delta y]^T$

Figure 11: Representation of a workspace as a density set.

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

An example of the description of a workspace as a density set is given in 11 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 given workspace coordinates $\mathbf{x} = [x, y, z]^T$, the corresponding array indices (i, j, k) can be find as follows:

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

$$\begin{aligned} i &= 0, 1, ..., i_0, i_0 + 1, ..., 2i_0, \\ j &= 0, 1, ..., j_0, j_0 + 1, ..., 2j_0, \\ k &= 0, 1, ..., k_0, k_0 + 1, ..., 2k_0. \end{aligned} \tag{20}$$

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:

$$\begin{aligned} x(i,j,k) &= x(i) = x_0 + \Delta x(i - i_0), \\ y(i,j,k) &= y(j) = y_0 + \Delta y(j - j_0), \\ z(i,j,k) &= z(k) = z_0 + \Delta z(k - k_0). \end{aligned} \tag{21}$$

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 $\lfloor x \rfloor$ is defined to be the largest integer that is smaller or equal to x. The following relationship holds:

$$Round(x) = \lfloor x + 0.5 \rfloor, \tag{22}$$

so that the inverse problem is solved as follows:

$$\begin{aligned} i(x,y,z) = i(x) &= \left\lfloor \tfrac{(x-x_0)}{\Delta x} + 0.5 \right\rfloor + i_0, \\ j(x,y,z) = j(y) &= \left\lfloor \tfrac{(y-y_0)}{\Delta y} + 0.5 \right\rfloor + j_0, \\ k(x,y,z) = k(z) &= \left\lfloor \tfrac{(z-z_0)}{\Delta z} + 0.5 \right\rfloor + k_0. \end{aligned} \tag{23}$$

5.3.3 The Workspace Mapping Algorithm

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

- index s denotes the s^{th} iteration of the mapping algorithm, $(s = 1, 2, \dots, B)$,
- index m denotes the m^{th} module considered in the s^{th} step, $(m = B, B-1, \dots, 1)$.

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: $C_m = \{(\mathbf{R_1}^{(m)}, \mathbf{b_1}^{(m)}), (\mathbf{R_2}^{(m)}, \mathbf{b_2}^{(m)}), \dots, (\mathbf{R_{2^{J_m}}}^{(m)}, \mathbf{b_{2^{J_m}}}^{(m)})\}$. One iteration of workspace mapping determines the density set $\mathcal{D}_{m-1}$ (representing the point density of workspace W_{m-1}) from given point density $\mathcal{D}_m$ and configuration set C_m.

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 $s = 1, 2, \dots, B$. 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 $(\mathbf{R_l}^{(m)}, \mathbf{b_l}^{(m)}) \in C_m$, $(l = 1, \dots, 2^{J_m})$, 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 $\mathbf{x} = [x(i), y(i), z(i)]^T$ from the array indices (i, j, k).
   (b) Calculate the coordinate vector $\mathbf{x'} = \mathbf{R_l}^{(m)}\mathbf{x} + \mathbf{b_l}^{(m)} \in W_{m-1}$.
   (c) Find the array indices (i', j', k') of $\mathbf{x'}$ in the new array.
   (d) Increment entries in the block of the new array by the corresponding entry of the old array: $D_{m-1}(i', j', k') \leftarrow (D_{m-1}(i', j', k') + D_m(i, j, k))$.

