Engineering Applications of Artificial Intelligence 64 (2017) 197–207

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

An improved Monte Carlo method based on Gaussian growth to calculate the workspace of robots

Adrián Peidró *, Óscar Reinoso, Arturo Gil, José María Marín, Luis Payá

Systems Engineering and Automation Department, Miguel Hernández University, Avda. de la Universidad s/n, 03202 Elche, Spain

A R T I C L E I N F O

Keywords:
Gaussian distribution
Monte Carlo method
Robot manipulator
Workspace

A B S T R A C T

This paper presents a new Monte Carlo method to calculate the workspace of robot manipulators, which we called the Gaussian Growth method. In contrast to classical brute-force Monte Carlo methods, which rely on increasing the number of randomly generated points in the whole workspace to attain higher accuracy, the Gaussian Growth method focuses on populating and improving the precision of poorly defined regions of the workspace. For this purpose, the proposed method first generates an inaccurate seed workspace using a classical Monte Carlo method, and then it uses the Gaussian distribution to densify and grow this seed workspace until the boundaries of the workspace are attained. The proposed method is compared with previous Monte Carlo methods using a 10-degrees-of-freedom robot as a case study, and it is demonstrated that the Gaussian Growth method can generate more accurate workspaces than previous methods requiring the same or less computation time.

1. Introduction

The workspace of a robot manipulator can be defined as a set of positions and/or orientations that can be attained by its end-effector, which is the body of the robot that usually interacts with the environment. Studying the workspace is very important for designing the robot and for planning its movements. Therefore, the calculation of the workspace has been the focus of many research studies during the last three decades. Most methods for calculating the workspace can be classified into one of three main classes (Merlet, 2006): geometrical methods, methods based on the Jacobian matrix, and discretization methods.

Geometrical methods are mainly used for parallel robots, in which the motion of the end-effector is controlled by two or more legs working in parallel. These methods consist in calculating first the individual workspaces of the different legs independently, whose workspaces are relatively simple geometrical objects such as annuli (Merlet et al., 1998), spherical shells (Gosselin, 1990), or solid tori (Liu and Wang, 2014). Then, the workspace of the complete robot is obtained as the intersection of the individual workspaces of all legs, using Computer Aided Design tools. Geometrical methods are very accurate and fast, but they are tailored to each specific robot, which limits their scope. Moreover, it is difficult to impose some restrictions when applying these methods, such as the existence of mechanical limits on the joints or the avoidance of collisions between different parts of the robot, although in some cases these restrictions can be included in this method (Merlet, 1995).

Jacobian-based methods, which are more general, focus on obtaining directly the boundaries that delimit the workspace. These boundaries can be obtained through imposing the rank deficiency of the Jacobian matrix constituted by the derivatives of all the constraints of the robot with respect to all the variables, excluding those variables that define the position and orientation of the end-effector. When imposing the rank deficiency of such a Jacobian matrix, one obtains a set of equations that can be solved to obtain the boundaries of the workspace. In some cases, it is possible to solve analytically these equations, obtaining analytical descriptions of the workspace boundaries (Abdel-Malek and Yang, 2006; Abdel-Malek et al., 2000). If this is not possible, they can be solved using numerical methods (Haug et al., 1995; Bohigas et al., 2012). A drawback of Jacobian-based methods is that all the constraints must be written as equalities, which may result in quite large systems of equations. Moreover, although some inequality constraints (like joint limits) may be easily transformed into equalities, other constraints (like avoidance of self-interferences) are difficult or impossible to model as equalities.

On the contrary, discretization methods are very flexible and can easily deal with all types of constraints, although they may be computer- and memory-intensive. These methods consist in discretizing the workspace or joint space into a regular grid of nodes, and solving

* Corresponding author.
E-mail addresses: apeidro@umh.es (A. Peidró), o.reinoso@umh.es (Ó Reinoso), arturo.gil@umh.es (A. Gil), jmarin@umh.es (J.M. Marín), lpaya@umh.es (L. Payá).

http://dx.doi.org/10.1016/j.engappai.2017.06.009
Received 11 February 2016; Received in revised form 19 May 2017; Accepted 12 June 2017
Available online 14 July 2017
0952-1976/© 2017 Elsevier Ltd. All rights reserved.

at each node the inverse or forward kinematic problem, respectively, to obtain a complete configuration of the robot (Macho et al., 2009; Bonev and Ryu, 2001; Pisla et al., 2013; Cervantes-Sánchez et al., 2000). Then, it is checked if each configuration belongs to the workspace and satisfies all the considered constraints (e.g. joint limits or avoidance of self-interferences).

A variation of the previous discretization methods (in which the joint space or workspace are discretized into regular grids) is the Monte Carlo method, in which a large number of configurations of the robot are randomly generated. The Monte Carlo method is a simple and widely used method especially suitable for computing the workspace of complex robots subject to complicated constraints, which have many degrees of freedom or are even kinematically redundant, like humanoid robots (Guan et al., 2008; Badescu and Mavroidis, 2004; Cao et al., 2011; Rastegar and Perel, 1990; Alciatore and Ng, 1994; Burgner-Kahrs et al., 2014; Wang et al., 2008). Despite being widely used, however, classical or usual Monte Carlo methods usually generate inaccurate workspaces that may be pretty different from the true workspaces. In most cases, to increase the accuracy of the workspace, the number of randomly generated configurations is simply increased. However, this is an inefficient solution that may be far from solving the accuracy problem.

To solve this accuracy problem, this paper proposes a new Monte Carlo method, which we called Gaussian Growth (GG). The GG method consists of generating first an inaccurate seed workspace using a classical Monte Carlo method, and then growing this seed workspace using the Gaussian (or normal) random distribution, until an accurate approximation of the workspace is obtained. The proposed method is more efficient than previous Monte Carlo methods, because it is able to compute more accurate workspaces requiring the same or less time than these methods.

The remainder of this paper is organized as follows. First, Section 2 reviews classical Monte Carlo methods and analyzes their accuracy problems. Next, Section 3 presents in detail the new proposed GG method. Section 4 describes a 10-degrees-of-freedom robot that will be used as a case study to illustrate the proposed method. Then, the GG method is compared with previous Monte Carlo methods in Section 5, analyzing some examples. Finally, Section 6 presents the conclusions.

2. The accuracy problems of classical Monte Carlo methods

This section reviews classical Monte Carlo methods for calculating the workspace of robot manipulators, and analyzes the accuracy problems of these methods. Let $\mathbf{q} = \left[q_1, \dots, q_d\right]^T$ denote the vector of joint coordinates of a robot with d degrees of freedom (DOF). The Monte Carlo method consists of generating a large number of random vectors $\mathbf{q}$ and, for each of them, solving the forward kinematic problem to obtain the position $\mathbf{X} \in \mathbb{R}^3$ of the end-effector of the robot. The components of each random vector $\mathbf{q}$ are randomly generated as follows:

$$q_k = q_k^{min} + (q_k^{max} - q_k^{min}) r_k, \quad k = 1, \dots, d \tag{1}$$

where $\left\{q_k^{min}, q_k^{max}\right\}$ are the joint limits of joint coordinate q_k and r_k is a random variable in $(0, 1)$. After generating each random position of the robot, one should check if it satisfies other additional constraints that may exist (for example, different parts of the robot should not interfere, or the end-effector should have a desired orientation). If all the constraints are satisfied, the generated point $\mathbf{X}$ is stored as a workspace point, and the set of all stored points constitutes a discrete approximation of the workspace of the manipulator.

The workspace generated in this way is a point cloud in $\mathbb{R}^3$ that can be represented graphically. However, for the practical use of the workspace (e.g. for path planning), it is necessary to build a database of the workspace using the generated random points (Guan et al., 2008). To build the database, the Cartesian space is discretized with desired resolution, obtaining a set of cells in this space. Then, all cells which contain at least one workspace point are classified as "reachable", whereas the remaining cells are considered "unreachable" (see Fig. 1(a)). The boundaries of the workspace can be approximated by the set of reachable cells which have at least one neighboring unreachable cell. In this paper, the neighbors that will be considered in 3D are the 26-neighbors, whereas in 2D the 8-neighbors will be considered (see Fig. 1(b)).

Usually, the variable r_k in Eq. (1) is a uniform random number in $(0, 1)$. However, as pointed out by Cao et al. (2011), this choice generally yields inaccurate and nonuniform workspaces, in which some regions are very dense and well-defined (regions populated by many workspace points) whereas other regions, especially those near the boundaries of the workspace, are too sparse (regions with comparatively much fewer points) and make it difficult to figure out the true shape of the workspace. For example, Fig. 2(a) shows the workspace of the robot described in Section 4, composed of $9 \cdot 10^6$ random workspace points obtained by sampling the joint coordinates from uniform distributions. Note that the true workspace of the robot, shown in Fig. 2(b), is bigger and has much better defined boundaries than the workspace obtained by sampling from uniform distributions, which has noisy and irregular boundaries.

