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

ARTICLE INFO

Keywords:
Gaussian distribution
Monte Carlo method
Robot manipulator
Workspace

ABSTRACT

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
0952-1976/

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 $\mathbf{q} = [q_1, \dots, q_d]^T$ 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 $\{q_k^{min}, q_k^{max}\}$ 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 $\mathbf{q} \rightarrow \mathbf{X}$. 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 $0 < r_k < 1$, $0 < \beta_k \leq 1$ and K is a normalization constant such that $\int_0^1 f(r_k, \beta_k)\, dr_k = 1$. 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.