A Design and an Implementation of an Inverse Kinematics Computation in Robotics Using Gröbner Bases

Noriyuki Horigome^{1,2P}, Akira Terui^{1(✉)} , and Masahiko Mikawa¹

¹ University of Tsukuba, Tsukuba, Japan

{horigome, terui}@math.tsukuba.ac.jp, mikawa@slis.tsukuba.ac.jp

² Internet Initiative Japan Inc., Tokyo, Japan

<https://researchmap.jp/aterui>

<https://www.iiij.ad.jp/en/>

Abstract. The solution and a portable implementation of the inverse kinematics computation of a 3 degree-of-freedom (DOF) robot manipulator using Gröbner bases are presented. The main system was written Python with computer algebra system SymPy. Gröbner bases are computed with computer algebra system Risa/Asir, called from Python via OpenXM infrastructure for communicating mathematical software systems. For solving a system of algebraic equations, several solvers (both symbolic and numerical) are used from Python, and their performance has been compared. Experimental results with different solvers for solving a system of algebraic equations are shown.

Keywords: Gröbner bases · Robotics · Inverse kinematics

1 Introduction

In robotics, inverse kinematics computation is one of the central topics in motion planning [17]. In the field of computer algebra, methods of the inverse kinematics computation using Gröbner bases have been proposed ([2, 7, 22, 23], and the references therein). After formulating the forward kinematics problem using the Denavit–Hartenberg convention, the inverse kinematics problem is derived as a system of algebraic equations by conversion of trigonometric expressions to polynomials. Then, the system is triangularized by computing the Gröbner basis with respect to the lexicographic ordering and solved by appropriate solvers.

Since methods using Gröbner bases solve the inverse kinematics problem directly, these methods have advantages that one can verify if the given inverse kinematics problem is solvable (with the certification of the solution if needed), and if it is solvable, one can obtain the configuration of parameters for the desired motion of the robot directly, before the actual motion. On the other hand, the computational cost of methods using Gröbner bases tends to be high compared to that of numerical methods, thus it is desired to decrease computational cost for

effective computation of solving the inverse kinematics problem using Gröbner bases. Furthermore, for the use of these methods in robotics simulators such as the Robot Operating System (ROS) [9], an implementation that can easily be integrated with these simulators is needed.

In this paper, we present the solution and a portable implementation of the inverse kinematics computation of a 3 degree-of-freedom (DOF) robot manipulator using Gröbner bases. For portable implementation and rapid development, the main program is developed in Python with computer algebra system SymPy [12]. Gröbner bases are computed effectively using computer algebra system Risa/Asir [14], connected to Python with OpenXM infrastructure for communicating mathematical software systems [10]. Then, the system of algebraic equations is solved using an appropriate solver called within Python. As the main focus of our paper, several solvers for solving a system of algebraic equations have been compared: an exact solver included in SymPy, a multivariate numerical solver using the secant method, and a univariate numerical solver with successive substitutions.

The rest of the paper is organized as follows. In Sect. 2, the method of inverse kinematics computation of a 3 DOF manipulator using Gröbner bases is explained. In Sect. 3, the description of the proposed system for solving the inverse kinematics problem is presented. In Sect. 4, the result of experiments is presented.

2 Inverse Kinematics of a 3 Degree-of-Freedom (DOF) Robot Manipulator

In this paper, an example of 3 degree-of-freedom (DOF) manipulator has been built using LEGO® MINDSTORMS® EV3 Education¹ (henceforth abbreviated to EV3) (Fig. 1). An EV3 set has a computer (which is called “EV3 Intelligent Brick”), servo motors and sensors (gyro, ultrasonic, color and touch sensors), along with bricks used for constructing building blocks of robots. While a GUI-based programming environment for controlling the robot is available, several programming languages such as Python, Ruby, C, and Java can also be used on the top of other programming environments.

We first solve the forward kinematics problem. Components of the manipulator are defined as shown in Fig. 2. Segments (links) are called Segment i ($i = 1, 2, 3, 4$) from the one fixed on the ground towards the end effector, and a joint connecting Segment i and $i + 1$ is called Joint i . For Joint i , the coordinate system Σ_i , with the x_i , y_i and z_i axes and the origin at Joint i , is defined according to the Denavit–Hartenberg convention [18] (Fig. 2). Note that, since the present coordinate system is right-handed, the positive axis pointing upwards and downwards is denoted by “ $\odot$ ” and “ $\otimes$ ”, respectively. Furthermore, let Σ_0 be the coordinate system satisfying that the origin is placed at the perpendicular foot from the origin of Σ_1 , and the direction of axes x_0 , y_0 and z_0 are the same

¹ LEGO and MINDSTORMS are trademarks of the LEGO Group.