The reason for this nonuniform density of the workspace is the nonlinearity of the forward kinematics transformation, which transforms joint coordinates $\mathbf{q}$ into position coordinates $\mathbf{X}$ of the end-effector. Although the joint coordinates are distributed uniformly, this uniformity is not conserved by the nonlinearity of the transformation $\mathbf{q} \to \mathbf{X}$. As a result, $\mathbf{X}$ is distributed according to a nonuniform distribution, which has high-probability regions (regions in which workspace points are generated more often, like the internal regions indicated in Fig. 2(a)) and regions of low probability (sparse regions in which points are hardly generated, like the workspace boundaries in Fig. 2(a)). It should be noted that, although this nonuniform density may be undesirable for obtaining accurate workspaces, it is useful as a measure of the degree of redundancy of the robot across its workspace (Burgner-Kahrs et al., 2014). Indeed, the denser a region of the workspace is, the higher the redundancy is, because it means that the end-effector can be placed in that region with a wider variety of configurations.

To correct this accuracy problem and increase the density of points in sparse regions, one may try to increase the number of randomly generated points, therefore increasing the computation time. However, this is not an efficient solution since most points still fall in high-probability regions (Cao et al., 2011). Alternatively, to solve this problem and achieve more accuracy (especially near the workspace boundaries), Cao et al. (2011) proposed using symmetric U-shaped beta distributions to sample the joint coordinates, instead of using uniform distributions. In that case, r_k in Eq. (1) is a random variable with the following probability density function:

$$f(r_k, \beta_k) = K\left[r_k\left(1 - r_k\right)\right]^{\beta_k - 1} \tag{2}$$

where $0 < r_k < 1$, $0 < \beta_k \le 1$ and K is a normalization constant such that $\int_0^1 f(r_k, \beta_k)\,\mathrm{d}r_k = 1$. This U-shaped distribution, shown in Fig. 3 for different values of β_k, diverges to infinity at $r_k = 0$ and $r_k = 1$, and it is symmetric with respect to $r_k = 0.5$, where the minimum probability occurs. Parameter β_k determines the shape of the distribution: the smaller β_k is, the less probable the values around $r_k = 0.5$ are, and the more probable the values near the limits ($r_k = 0$ and $r_k = 1$) are. As β_k increases, the distribution adopts a more horizontal shape, and all values of $r_k \in (0, 1)$ acquire a more similar probability. The uniform distribution is a particular case of the beta distribution when β_k tends to 1 (see the case $\beta_k = 0.99$ in Fig. 3).

As demonstrated in Cao et al. (2011), using the beta distribution of Eq. (2) to randomly sample the joint coordinates may yield more uniform workspaces, and with better defined boundaries, than using uniform distributions (generating in both cases the same number of random points). This is because, in many cases, the boundaries are typically attained when some joint coordinates reach their joint limits.

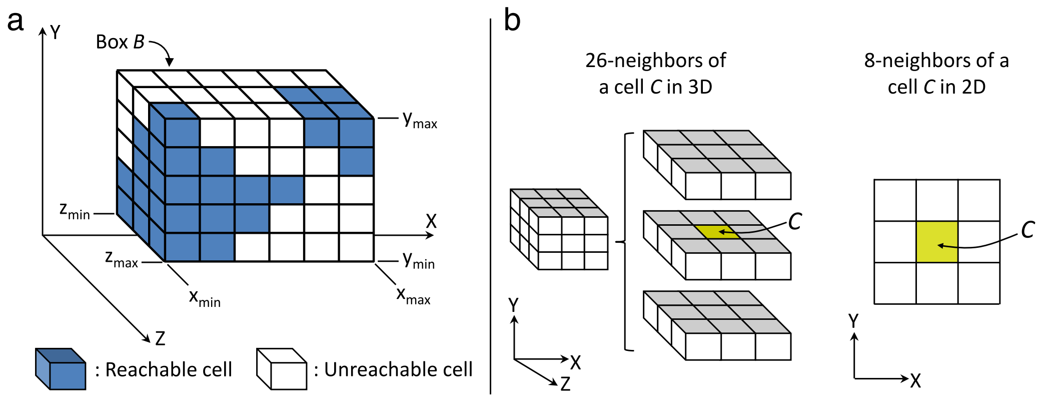


Fig. 1. (a) A box B is defined and discretized into n_x, n_y and n_z cells along the X, Y, and Z axes, respectively (in this figure: $n_x = 6$, $n_y = 5$ and $n_z = 3$). (b) Illustration of the neighbors of a cell C considered in this paper, in three and two dimensions.

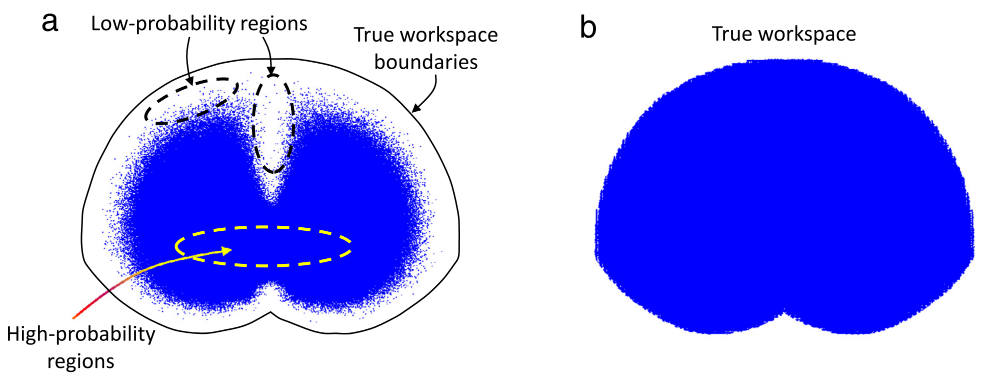


Fig. 2. Comparison between the workspace obtained sampling the joint coordinates from uniform distributions (a) and the true workspace (b) of the robot described in Section 4. The workspace in (b) has been obtained using the new method proposed in Section 3. The time required to compute the workspaces (a) and (b) is the same.

If the beta distribution of Eq. (2) is used, values of r_k near 0 and 1 will be generated more often than other values. Thus, according to Eq. (1), more vectors of joint coordinates $\mathbf{q}$ will be generated near the joint limits. When transforming these vectors into workspace points, more points will be generated near the boundaries and, as a consequence, these boundaries will be better defined.

Although using beta distributions may yield more accurate workspaces than uniform distributions, we will show that this higher accuracy may still be insufficient. In this paper we propose a new Monte Carlo method based on normal distributions to compute the workspace of robot manipulators. This new method is compared with classical methods that use uniform or beta distributions. The proposed method, which is described in the next section, is able to obtain much more accuracy than previous Monte Carlo methods, requiring the same or less computation time than other methods.

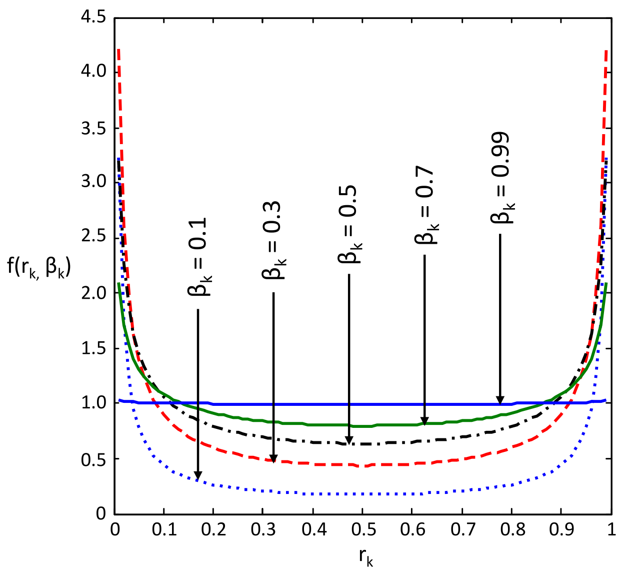


Fig. 3. Symmetric U-shaped beta distribution.

3. A new Monte Carlo method based on Gaussian growth

A drawback of classical Monte Carlo methods, in which the joint coordinates are sampled from some random distribution (e.g., from uniform or beta distributions), is the fact that the random workspace points generated by these methods are distributed nonuniformly throughout the workspace. Thus, some regions are very dense and accurately defined whereas other regions are sparse and poorly defined. To densify these sparse regions and improve the accuracy of the workspace, it is necessary to increase the number of randomly generated points. However, that solution is not efficient because most of the newly generated points still fall in high-probability regions, i.e., much of the effort made to densify sparse regions is wasted in populating areas of the workspace that are already sufficiently populated.

Instead of using previous brute-force methods, in which more and more random points are generated in the whole workspace only to densify sparse regions, it would be more efficient to directly focus on densifying low-density regions until a uniform density is achieved throughout the workspace. In that case, all regions of the workspace would be equally well defined, including the boundaries. This is the objective of the method proposed in this section. Basically, the proposed method consists of generating an initial or *seed* imprecise workspace using a classical Monte Carlo method, and then growing and densifying low-density regions of this seed workspace using Gaussian (or normal) distributions. The method is divided into two stages, which are described next.

3.1. Stage 1: generating a seed workspace

First, a classical Monte Carlo method is used to generate N_s workspace points, where N_s is much smaller than the number of points that will constitute the final dense workspace that will be obtained after both stages of the method have been completed. The objective at this point is to quickly generate, with little effort, an initial inaccurate approximation of the workspace, whose points will be used as seeds around which an accurate and dense approximation of the workspace will be grown during the second stage of the method. Seed points can be generated sampling the joint coordinates from any random distribution, e.g., from uniform or beta distributions. The choice of this initial distribution will be briefly discussed in Section 5.