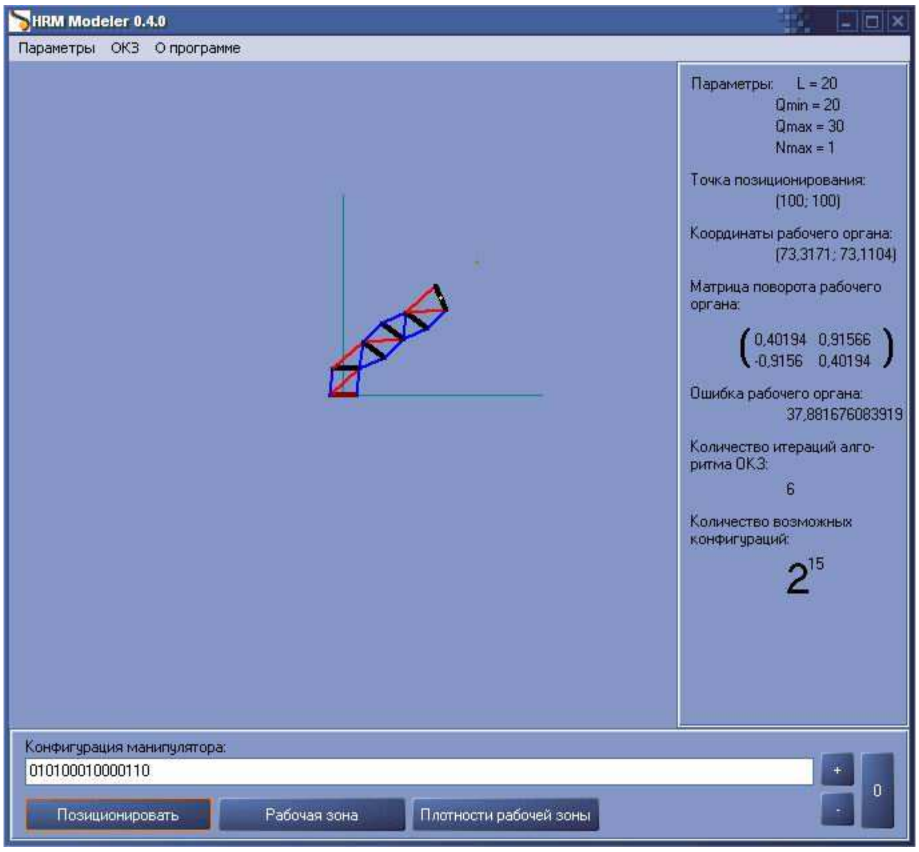
To estimate size and location of workspace W_{m-1}, all 2^{J_m} homogeneous transforms are applied to the eight comers of the density array D_m (four for the planar case). The resulting maximal and minimal values in each coordinate axis are taken as the boundaries of the next density array, D_{m-1}. 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.

There are several works based on the algorithm for efficient workspace generation using workspace densities [31, 36], including a recent extremely fast and most efficient algorithm for the inverse kinematics of binary manipulators [37]. This algorithm are planned to be studied and implemented in near future.

6 HRM Modeler

In 2005 the computer program "HRM Modeler" was written by the author while working towards the Bachelor's Degree in St.-Petersburg State Polytechnic University. This software models binary manipulator kinematics. Planar VGT-based manipulator (see 4.1.1) as modeling object, and algorithms for forward kinematics and inverse kinematics, described in 5.1 and 5.2, was chosen for this purpose.

To date, efficient algorithm for workspace generation, described in 5.3, is implemented in modeling program. Presently the latest and most efficient algorithm for inverse kinematics is studying and implementing.



Параметры: L = 20
Qmin = 20
Qmax = 30
Nmax = 1

Точка позиционирования:
(100; 100)

Координаты рабочего органа:
(73,3171; 73,1104)

Матрица поворота рабочего органа:

$$\begin{pmatrix} 0,40194 & 0,91566 \\ -0,9156 & 0,40194 \end{pmatrix}$$

Ошибка рабочего органа:
37,881676083919

Количество итераций алгоритма ОКЗ:
6

Количество возможных конфигураций:
2^{15}

Конфигурация манипулятора:
010100010000110

Figure 12: The main window of the modeling program.

Figure 12 shows the main window of the program. "HRM Modeler" was written in Object Pascal programming language in programming environment Delphi 7. Its main features are:

- VGT structure graphic modeling;
- end-effector coordinates and orientation calculation;
- manipulator configuration, that allows to reach any given point, computing;
- position error minimization by reducing the distance between end-effector and given point;
- manipulator workspace computing and graphic representation.