Fig. 1. A 3 DOF manipulator built with EV3.

as that of axes x_1 , y_1 and z_1 , respectively. Also, let Joint 5 be the end effector, and Σ_5 be the coordinate system with the origin placed at the position of Joint 5 and that the axes x_5 , y_5 and z_5 have the same direction as the axes x_4 , y_4 and z_4 , respectively.

Let a_i be the distance between axes z_{i-1} and z_i , α_i the angle between axes z_{i-1} and z_i with respect to x_i axis, d_i the distance between axes x_{i-1} and x_i , and θ_i be the angle between axes x_{i-1} and x_i with respect to z_i axis. Then, the coordinate transformation matrix ${}^{i-1}T_i$ from the coordinate system Σ_{i-1} to Σ_i is expressed as

$$\begin{aligned}
 {}^{i-1}T_i &= \begin{pmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha_i & -\sin \alpha_i & 0 \\ 0 & \sin \alpha_i & \cos \alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos \theta_i & -\sin \theta_i & 0 & 0 \\ \sin \theta_i & \cos \theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
 &= \begin{pmatrix} \cos \theta_i & -\sin \theta_i & 0 & a_i \\ \cos \alpha_i \sin \theta_i & \cos \alpha_i \cos \theta_i & -\sin \alpha_i & -d_i \sin \alpha_i \\ \sin \alpha_i \sin \theta_i & \sin \alpha_i \cos \theta_i & \cos \alpha_i & d_i \cos \alpha_i \\ 0 & 0 & 0 & 1 \end{pmatrix}.
 \end{aligned}$$

Since the joint parameters a_i , α_i , d_i and θ_i for EV3 are given as shown in Table 1 (note that the unit of a_i and d_i is centimeters) and the transformation matrix T from the coordinate system Σ_5 to Σ_0 is calculated as $T = {}^0T_1 {}^1T_2 {}^2T_3 {}^3T_4 {}^4T_5$, the position ${}^t(x, y, z)$ of the end effector with respect to the coordinate system Σ_0 is expressed as

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 8(2 \cos(\theta_2 + \pi/4) + 2 \cos(\theta_2 + \theta_3 + \pi/4) + \sqrt{2}) \cos \theta_1 \\ 8(2 \cos(\theta_2 + \pi/4) + 2 \cos(\theta_2 + \theta_3 + \pi/4) + \sqrt{2}) \sin \theta_1 \\ 16 \sin(\theta_2 + \theta_3 + \pi/4) + 8 + 8\sqrt{2} \end{pmatrix}. \quad (1)$$

Fig. 2. Components and the coordinate systems of the manipulator.

Table 1. Joint parameters for EV3.

i	a_i (cm)	α_i	d_i (cm)	θ_i
1	0	0	8	θ_1
2	0	$\pi/2$	0	$\pi/4$
3	16	0	0	θ_2
4	16	0	0	θ_3
5	16	0	0	0

Next, for solving the inverse kinematics problem, we solve Eq. (1) with respect to $\theta_1, \theta_2, \theta_3$. With the expansion of trigonometric functions using the trigonometric addition formulas and transformation of trigonometric functions defined as

$$c_i = \cos \theta_i, \quad s_i = \sin \theta_i,$$

subject to $c_1^2 + s_1^2 = 1$, Eq. (1) is transferred to a system of algebraic equations:

$$\begin{aligned}
 f_1 &= 8\sqrt{2}c_1(c_2 + c_3(c_2 - s_2) - s_2 - s_3(c_2 + s_2) + 1) - x = 0, \\
 f_2 &= 8\sqrt{2}s_1(c_2 + c_3(c_2 - s_2) - s_2 - s_3(c_2 + s_2) + 1) - y = 0, \\
 f_3 &= 8\sqrt{2}(c_2 + c_3(c_2 + s_2) + s_2 + s_3(c_2 - s_2) + 1) + 8 - z = 0, \\
 f_4 &= s_1^2 + c_1^2 - 1 = 0, \quad f_5 = s_2^2 + c_2^2 - 1 = 0, \quad f_6 = s_3^2 + c_3^2 - 1 = 0.
 \end{aligned} \tag{2}$$

Then, by putting the coordinate ${}^t(x, y, z)$ into Eq. (2) and computing Gröbner basis of an ideal $\langle f_1, \dots, f_6 \rangle$ with respect to the lexicographic (lex) ordering, a “triangularized” system of equations is obtained. By solving the triangularized system of equations, configuration of the joint angles $\theta_1, \theta_2, \theta_3$ are obtained.