To identify the regions of the workspace that have low density of points and need to be densified, it is necessary to discretize the 3D space into a set of cells with desired resolution along each dimension, and count the number of workspace points inside each cell. To this end, a box $\mathcal{B} = [x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [z_{min}, z_{max}]$ is first defined, which will enclose the densified workspace. Next, this box $\mathcal{B}$ is discretized into smaller cells, dividing its X, Y, and Z dimensions into n_x, n_y, and n_z equal parts, respectively (see Fig. 1(a)).

Next, a database of cells is created, and the N_s seed points are stored in the corresponding cells of this database, storing up to N_c points in each cell (if the method attempts to store a point in a cell which already has N_c points, then that point is discarded). When storing each workspace point $\mathbf{X}^m$ in the cell in which it falls ($m = 1, \dots, N_s$), we also store the vector of joint coordinates $\mathbf{q}^m$ that generated $\mathbf{X}^m$ (see Fig. 4(c)). This is because $\mathbf{q}^m$ will be used in the second stage of the method to densify and grow the workspace. This step is illustrated in Fig. 4 for the 2D case, for the sake of clarity.

After finishing the previous step, the cells containing N_c points are considered to be sufficiently dense, and the second stage of the method will not try to densify them. On the contrary, non-empty cells containing less than N_c points are included in a list PC of "pending cells", which is the set of cells that will be densified during the second stage of the method, since it is considered that they do not contain enough points to accurately define a workspace region.

3.2. Stage 2: densifying and growing the seed workspace

Upon completion of the first stage, we have a list PC of non-empty cells which should be densified since they contain less than N_c workspace points. The second stage of the proposed method focuses on densifying all pending cells until PC is empty, growing also the workspace during this densification process. The steps of this stage are summarized in Algorithm 1. Next, these steps will be detailed, omitting first some lines of Algorithm 1 for ease of exposition. The omitted lines will be described later.

The second stage of the method will not stop until list PC is empty (line 1 of Algorithm 1), i.e., until all pending cells have been densified. For each cell $C \in PC$, the algorithm will try to generate new random workspace points in C, until C contains N_c points (line 5). To generate a new point in C, one of the workspace points already stored in C is selected randomly (line 12). This randomly chosen workspace point of C is denoted by $\mathbf{X}^0$, and the vector of joint coordinates that generated $\mathbf{X}^0$ is denoted by $\mathbf{q}^0$ (line 13). Note that $\mathbf{q}^0$ is known because it was stored in the database created in the first stage, together with $\mathbf{X}^0$. Then, a new vector of joint coordinates $\mathbf{q}^*$ can be generated in the neighborhood of $\mathbf{q}^0$ using a multivariate normal distribution with mean $\mathbf{q}^0$ and appropriate covariance matrix. Alternatively, instead of sampling from a multivariate normal distribution, it may be simpler and sufficient to sample each joint coordinate q_k^* of $\mathbf{q}^*$ independently from a univariate normal distribution with mean q_k^0 and standard deviation σ_k ($k = 1, \dots, d$), as shown in line 14.

If the newly generated vector of joint coordinates $\mathbf{q}^*$ satisfies the joint limits, the forward kinematic problem is solved to obtain the position $\mathbf{X}^*$ of the end-effector (lines 15 and 16). Next, it is checked if the generated position satisfies other *additional constraints* that may be considered, such as a desired orientation for the end-effector, or the absence of self-interferences or interferences with obstacles of the environment (line 17). If all constraints are satisfied, we proceed to the following steps.

Following, if point $\mathbf{X}^*$ is inside box $\mathcal{B}$, both $\mathbf{X}^*$ and $\mathbf{q}^*$ are stored in the cell C^* of $\mathcal{B}$ in which $\mathbf{X}^*$ falls, provided that C^* is not full (lines 18 to 25). Note that if $\mathbf{X}^*$ is the first point stored in cell C^* (i.e., the cell was empty prior to storing $\mathbf{X}^*$ in it), this cell is included in the list PC of pending cells which require densification (lines 21 and 22). Similarly, if $\mathbf{X}^*$ is the point that fills cell C^* (i.e. $\mathbf{X}^*$ is the N_c-th point stored in C^*), this cell is removed from the list of pending cells (lines 24 and 25).

Note that the cell C^* in which the generated point $\mathbf{X}^*$ falls may not be the current cell C that we are trying to densify, especially if the standard deviations used to generate $\mathbf{X}^*$ are too large (i.e., $\mathbf{q}^*$ is generated too far from $\mathbf{q}^0$). Thus, it may take too long to fill each pending cell C if the algorithm finds it difficult to generate points inside C. To avoid this, variable standard deviations are used instead of constant deviations. When beginning to densify each cell C, the standard deviations used to sample random joint coordinates from normal distributions are initialized to desired values σ_k^{ini} (line 3). Then, if the algorithm fails too often to generate points in C (because most randomly generated workspace points fall in cells other than C), the standard deviations are decreased so that the normal distributions centered at $\mathbf{q}^0$ become narrower and the probability of generating a point close to $\mathbf{q}^0$ (which *does* generate a point in C) increases.

The decrease of standard deviations is implemented as follows. Every time the algorithm fails to generate a point in current cell C, a counter variable n_f (which counts the number of successive failed attempts to generate a point in C) is increased by one unit (line 11). When n_f exceeds a threshold n_f^{max}, n_f is reset and each standard deviation σ_k is decreased dividing it by $\omega_k > 1$ (lines 8 to 10). Whenever the algorithm manages to generate a point in C, counter n_f is reset (lines 26 and 27).

The proposed method not only guarantees a uniform densification of the workspace, it also guarantees the growth of the workspace beyond the initial seed workspace obtained during the first stage of the method. When attempting to generate more points in each pending cell C, some of the generated points will fall in nearby cells due to the shape of the normal distribution (which has a decreasing but non-zero probability density as we move away from the mean). This means that, when trying to densify each cell C, the algorithm will also densify nearby cells, and in particular it will store points in nearby cells that were previously empty. These cells will be included in the list of pending cells, and when the algorithm tries to densify them the process will repeat: new cells around these will be populated, and this process will continue until the boundaries of the workspace are attained.

Finally, it should be noted that the algorithm does not necessarily have to completely densify all workspace cells to obtain an accurate workspace, i.e., it is not necessary to store exactly N_c points inside each cell, as justified next. When trying to densify an arbitrary pending cell C, if all its neighboring cells contain points, then we can conclude that C is not a boundary cell and we can guarantee that both C and its neighbors can be attained by the robot. Thus, there is no need to continue generating more points in C, which can be removed from list PC (lines 6 and 7). If C has empty neighboring cells, C may be a boundary cell and the algorithm should continue densifying C and trying to populate its neighbors, to ascertain whether the workspace has a boundary at C or, on the contrary, the workspace can be grown further beyond C. Avoiding the complete densification of non-boundary cells saves computation time without affecting the accuracy of the final result.

After the second stage of the method has finished, the workspace is constituted by all points that have been stored in cells of box $\mathcal{B}$, and the

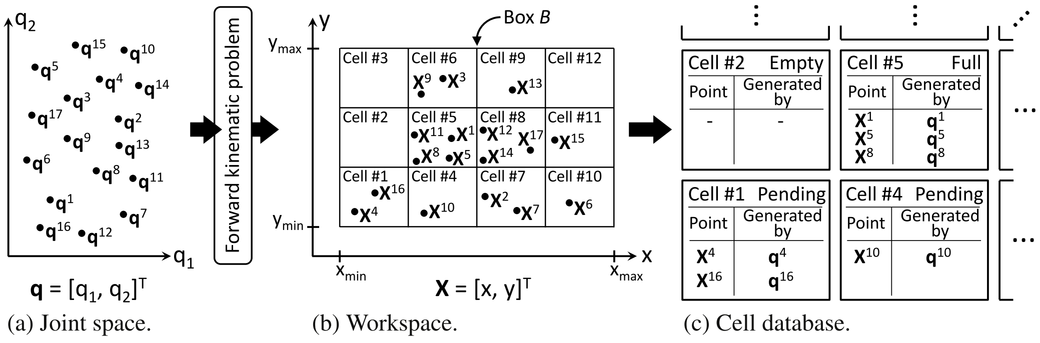


Fig. 4. A 2D example of the first stage of the proposed GG method. (a) First, $N_s = 17$ seed joint coordinate vectors are randomly sampled in the joint space. (b) These vectors are mapped to the workspace solving the forward kinematic problem. The workspace is enclosed by a box $\mathcal{B}$, which is discretized into $n_x = 4$ and $n_y = 3$ cells along the X and Y axes, respectively. (c) A database of cells is created, and up to $N_c = 3$ points are stored in each cell. Cells $\{\#2, \#3, \#12\}$ are empty, cells $\{\#1, \#4, \#6, \#7, \#9, \#10, \#11\}$ are pending cells, and cells $\{\#5, \#8\}$ are full. Although cell #5 contains four points, only the first three points $\{\mathbf{X}^1, \mathbf{X}^5, \mathbf{X}^8\}$ are stored in the database since $N_c = 3$.

boundaries can be approximated by the set of non-empty cells that have empty neighboring cells.