In Figure 13 outcome of a combinatorial algorithm for the inverse kinematics, executed in several iterations for reducing the position error, are shown. In Figure 14(A) manipulator workspace generated with algorithm, described in 5.3, compared with workspace obtained with simple enumeration of all manipulator configurations (B) is shown.

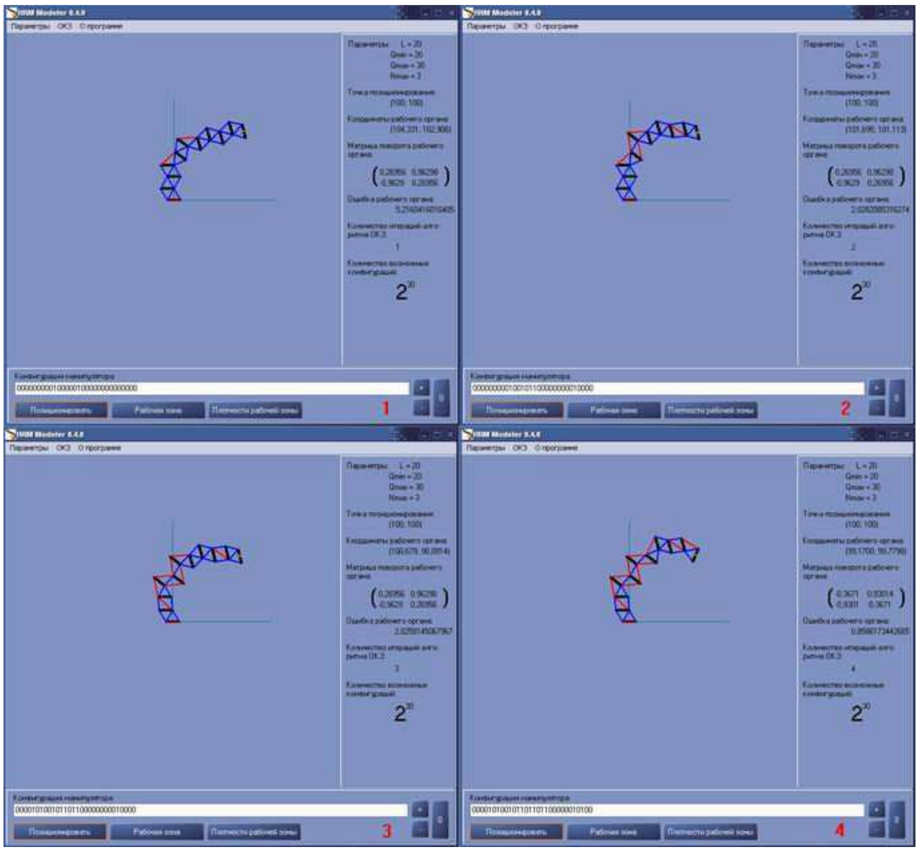


Figure 13: The outcome of a combinatorial algorithm for the IK of 30-DOF VGT manipulator.

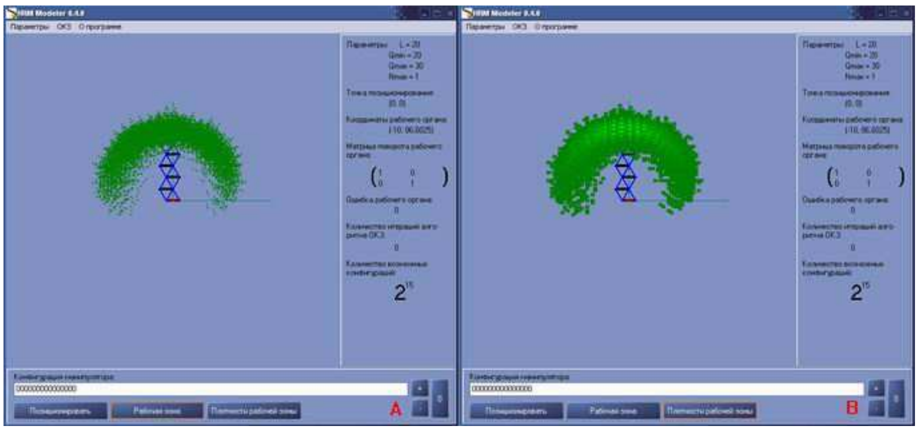


Figure 14: The 15-DOF VGT manipulator workspace.