Remark 1. We see that the ideal $\langle f_1, \dots, f_6 \rangle$ is zero-dimensional for generic values of x, y, z , as follows. By computing the comprehensive Gröbner system [8] of $\langle f_1, \dots, f_6 \rangle$ with parameters x, y, z and variables $c_1, s_1, c_2, s_2, c_3, s_3$ in $\mathbb{R}[x, y, z, c_1, s_1, c_2, s_2, c_3, s_3]$ with respect to lex order with $c_1 \succ s_1 \succ c_2 \succ s_2 \succ c_3 \succ s_3$, we have $\{g_1, g_2, g_3, g_4, g_5, g_6\}$ as the Gröbner basis for the generic case². In the generic case, we have $\text{LM}(g_1) = s_3^4$, $\text{LM}(g_2) = c_3$, $\text{LM}(g_3) = s_2$, $\text{LM}(g_4) = c_2$, $\text{LM}(g_5) = s_1$, $\text{LM}(g_6) = c_1$, where $\text{LM}(g)$ denotes the leading monomial of g . This shows that the ideal is zero-dimensional for the generic case [1].

3 Implementation

We have implemented a system for the inverse kinematics computation of the manipulator in SymPy [12] on the top of Python, connecting with the computer algebra system Risa/Asir [14] via OpenXM infrastructure for communicating mathematical software systems [10].

Python (and SymPy) has been chosen for rapid development including the use of the library for solving algebraic equations, and interoperability with the Robot Operating System (ROS) [9] for embedding our present implementation as a simulation environment or an inverse kinematics solver in the future.

OpenXM (which stands for “Open message eXchange protocol for Mathematics”) consists of definitions of protocols and data formats for communication and/or interchange of mathematical information among mathematical software systems. It also includes implementation of interface for various mathematical softwares including Risa/Asir, Kan/sm1 [20], Maple [11], Mathematica [26], MixedVol [3], NTL [16], PHC Pack [25] and TiGERS [4].

Risa/Asir is used for effective computation of Gröbner bases. After receiving input polynomials from Python/SymPy, it first computes the Gröbner basis with respect to the graded reverse lexicographic (grlex) ordering. Then, it converts the basis to the one with respect to lex ordering (with a modular FGLM algorithm [14]) before returning the final result to the host program. Risa/Asir can be called from Python easily using `ctypes` library [15] with the OpenXM interface library for Risa/Asir.

4 Experiments

We have tested our implementation for inverse kinematics computation for randomly given points within the feasible region.

For solving a system of algebraic equations, the following solvers have been used:

1. a built-in exact solver in SymPy (`sympy.solvers.solvers.solve`),

² We have computed the comprehensive Gröbner system on Risa/Asir using an implementation by Nabeshima [13].

2. a numerical solver in Python’s npmath library [6] (`mpmath.findroot`) using multivariate secant method³ (called by `sympy.solvers.solvers.nsolve` function in SymPy package),
3. a numerical solver in Python’s NumPy package [24] (`numpy.roots`) solving univariate algebraic equations with successive substitutions.

For each solver, 10 sets of experiments were conducted with 100 random points given in each set of experiments (thus 1000 random points were given in total).

The computing environment is as follows.

Host environment. Intel Core i5-7360U 2.3GHz, RAM 8GB, macOS 10.15.1, Parallels Desktop Lite 1.4.0.

Guest environment. RAM 2GB, Linux 4.15.0, Python 2.7.12, SymPy 1.4, mpmath 1.1.0, numpy 1.11.0, OpenXM 1.3.3-1, Asir 20191101.

Remark 2. As shown in Remark 1, if the given point is within the feasible region, the system of algebraic equations $g_1 = \dots = g_6 = 0$ has real solution(s) and can be solved rigorously, where $\{g_1, \dots, g_6\}$ is the Gröbner basis of the ideal $\langle f_1, \dots, f_6 \rangle$ with respect to lex order.

Remark 3. For the exact computation of Gröbner bases, the coordinates of the sample points are given as rational numbers with the magnitude of the denominator is less than 100.

4.1 The Result with an Exact Solver (solve)

Table 2 shows the result of experiments with the exact solver (`sympy.solvers.solvers.solve`) [19]. For a system of algebraic equations, the solver computes a Gröbner basis with lex order, solve univariate equations and substitute the roots in the other equations to obtain the other coordinate⁴.

In each test, T_{GB} is the average of computing time of Gröbner basis, T_{Solve} is the average of computing time for solving the system of algebraic equations, T_{Total} is the average of total computing time for inverse kinematics computation, and ‘Error’ is the average of absolute error of the position of the end effector with the configuration of joint angles $\theta_1, \theta_2, \theta_3$, computed by solving the inverse kinematics problem, from the randomly given position. Note that T_{Total} includes the time for synchronizing received data of Gröbner basis from Risa/Asir ($\simeq 1.5$ s) and the time for evaluation of Error. The bottom row ‘Average’ shows the average of the values in each column of the 10 test sets.