Next, the performance of the method proposed in this section will be compared with classical Monte Carlo methods. The comparisons will be performed using a 10-DOF robot manipulator as a case study, which is described in the next section.

Algorithm 1 Gaussian densification and growth of the seed workspace

```
1: while list $PC$ of pending cells is not empty do
2:     $C \leftarrow$ first pending cell of list $PC$
3:     Initialize standard deviations: $\sigma_k \leftarrow \sigma_k^{ini}$ $(k = 1, \dots, d)$
4:     $n_f \leftarrow 0$
5:     while $C$ contains less than $N_c$ points do
6:         if All neighboring cells of $C$ contain points then
7:             Remove $C$ from $PC$ and go back to line 1
8:         if $n_f > n_f^{max}$ then
9:             $n_f \leftarrow 0$
10:            Decrease standard deviations: $\sigma_k \leftarrow \sigma_k / \omega_k$ $(k = 1, \dots, d)$
11:        $n_f \leftarrow n_f + 1$
12:        $\mathbf{X}^0 \leftarrow$ workspace point randomly picked among those in $C$
13:        $\mathbf{q}^0 \leftarrow$ vector of joint coordinates that generated $\mathbf{X}^0$
14:        Sample $\mathbf{q}^*$ around $\mathbf{q}^0$: $q_k^* \leftarrow Normal(q_k^0, \sigma_k)$ $(k = 1, \dots, d)$
15:        if $\mathbf{q}^*$ satisfies joint limits then
16:            Solve the forward kinematic problem, obtaining $\mathbf{X}^*$ from $\mathbf{q}^*$
17:            if $\mathbf{X}^*$ satisfies additional constraints then
18:                if $\mathbf{X}^* \in \mathcal{B}$ then
19:                    Find the cell $C^*$ of $\mathcal{B}$ in which $\mathbf{X}^*$ falls
20:                    if $C^*$ has less than $N_c$ points then
21:                        if $C^*$ is empty then
22:                            Add $C^*$ to list $PC$
23:                        Store $\mathbf{X}^*$ (and also $\mathbf{q}^*$) in cell $C^*$ of the database
24:                        if $C^*$ contains $N_c$ points then
25:                            Remove $C^*$ from list $PC$
26:                    if $C^* = C$ then
27:                        $n_f \leftarrow 0$
```

4. Case study: a 10-DOF climbing robot

This section describes a biped robot that will be used as a case study in Section 5 to compare different Monte Carlo methods. The robot, shown in Fig. 5(a), is designed for climbing and exploring 3D structures (such as metallic bridges or skeletons of buildings) in order to inspect and maintain them. The feet carry magnets with which the robot adheres to the structure. For example, Fig. 5(b) shows the robot performing a transition between two perpendicular beams in a structure. For further information about this robot, the reader is referred to (Peidró et al., 2016; 2015).

The robot has 10 DOF: eight linear actuators (four per leg) and two revolute actuators in the hip, which allow the robot to rotate the legs as indicated by angles $\{\theta_A, \theta_B\}$ of Fig. 5(a). Fig. 5(c) shows a detailed view of the geometry of an arbitrary leg j (j = A, B). The joint coordinates are the lengths $\{l_{ij}, r_{ij}\}$ of the linear actuators and rotations $\{\theta_A, \theta_B\}$. Thus, the vector of joint coordinates is: $\mathbf{q} = [l_{1A}, r_{1A}, l_{2A}, r_{2A}, l_{1B}, r_{1B}, l_{2B}, r_{2B}, \theta_A, \theta_B]^T$.

Regarding the workspace of the robot, we are interested in calculating the set of positions that foot B can attain while foot A is firmly attached to the structure. The position $\mathbf{X} = [x, y, z]^T$ of foot B relative to foot A can be obtained as follows (Peidró et al., 2016):

$$\mathbf{X} = y_A \begin{bmatrix} s_{\varphi_{1A}} \\ c_{\varphi_{1A}} \\ 0 \end{bmatrix} + y_B \begin{bmatrix} -c_\Theta c_{\Phi_A} s_{\varphi_{2B}} - s_{\Phi_A} c_{\varphi_{2B}} \\ c_\Theta s_{\Phi_A} s_{\varphi_{2B}} - c_{\Phi_A} c_{\varphi_{2B}} \\ s_\Theta s_{\varphi_{2B}} \end{bmatrix} + t \begin{bmatrix} c_{\theta_A} c_{\Phi_A} \\ -c_{\theta_A} s_{\Phi_A} \\ -s_{\theta_A} \end{bmatrix} \quad (3)$$

where $\Phi_j = \varphi_{1j} - \varphi_{2j}$, $y_j = y_{1j} + y_{2j} - h$, and $\Theta = \theta_A - \theta_B$. The variables $\{y_{ij}, \varphi_{ij}\}$ can be analytically obtained from $\{l_{ij}, r_{ij}\}$ solving the forward kinematic problem of the legs (Peidró et al., 2015). The orientation of foot B relative to foot A is defined by the following rotation matrix (Peidró et al., 2016):

$$\mathbf{R} = \begin{bmatrix} s_{\Phi_A} s_{\Phi_B} + c_\Theta c_{\Phi_A} c_{\Phi_B} & s_{\Phi_A} c_{\Phi_B} - c_\Theta c_{\Phi_A} s_{\Phi_B} & s_\Theta c_{\Phi_A} \\ c_{\Phi_A} s_{\Phi_B} - c_\Theta s_{\Phi_A} c_{\Phi_B} & c_{\Phi_A} c_{\Phi_B} + c_\Theta s_{\Phi_A} s_{\Phi_B} & -s_\Theta s_{\Phi_A} \\ -s_\Theta c_{\Phi_B} & s_\Theta s_{\Phi_B} & c_\Theta \end{bmatrix}. \quad (4)$$

To compute the workspace by randomly sampling the joint coordinates, the following joint limits are assumed for the linear actuators: $l_{ij}, r_{ij} \in [\rho_0, \rho_0 + \Delta\rho]$, where $\rho_0 > 0$ is the minimum length of the actuators and $\Delta\rho > 0$ is their stroke.

For joint coordinates $\{\theta_A, \theta_B\}$, it will be assumed that they do not have joint limits, i.e., the legs can rotate freely with respect to the hip. However, since the configuration of the robot will not be affected if an integer multiple of 2π rad is added to angles $\{\theta_A, \theta_B\}$, we will restrict these angles to interval $[0, 2\pi]$. Although $\{\theta_A, \theta_B\}$ are restricted to this interval, since 0 and 2π are not true joint limits of these joint coordinates, the beta distribution should not be used to sample them. Therefore, when using classical Monte Carlo methods in the following section, angles $\{\theta_A, \theta_B\}$ will always be uniformly sampled in $[0, 2\pi]$, even if the remaining joint coordinates (lengths $\{l_{ij}, r_{ij}\}$ of the eight linear actuators) are sampled from beta distributions.

To calculate the workspace of this robot in the next section, the following values will be used for the design parameters: $b = p = 4$, $h = 16$, $t = 15.6$, $\rho_0 = 19$, and $\Delta\rho = 6$ (all values are in cm; $\{b, p, t, h\}$ are indicated in Fig. 5).

Finally, it should be remarked that, in all examples of the following section, we will impose the condition that the legs of the robot should not collide. To check if the legs collide, each leg is approximated by the union of two cuboids (one for the foot and another for the mechanism that connects the foot to the hip). Then, the Separating Axis Theorem (Ericson, 2004) is used to check if any cuboid of one leg intersects any cuboid of the other leg.

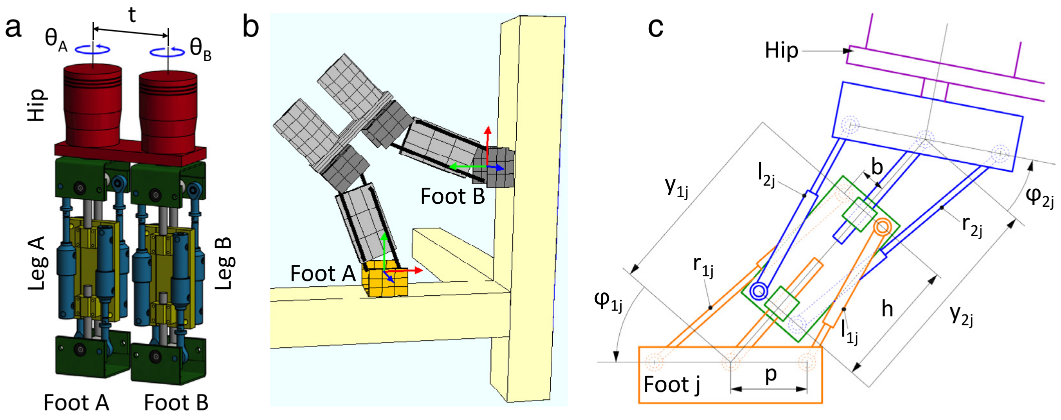
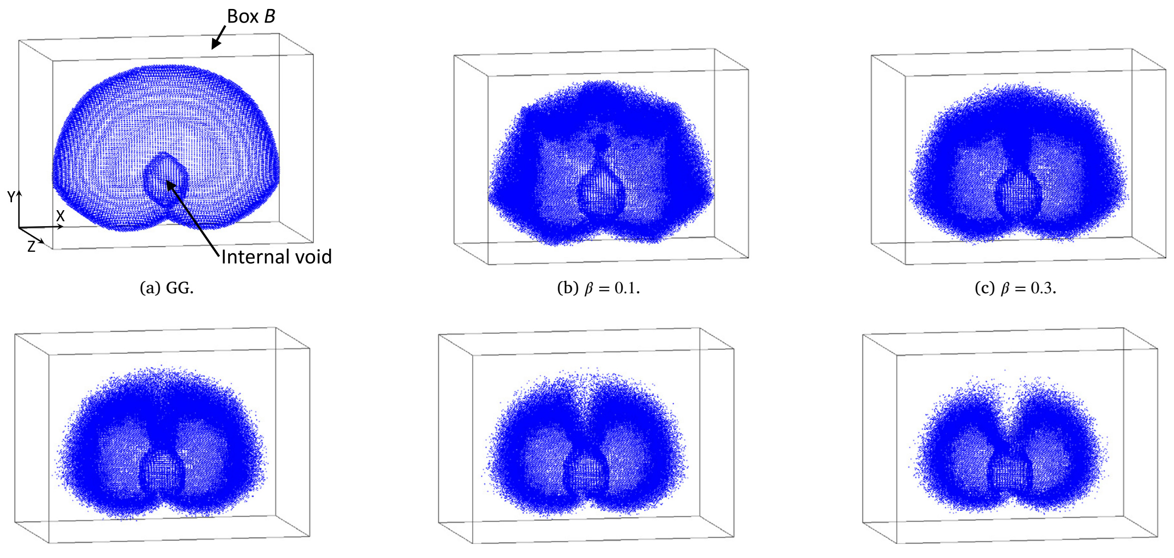