7 Conclusions

In this report we've discussed several basic concepts of the binary manipulator, e.g., sensorless systems, discretely actuated and hyper-redundant manipulators. Some descriptions with examples of binary hyper-redundant manipulators were introduced. Previous efforts on the concept and binary manipulator kinematics and motion planning were described. In conclusion we're presenting one more time a list of advantages and disadvantages of binary manipulators manipulator, and examples of its applications.

As it was shown at the beginning of this report, binary manipulators are promising alternative to traditional continuously actuated robots. Such advantages of binary robots, like low cost, light weight, high task repeatability and other, makes efforts in studying of binary manipulators very perspective and practical. Moreover, while the latest algorithm is extremely fast and efficient, there is some limitation in number of binary manipulator modules. When this limitation is exceeded high computational costs become an insoluble problem again. Thus, there is still a problem of finding new decisions, making new algorithms in binary manipulator motion planning.

References

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

[2] Chirikjian G.S. *Theory and Applications of Hyper-Redundant Robotic Manipulators.* PhD thesis, Department of Applied Mechanics, Division of Engineering and Applied Science, California Institute of Technology, June 1992.

[3] Chirikjian G.S. Burdick J.W. An obstacle avoidance algorithm for hyper-redundant manipulators. In *IEEE International Conference on Robotics and Automation*, pages 625–631, Cincinnati, OH, May 1990.

[4] Anderson V.V. Horn R.C. Tensor-arm manipulator design. *ASME Transactions*, 67-DE-57:1–12, 1967.

[5] Hirose S. Umetani Y. Kinematic control of active cord mechanism with tactile sensors. In *2nd International CISM-IFTMM Symposium on Theory and Practice of Robots and Manipulators*, pages 241–252, 1976.

[6] Hirose S. Yokoshima K. Ma S. 2 dof moray drive for hyper-redundant manipulator. In *IROS*, pages 1735–1740, Raleigh, NC, July 1992.

[7] Kobayashi H. Shimemura E. Suzuki K. A distributed control for hyper-redundant manipulator. In *IROS*, pages 1958–1963, Raleigh, NC, July 1992.

[8] Shahinpoor M. Kalhor H. Jamshidi M. On magnetically activated robotic tensor arms. In *International Symposium on Robot Manipulators: Modeling, Control, and Education*, Albuquerque, New Mexico, November 1983.

[9] Wilson J.F. Mahajan U. The mechanics and positioning of highly flexible manipulator limbs. *Journal of Mechanisms, Transmissions, and Automation in Design*, 111, 1989.

[10] Roth B. Rastegar J. Scheinman V. On the design of computer controlled manipulators. In *1st International CISM-IFTMM Symposium on Theory and Practice of Robots and Manipulators*, pages 93–113, 1973.

[11] Koliskor A. The 1-coordinate approach to the industrial robots design. In *5th IFAC/IFIP/IMACS/IFORS Conference*, pages 225–232, Suzdal, USSR, 1986.

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

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

[14] Canny J. Goldberg K. A rise paradigm for industrial robotics. Technical Report ESRC 93-4/RAMP 93-2, Engineering Systems Research Center, University of California at Berkeley, 1993.

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

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

[17] Erdmann M.A. Mason M.T. Exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation*, 14:369–379, August 1988.

[18] Bergstrom P.L. Tamagawa T. Polla D.L. Design and fabrication of micromechanical logic elements. In *IEEE Micro Electro Mechanical Systems Workshop*, pages 15–20, Napa, CA, February 1990.