Fig. 5. (a) A 10-DOF climbing robot used as a case study. (b) Example of the robot performing a transition between two perpendicular beams in a structure. (c) Detailed view of the geometry of a leg of the robot (both legs are identical).



(a) GG. (b) $\beta = 0.1$. (c) $\beta = 0.3$.

(d) $\beta = 0.5$. (e) $\beta = 0.7$. (f) Uniform.

Fig. 6. Boundaries of the reachable workspace, obtained with different Monte Carlo methods. The shown boundaries have been extracted from workspaces composed of 3,527,664 points each.

5. Comparative analysis

This section presents a comparison of different Monte Carlo methods to compute the workspace of the 10-DOF robot described in the previous section. The new Gaussian Growth (GG) method proposed in Section 3 will be compared with classical Monte Carlo methods (described in Section 2). When using classical Monte Carlo methods, the joint coordinates will be sampled from uniform or beta distributions. As explained in Section 4, when sampling from beta distributions, only the joint coordinates with true joint limits (i.e., lengths $\{l_{ij}, r_{ij}\}$ of the linear actuators) will be sampled from beta distributions; the rotations of the hip (θ_A and θ_B) will be uniformly sampled in $[0, 2\pi]$.

All examples shown in this section have been implemented in Java programming language and have been tested on a Mac Pro with a 3 GHz 8-Core Intel Xeon E5 processor and 16 GB RAM. Normally distributed random numbers were generated from uniformly distributed random numbers in $(0, 1)$ using Box–Muller's method [see Devroye, 1986, p. 235]. Similarly, beta random numbers were generated from uniform random numbers in $(0, 1)$ using Johnk's method [see Devroye, 1986, p. 432], which is a fast method when the shape parameter β satisfies $0 < \beta < 1$ (which is our case).

In all examples that follow, the seed workspace of the new proposed GG method will be generated by sampling the joint coordinates with joint limits from the beta distribution with $\beta = 0.1$ (θ_A and θ_B will be uniformly sampled). This choice is motivated by two facts that will be observed in the following experiments. In the first place, when generating the same number of workspace points using different values of β, time usually decreases with β. In the second place, when generating the same number of workspace points, these points are more diverse when β is smaller, whereas using a higher value of β (or a uniform distribution, in the limit $\beta = 1$) generates points that are more similar, which is not good for the growing stage of the GG method. To begin growing the workspace in the GG method, it is more convenient to start from diverse and scattered seed points, than from very similar points concentrated in few regions.

5.1. Example 1: reachable workspace

In this example, we compute the reachable workspace, which is the set of points that can be reached by foot B with at least one orientation, i.e., we are not concerned about the orientation of foot B. To obtain this workspace, we only need to randomly sample the joint coordinates and solve the forward kinematic problem to obtain the position $\mathbf{X}$ of foot B [Eq. (3)], checking for the absence of collisions between the legs of the robot.

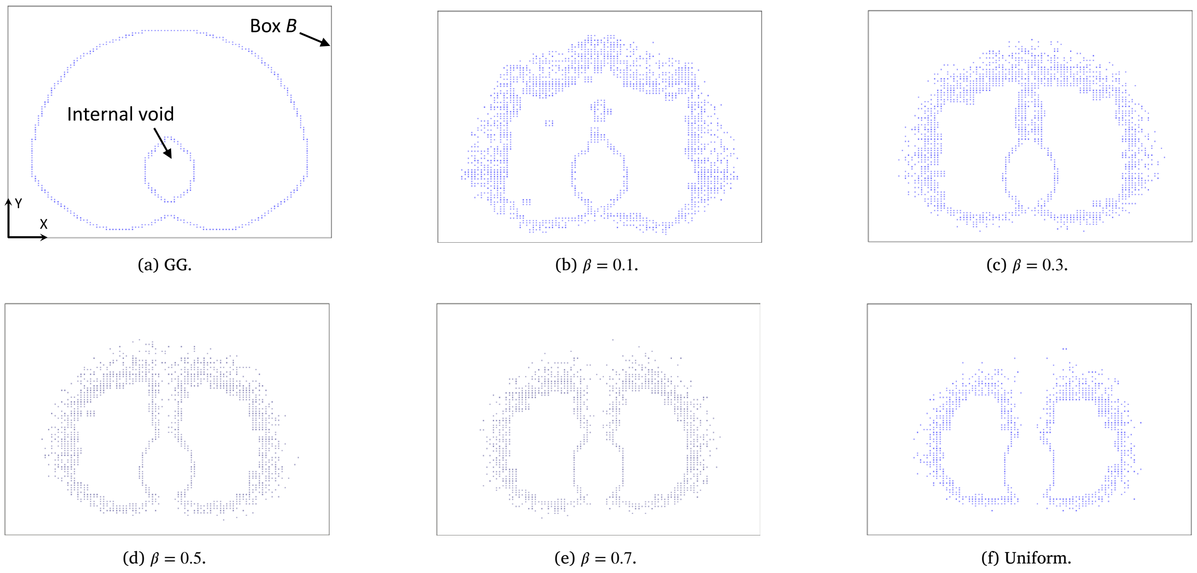


Fig. 7. Intersections between the boundaries of Fig. 6 and the plane $z = 0.45$ cm.

First, the proposed GG method is used to calculate the workspace. A seed workspace of $N_s = 10{,}000$ points is generated by sampling all joint coordinates from beta distributions with $\beta = 0.1$, except for rotations θ_A and θ_B, which are uniformly sampled in $[0, 2\pi]$. Next, the second stage of the method is executed to grow and densify this seed workspace, defining the following box to enclose the workspace: $\mathcal{B} = [-70, 70] \times [-30, 70] \times [-45, 45]$ (all dimensions are in cm). This box is discretized into 100 cells along each axis, i.e., $n_x = n_y = n_z = 100$. The number of desired points in each cell is $N_c = 10$. The maximum number of consecutive failed attempts when trying to generate a point in each cell is $n_f^{max} = 10$. Whenever this maximum is exceeded, all standard deviations are divided by $\omega_k = 1.01$. The initial standard deviations of the joint coordinates when beginning to densify each cell are: $\sigma_k^{ini} = \Delta q_k/6$, where $\Delta q_k = \Delta\rho$ for the lengths $\{l_{ij}, r_{ij}\}$ of the linear actuators, and $\Delta q_k = 2\pi$ rad for $\{\theta_A, \theta_B\}$.

Executing the GG method with the previous parameters, it takes 17.76 min to generate a workspace of 3,527,664 points. This is the time necessary to execute the second stage of the method (densification and growth); the time required to generate the seed workspace is only about 0.04 min, which makes it negligible. Next, these points are assigned to the corresponding cells of box $\mathcal{B}$ and the boundary cells are extracted. Fig. 6(a) shows the boundary surface of the calculated workspace, which is approximated by the centers of all boundary cells. The intersection between this boundary and the plane $z = 0.45$ cm is shown in Fig. 7(a). As these figures show, the reachable workspace has a single large connected component with a lens-shaped void inside. This internal void encloses foot A, and it originates from the condition of no-interference between the legs of the robot.

Next, a classical Monte Carlo method is used to calculate the workspace, generating the same number of points as the GG method (3,527,664 points). The boundaries of the workspaces obtained with classical Monte Carlo methods are shown in Fig. 6(b)–(f). Fig. 6(f) shows the boundaries obtained when all joint coordinates are uniformly sampled, whereas Fig. 6(b)–(e) show the boundaries obtained when sampling all joint coordinates from beta distributions with the indicated values of β (except for $\{\theta_A, \theta_B\}$, which are uniformly sampled in all these cases).

To facilitate the assessment of the precision of the obtained workspaces, Fig. 7 shows the intersections of the boundaries of Fig. 6 with the plane $z = 0.45$ cm. As these figures show, for the same number of random points, the proposed GG method generates a much more precise workspace than classical Monte Carlo methods, which yield noisy, thick and inaccurately defined workspace boundaries in all cases. The second best result after the GG method is obtained for $\beta = 0.1$. Note that the shape of the workspace becomes more distorted as β increases, and for $\beta \geq 0.5$ (approximately) the region above the internal void seems unreachable, which is false. The workspace obtained when sampling uniformly all joint coordinates is worthy of special mention, since it is the least accurate one (Figs. 6(f) and 7(f)).

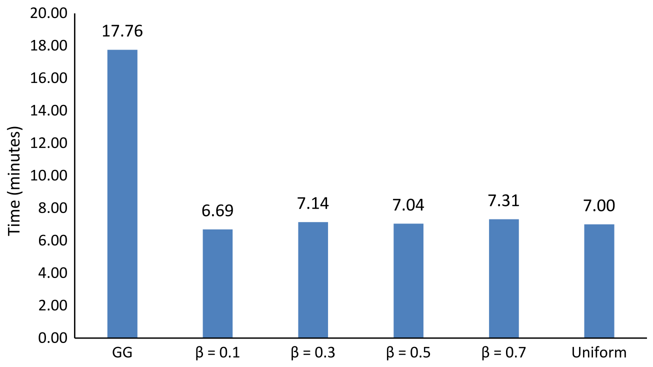
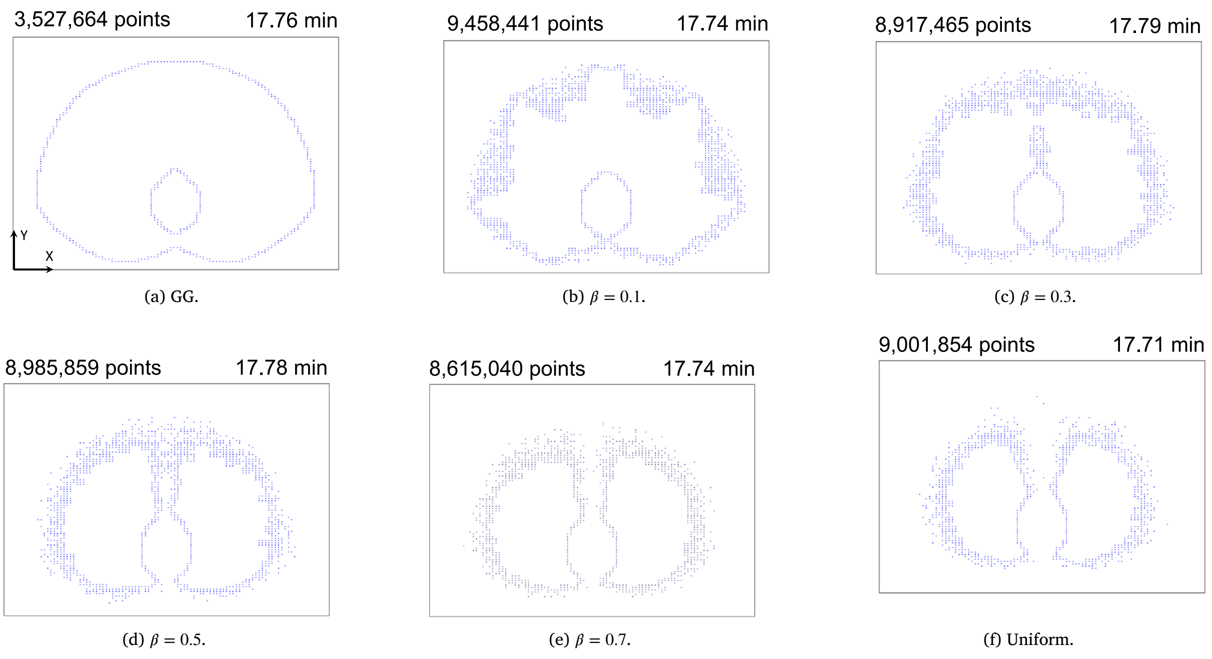


Fig. 8. Time required to generate 3,527,664 workspace points with different Monte Carlo methods.

Fig. 8 shows the time (in minutes) taken by each method to generate 3,527,664 random points. Note that the proposed GG method takes about 2.5 times more time than the other methods, which is not surprising due to its higher precision. The question that arises now is: can the other methods generate as accurate workspaces as the GG method if the number of random points is increased until the computation time equals the time of the GG method? To answer this, the number of random points is increased for classical Monte Carlo methods, until their computation time equals approximately 17.76 min. Fig. 9(b)–(f) show the intersections of the resulting workspace boundaries with the plane $z = 0.45$ cm, along with the number of random points and the actual computation time in each case.

As Fig. 9 shows, the GG method is still much more accurate than the other methods for the same computation time. Note that, although the precision of the other methods has improved slightly, they generate boundaries that are still too noisy and inaccurately defined. This supports the idea that increasing the number of random workspace points



(a) GG. (b) $\beta = 0.1$. (c) $\beta = 0.3$. (d) $\beta = 0.5$. (e) $\beta = 0.7$. (f) Uniform.

Fig. 9. Slices at $z = 0.45$ cm of the workspaces obtained using different Monte Carlo methods (the time is the same in all cases). The number of points of the complete workspace (i.e., not only of the shown planar boundaries) is indicated in each case.

is not an efficient solution for improving the accuracy of classical Monte Carlo methods (Cao et al., 2011).

5.2. Example 2: workspace with equality constraints

In this example, we are interested in calculating a constant-orientation workspace, i.e. the set of points that can be attained with a desired orientation of foot B. The desired orientation is defined by the following rotation matrix, which is a rotation of $\pi/2$ rad about the Z axis:

$$\mathbf{R}^{desired} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{5}$$

As shown in Fig. 5(b), this orientation is necessary to perform a transition between two perpendicular beams of a 3D structure. Furthermore, we are interested only in the intersection of this constant-orientation workspace with the plane $z = 0$, since the motion necessary to perform the transition between the beams is planar. Equating the rotation matrices of Eqs. (4) and (5) yields the following two equations:

$$\cos(\theta_A - \theta_B) = 1 \quad \text{and} \quad \sin(\Phi_A - \Phi_B) = -1. \tag{6}$$

The previous two equations, together with the condition $z = 0$, impose three additional equality constraints to the problem of calculating the workspace. Next, these constraints will be handled following two different approaches.

5.2.1. First approach: approximating equalities by inequalities

The constant-orientation workspace can be calculated following the same procedure followed to calculate the reachable workspace in the previous example, using any Monte Carlo method described in this paper. The only difference is that, in addition to checking the condition of avoiding self-interferences, one should also check that the three aforementioned equality constraints are satisfied for each randomly generated workspace point. However, due to the numerical and random nature of Monte Carlo methods, it is practically impossible that a randomly generated workspace point satisfies exactly all three equalities simultaneously. Thus, it is necessary to transform the equalities into inequalities (Guan et al., 2008), which are easier to satisfy. In this example, the three equality constraints can be approximated by the following inequalities:

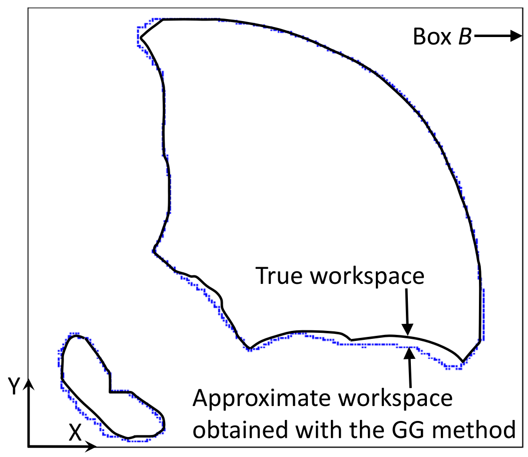


Fig. 10. Planar constant-orientation workspace, obtained approximating all equality constraints by narrow inequalities.

$$|z| < \epsilon_1, \qquad |\cos(\theta_A - \theta_B) - 1| < \epsilon_2, \qquad |\sin(\Phi_A - \Phi_B) + 1| < \epsilon_3 \tag{7}$$

where ϵ_i are sufficiently small. Evidently, the approximation will be better when ϵ_i are smaller, but it will also be more difficult to satisfy the inequalities and the computation time will increase (more random points will need to be sampled until we obtain a sufficiently high number of points that satisfy all inequalities). Next, we will compute the constant-orientation workspace using different Monte Carlo methods, including the inequality restrictions of Eq. (7) in the calculation with the following thresholds: $\epsilon_1 = 0.5$ cm, $\epsilon_2 = 0.05$, and $\epsilon_3 = 0.001$.