[19] Ebert-Uphoff I. Chirikjian G.S. Efficient workspace generation for binary manipulators with many actuators. *Journal of Robotic Systems*, 12:383–400, June 1995.

[20] Chirikjian G.S. Synthesis of mechanisms and robotic manipulators with binary actuators. *ASME Journal of Mechanical Design*, 117:573–580, 1995.

[21] Lees D. Chirikjian G.S. Inverse kinematics of binary manipulators with applications to service robotics. In *IROS*, pages 65–71, Pittsburgh, PA, August 1995.

[22] Chirikjian G.S. A binary paradigm for robotic manipulators. In *IEEE International Conference on Robotics and Automation*, pages 3063–3069, San Diego, CA, 1994.

[23] Hughes P.C. Trussarm – a variable-geometry-truss manipulator. *Journal of Intelligent Materials, Systems and Structures*, 2:148–160, April 1991.

[24] Ebert-Uphoff I. *On the Development of Discretely-Actuated Hybrid-Serial-Parallel Manipulators.* PhD thesis, Johns Hopkins University, 1997.

[25] Suthakorn J. Binary hyper-redundant robotic manipulator concept. 2004.

[26] Wingert A. Lichter M. Dubowsky S. Hafez M. Hyper-redundant robot manipulators actuated by optimized binary dielectric polymers. In *SPIE vol. 4695 (Electroactive Polymer Actuators and Devices) from Smart Structures and Materials Symposium*, San Diego, CA, March 2002.

[27] Sujan V.A. Dubowsky S. Design of a lightweight hyper-redundant deployable binary manipulator. *ASME Journal of Mechanical Design*, 126:29–39, January 2004.

[28] Lichter M.D. Sujan V.A. Dubowsky S. Experimental demonstration of a new design paradigm in space robotics. In *7th International Symposium on Experimental Robotics (ISER)*, pages 10–13, Honolulu, Hawaii, December 2000.

[29] Sujan V.A. Lichter M.D. Dubowsky S. Lightweight hyper-redundant binary elements for planetary exploration robots. In *IEEE/ASME Conference on Advanced Intelligent Mechatronics (AIM)*, Como, Italy, July 2001.

[30] Kyatkinand A.B. Chirikjian G.S. Synthesis of binary manipulators using the fourier transform on the euclidean group. *ASME Journal of Mechanical Design*, 121:9–14, 1999.

[31] Chirikjian G.S. Ebert-Uphoff I. Numerical convolution on the euclidean group with applications to workspace generation. *IEEE Transactions on Robotics and Automation*, 14(1):123–136, 1998.

[32] Lees D.S. Chirikjian G.S. An efficient trajectory planning method for binary manipulators. In *ASME Mechanisms Conference, 96-DETC/MECH-1*, volume 161, 1996.

[33] Lees D.S. Chirikjian G.S. An efficient method for computing the forward kinematics of binary manipulators. In *IEEE International Conference on Robotics and Automation*, pages 1012–1017, Minneapolis, MN, 1996.

[34] Lees D.S. Chirikjian G.S. A combinatorial approach to trajectory planning for binary manipulators. In *IEEE International Conference on Robotics and Automation*, pages 2749–2754, Minneapolis, MN, 1996.

[35] Chirikjian G.S. Inverse kinematics of binary manipulators using a continuum model. *Journal of Intelligent Robotic Systems*, 19:5–22, 1997.

[36] Ebert-Uphoff I. Chirikjian G.S. Inverse kinematics of discretely actuated hyper-redundant manipulators using workspace densities. In *IEEE International Conference on Robotics and Automation*, pages 139–145, 1996.

[37] Suthakorn J. Chirikjian G.S. A new inverse kinematics algorithm for binary manipulators with many actuators. *Advanced Robotics*, 15(2):225–244, 2001.

[38] Ross S. *A First Course in Probability.* MacMillan, New York, NY, 1984.

[39] Jackson B.W. Thoro D. *Applied Combinatorics with Problem Solving.* Addison-Wesley, Reading, MA, 1990.