First, the GG method is used. A seed workspace of 1000 points is generated by sampling all joint coordinates from beta distributions with $\beta = 0.1$, except for $\{\theta_A, \theta_B\}$, which are uniformly sampled in $[0, 2\pi]$. The time required to generate the seed workspace in this case is 10.42 min. This increase in time with respect to the previous example is due to the

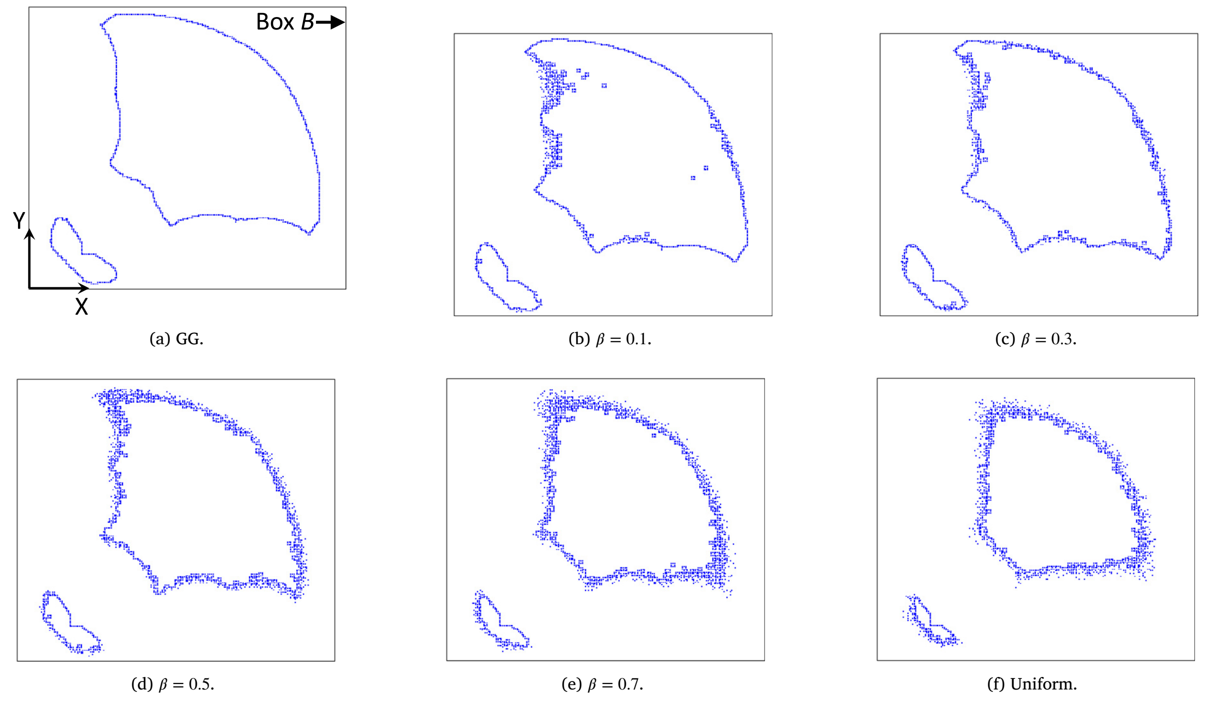


Fig. 11. Boundaries of the planar constant-orientation workspace, obtained with different Monte Carlo methods. The shown boundaries have been extracted from workspaces composed of 521,212 points each.

fact that the workspace points must satisfy the narrow inequalities of Eq. (7) in addition to the condition of avoiding self-interferences. After generating the seed workspace, the densification and growth stage of the GG method is executed. To enclose the workspace, the following box is defined: $\mathcal{B} = [-25, 65] \times [-20, 60] \times [-0.5, 0.5]$ (values in cm). This box is divided into $n_x = 200$, $n_y = 200$, and $n_z = 1$ cells along the X, Y, and Z axes, respectively (since the workspace is planar, the box is not discretized along the Z axis). The number of desired points in each cell is $N_c = 50$, and the maximum number of successive failed attempts is $n_f^{max} = 100$. Whenever this maximum is exceeded, all standard deviations are divided by $\omega_k = 1.01$, and the initial standard deviations of the joint coordinates when beginning to densify each cell are again: $\sigma_k^{ini} = \Delta q_k/6$.

Running the second stage of the GG method with the previous parameters, it takes 4.01 min to densify and grow the seed workspace. The workspace obtained after the second stage has finished is composed of 352,330 points. Thus, the net time necessary to execute the GG method in this example is 14.43 min, which is the sum of the time necessary to generate the seed workspace (10.42 min, not negligible) and the time necessary to grow and densify the seed workspace (4.01 min). The boundaries of the resulting planar workspace are shown in Fig. 10, along with the true boundaries, which are shown in continuous line. Note that the obtained workspace does not exactly coincide with the true one. This is because the equality constraints have been approximated by the inequalities of Eq. (7).

If we tried to generate the workspace with the same number of points (352,330) using classical Monte Carlo methods, the computation time would increase noticeably. For example, as shown immediately above, it takes 10.42 min to generate 1000 seed points when sampling the joint coordinates from beta distributions with $\beta = 0.1$ (except $\{\theta_A, \theta_B\}$, which are uniformly sampled). If all joint coordinates are sampled from uniform distributions, the time necessary to generate 1,000 workspace points increases to 39.47 min. Extrapolating, it is easy to conclude that the time necessary to generate 352,330 points using classical methods would be several hours (or even days), whereas the GG method only needs about 15 min to create the seed workspace and grow/densify it.

This example demonstrates that the proposed GG method is advantageous over classical Monte Carlo methods when narrow constraints are present, such as those obtained when approximating equalities by inequalities. This is because classical Monte Carlo methods generate points in the whole workspace without restriction or guidance, which makes it difficult to generate points that satisfy all narrow inequalities. On the contrary, the GG method focuses on generating points near those that already satisfy the narrow inequalities, i.e., it is a more directed search.

5.2.2. Second approach: solving the equality constraints

Although the first approach is simple, the drawback of approximating equality constraints by narrow inequalities is that the computation time can be high (since it is difficult to generate points that satisfy all narrow inequalities), and the result is always approximate (see Fig. 10). Wang et al. (2008) proposed an alternative way of dealing with equality constraints in Monte Carlo methods, which is explained next.

First, the joint coordinates $\mathbf{q}$ are divided into two classes: independent joint coordinates $\mathbf{q}^{ind}$ and dependent joint coordinates $\mathbf{q}^{dep}$. Next, only the independent joint coordinates are randomly sampled, whereas the dependent joint coordinates are solved from the equality constraints in terms of $\mathbf{q}^{ind}$. Then, if $\mathbf{q}^{dep}$ satisfies the joint limits, the method continues as usual: the forward kinematic problem is solved and $\mathbf{X}$ is obtained. Since the dependent joint coordinates satisfy the equality constraints, this guarantees that vector $\mathbf{X}$ also satisfies these constraints exactly.

This approach is faster and more accurate than the one of Section 5.2.1. However, it is only feasible if the dependent joint coordinates $\mathbf{q}^{dep}$ can be easily solved from the constraints, preferably if we can obtain $\mathbf{q}^{dep}$ analytically in terms of $\mathbf{q}^{ind}$. In the example studied in this section, it can be shown that, if the joint coordinates are partitioned as $\mathbf{q}^{ind} = [l_{1A}, l_{2A}, r_{2A}, l_{1B}, r_{1B}, l_{2B}, r_{2B}]^T$ and $\mathbf{q}^{dep} = [\theta_A, \theta_B, r_{1A}]^T$, then it is possible to solve analytically $\mathbf{q}^{dep}$ in terms of $\mathbf{q}^{ind}$ from the equality constraints. Thus, the method described in Wang et al. (2008) can be used in this example.

Once the joint coordinates have been divided into independent and dependent joint coordinates, both the classical and GG Monte Carlo methods described in this paper can be applied with only two modifications. In the first place, instead of randomly sampling the vector

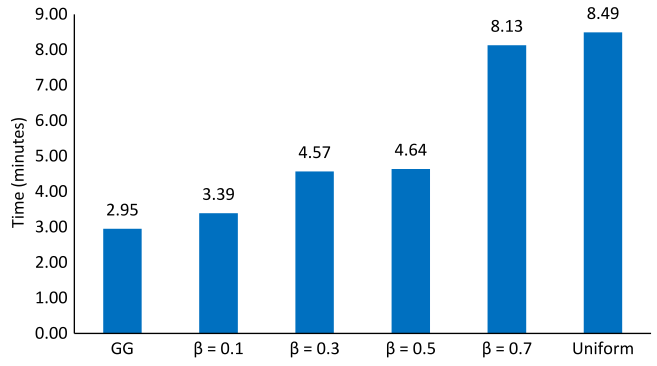


Fig. 12. Time required to generate 521,212 workspace points with different Monte Carlo methods, when solving the equality constraints.

of all joint coordinates $\mathbf{q}$ from any random distribution (uniform, beta, or normal) in any of the previous methods, only the vector of independent joint coordinates $\mathbf{q}^{ind}$ must be sampled (similarly, $\mathbf{q}^{ind}$ instead of $\mathbf{q}$ must be stored in the cell database in the GG method). In the second place, after randomly sampling $\mathbf{q}^{ind}$, $\mathbf{q}^{dep}$ must be solved from the equality constraints and it must be checked if $\mathbf{q}^{dep}$ satisfies the joint limits.

Next, the GG method will be used with the modifications described in this section to compute the planar constant-orientation workspace solving the three aforementioned equality constraints. The parameters of the densification and growth stage of the GG method have the same values as in Section 5.2.1. First, a seed workspace of 1,000 points is generated sampling the independent joint coordinates $\mathbf{q}^{ind} = [l_{1A}, l_{2A}, r_{2A}, l_{1B}, r_{1B}, l_{2B}, r_{2B}]^T$ from beta distributions with $\beta = 0.1$, which takes about 0.02 min. Then, the densification and growth stage of the GG method is executed, which takes 2.95 min and generates a planar constant-orientation workspace consisting of 521,212 points. The boundaries of the resulting workspace, which has two connected components, are shown in Fig. 11(a). In this case, the time necessary to generate the seed workspace is negligible compared to the time of the densification and growth stage.

Next, classical Monte Carlo methods are used to generate 521,212 workspace points, sampling all independent joint coordinates from uniform or beta distributions, and solving the dependent joint coordinates $\mathbf{q}^{dep} = [\theta_A, \theta_B, r_{1A}]^T$ from the equality constraints. The boundaries of the obtained workspaces are shown in Fig. 11, and the times required to generate the 521,212 points in each case are shown in Fig. 12.

As Fig. 11 shows, the most accurate workspace is obtained using the GG method, while the other methods produce noisy and inaccurate boundaries. Again, the worst results are obtained when sampling from uniform distributions, and smaller values of β yield better results than higher values.

Moreover, Fig. 12 shows that, in this example, the GG method is the fastest method for generating the same number of random workspace points (521,212), unlike in the example of Section 5.1. Thus, we can conclude that the GG method can obtain more precise workspaces than classical Monte Carlo methods even requiring less computation time, as this example shows.

6. Conclusions

This paper has presented a new Monte Carlo method based on Gaussian growth for computing the workspace of robots more efficiently than by using classical Monte Carlo methods. While previous Monte Carlo methods try to improve the accuracy by increasing the number of random points in the whole workspace, the GG method focuses on populating poorly defined regions of the workspace. To this end, the GG method consists of two stages: first, an inaccurate seed workspace is generated using a classical Monte Carlo method. Next, Gaussian distributions are used to densify and grow the seed workspace, until the workspace boundaries are attained.

Using a 10-DOF climbing robot as a case study, the proposed GG method has been compared with classical Monte Carlo methods that use uniform or beta distributions. The experiments demonstrate that the GG method can calculate more accurate workspaces than previous Monte Carlo methods requiring the same calculation time. Moreover, if additional constraints are imposed to the calculation of the workspace (e.g., a desired orientation for the end-effector), the GG method may even require less time than previous Monte Carlo methods, attaining also higher accuracy.

The experiments have also confirmed two features of classical Monte Carlo methods observed previously by Cao et al. (2011), namely: using beta distributions may yield better results than using uniform distributions, and increasing the number of random points is not an efficient solution for improving the accuracy. Besides, although the uniform distribution is the most widely used one for calculating the workspace of robots using Monte Carlo methods, the performed experiments discourage using this distribution since it yields poor results when compared with other methods. Still, there are some cases in which the uniform distribution may perform quite well. For example, comparing the GG and uniform sampling methods in robots containing only unbounded revolute joints (i.e., without joint limits), like a general 7R serial arm, reveals that sampling from uniform distributions might generate workspaces that are almost as accurate as those obtained with the proposed GG method (requiring the same computation time). Nevertheless, when joint limits are included in these revolute robots, the GG method outperforms again both beta and uniform sampling methods, obtaining higher accuracy requiring the same computation time.

Finally, it may be possible that the proposed method misses some small components of the workspace when it is composed of some components, as in the example of Fig. 11. Since the GG method is based on the growth of workspace regions from the seed workspace, if the first stage of the method does not generate at least one seed point in a given component of the workspace, then the method may not grow and densify that component. There are several solutions to this problem, which may also be combined. For example, the simplest solution consists in increasing the number N_s of seeds generated during the first stage of the algorithm, with the purpose of increasing the probability of generating at least one seed in every component. Note that, in the example of Section 5.2.2, a densified workspace of 521,212 final points was grown from a seed workspace of 1,000 points, which constitutes less than 0.2% of the number of final points. Such a negligible ratio suggests that we may increase N_s by at least one order of magnitude to try to generate seeds in all components of the workspace, without practically affecting the overall performance of the method. Another solution may consist in choosing an appropriate distribution for the generation of the seed workspace, a distribution that favors the creation of diverse and scattered points in all components of the workspace. As discussed at the beginning of Section 5, beta distributions may be useful for this purpose, but other random distributions may also be explored.

Acknowledgments

This work has been supported by the Spanish Ministry of Education through grant FPU13/00413, by the Spanish Ministry of Economy through project DPI2013-41557-P, and by the Generalitat Valenciana through project AICO/2015/021.

References

Abdel-Malek, K., Yang, J., 2006. Workspace boundaries of serial manipulators using manifold stratification. Int. J. Adv. Manuf. Technol. 28 (11), 1211–1229.
Abdel-Malek, K., Yeh, H.J., Othman, S., 2000. Interior and exterior boundaries to the workspace of mechanical manipulators. Robot. Comput.-Integr. Manuf. 16 (5), 365–376.

Alciatore, D.G., Ng, C.-C.D., 1994. Determining manipulator workspace boundaries using the Monte Carlo method and least squares segmentation. ASME Robotics: Kinematics, Dynamics and Controls 72, 141–146.

Badescu, M., Mavroidis, C., 2004. New performance indices and workspace analysis of reconfigurable hyper-redundant robotic arms. Int. J. Robot. Res. 23 (6), 643–659.

Bohigas, O., Manubens, M., Ros, L., 2012. A complete method for workspace boundary determination on general structure manipulators. IEEE Trans. Robot. 28 (5), 993–1006.

Bonev, I.A., Ryu, J., 2001. A new approach to orientation workspace analysis of 6-DOF parallel manipulators. Mech. Mach. Theory 36 (1), 15–28.

Burgner-Kahrs, J., Gilbert, H.B., Granna, J., Swaney, P.J., Webster III, R.J., Workspace characterization for concentric tube continuum robots. In: Proceedings of the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems IROS 2014, 2014, pp. 1269–1275.

Cao, Y., Lu, K., Li, X., Zang, Y., 2011. Accurate numerical methods for computing 2D and 3D robot workspace. Int. J. Adv. Rob. Syst. 8 (6), 1–13.

Cervantes-Sánchez, J.J., Hernández-Rodríguez, J.C., Rendón-Sánchez, J.G., 2000. On the workspace, assembly configurations and singularity curves of the RRRRR-type planar manipulator. Mech. Mach. Theory 35 (8), 1117–1139.

Devroye, L., 1986. Non-Uniform Random Variate Generation. Springer-Verlag, New York.

Ericson, C., 2004. Real-Time Collision Detection. CRC Press.

Gosselin, C., 1990. Determination of the workspace of 6-DOF parallel manipulators. ASME J. Mech. Des. 112 (3), 331–336.

Guan, Y., Yokoi, K., Zhang, X., 2008. Numerical methods for reachable space generation of humanoid robots. Int. J. Robot. Res. 27 (8), 935–950.

Haug, E.J., Luh, C.M., Adkins, F.A., Wang, J.Y., 1995. Numerical algorithms for mapping boundaries of manipulator workspaces. ASME J. Mech. Des. 118 (2), 228–234.

Liu, X.J, Wang, J, 2014. Parallel kinematics. In: Type, Kinematics, and Optimal Design. Springer-Verlag, Berlin Heidelberg.

Macho, E., Altuzarra, O., Amezua, E., Hernandez, A., 2009. Obtaining configuration space and singularity maps for parallel manipulators. Mech. Mach. Theory 44 (11), 2110–2125.

Merlet, J.P., 1995. Determination of the orientation workspace of parallel manipulators. J. Intell. Robot. Syst. 13 (2), 143–160.

Merlet, J.P, 2006. Parallel Robots. Springer, The Netherlands.

Merlet, J.P., Gosselin, C.M., Mouly, N., 1998. Workspaces of planar parallel manipulators. Mech. Mach. Theory 33 (1–2), 7–20.

Peidró, A., Gil, A., Marín, J.M., Berenguer, Y., Payá, L., Reinoso, Ó, 2016. Monte-Carlo workspace calculation of a serial-parallel biped robot. In: Reis, L.P. Moreira A.P. Lima P.U. Montano L., Muñoz Martinez, V. (Eds.), ROBOT 2015: Second Iberian Robotics Conference. Advances in Robotics, Vol 2. Springer International Publishing, Switzerland, pp. 157–169.

Peidró, A., Gil, A., Marín, J.M., Berenguer, Y., Reinoso, Ó., Kinematic analysis and simulation of a hybrid biped climbing robot. In: Proceedings of the 12th International Conference on Informatics in Control, Automation and Robotics, ICINCO 2015, Vol. 2, 2015, pp.24–34.

Pisla, D., Szilaghyi, A., Vaida, C., Plitea, N., 2013. Kinematics and workspace modeling of a new hybrid robot used in minimally invasive surgery. Robot. Comput.-Integr. Manuf. 29 (2), 463–474.

Rastegar, J., Perel, D., 1990. Generation of manipulator workspace boundary geometry using the monte carlo method and interactive computer graphics. ASME J. Mech. Des. 112 (3), 452–454.

Wang, L., Wu, J., Tang, D., Research on Workspace of Manipulator with Complicated Constraints. In: Proceedings of the 7th World Congress on Intelligent Control and Automation, WCICA 2008, 2008, pp. 995–999.