In all the test cases, the system of algebraic equations has been rigorously solved with finding appropriate real roots, although it took much time for finding the roots. For finding joint angles $\theta_1, \theta_2, \theta_3$, the solutions $s_1, c_1, s_2, c_2, s_3, c_3$ have been approximated by double precision floating-point numbers for efficient use of arctan function. We see that, though the approximation of the solution, the position of the end effector has been computed with high precision, compared to the length of the segments (Table 1).

³ As the initial values, $(c_1, s_1, c_2, s_2, c_3, s_3) = (1, 1, 1, 1, 1, 1)$ were given.

⁴ The solver may not need a Gröbner basis of lex order as an input, but it might be better to compute beforehand for faster computation.

Table 2. A result of the inverse kinematics computation with the exact solver (`nsolve`).

Test	T_{GB} (s)	T_{solve} (s)	T_{Total} (s)	Error (cm)
1	0.511	37.410	58.771	4.775×10^{-11}
2	0.244	38.268	60.528	4.747×10^{-11}
3	0.378	37.890	59.970	5.015×10^{-11}
4	0.311	38.123	57.955	4.951×10^{-11}
5	0.234	38.615	61.405	4.633×10^{-11}
6	0.471	37.599	59.809	4.525×10^{-11}
7	0.292	37.839	59.784	4.884×10^{-11}
8	0.166	22.145	36.476	4.976×10^{-11}
9	0.254	21.568	35.650	4.794×10^{-11}
10	0.267	21.501	34.217	4.960×10^{-11}
Average	0.312	33.092	52.457	4.826×10^{-11}

4.2 The Result with the Multivariate Numerical Solver (`nsolve`)

Table 3 shows the result of experiments with the multivariate numerical solver (`mpmath.findroot`) [5]. The columns ‘Test’, T_{GB} , T_{Solve} , T_{Total} , ‘Error’, and the bottom row ‘Average’ are the same as those in Table 2. In these sets of experiments, we have observed that the method did not converge in many cases, so the column ‘#Fail’ shows the number of tests in which the method did not converge in each test set. Note that the data in T_{GB} , T_{Solve} , T_{Total} , ‘Error’ and ‘Average’ have been taken for the tests in which the method has successfully converged. The number ‘(564)’ in the ‘Average’ row and the ‘#Fail’ column shows the total number of tests in which the method did not converge⁵.

The result shows that the method did not converge in approximately half of the test cases, while, in the cases that the method converged, the method is more efficient than the exact root-finding method. It also shows that, in the cases that the method converged, the magnitude of the absolute error of the solution is approximately 10 times larger than those in the case of the exact method, which is sufficiently small for practical use⁶.

⁵ We have tested the method with other initial values. With the initial values (1, 0, 1, 0, 1, 0), the number of test cases in which approximate roots do not converge was the same as the test cases with initial values (1, 1, 1, 1, 1). With initial values (0, 1, 0, 1, 0, 1), the approximate roots have never converged to the roots.

⁶ We have also applied the multivariate numerical solver to the original system of equations with initial values (1, 1, 1, 1, 1, 1), (0, 0, 0, 0, 0, 0), (1, 0, 1, 0, 1, 0) and (0, 1, 0, 1, 0, 1), and found that none of the initial values converge to true roots in all the test cases.

Table 3. A result of the inverse kinematics computation with the multivariate numerical solver (`nsolve`).

Test	T_{GB} (s)	T_{Solve} (s)	T_{Total} (s)	Error (cm)	#Fail
1	0.249	0.261	4.096	2.689×10^{-10}	54
2	0.248	0.256	4.086	2.819×10^{-10}	49
3	0.274	0.275	4.140	2.648×10^{-10}	55
4	0.264	0.268	4.112	3.182×10^{-10}	60
5	0.264	0.271	4.121	2.375×10^{-10}	62
6	0.264	0.269	4.119	2.618×10^{-10}	68
7	0.268	0.276	4.126	2.612×10^{-10}	52
8	0.272	0.274	4.133	2.770×10^{-10}	50
9	0.270	0.265	4.119	2.995×10^{-10}	50
10	0.290	0.254	4.118	2.705×10^{-10}	64
Average	0.266	0.267	4.117	2.741×10^{-10}	(564)

Table 4. The result of the inverse kinematics computation with the univariate numerical solver (`roots`) with successive substitutions.

Test	T_{GB} (s)	T_{Solve} (s)	T_{Total} (s)	Error (cm)
1	0.245	0.309	4.181	2.720×10^{-10}
2	0.246	0.306	4.179	2.659×10^{-10}
3	0.246	0.306	4.179	2.639×10^{-10}
4	0.247	0.307	4.181	2.741×10^{-10}
5	0.251	0.313	4.187	2.757×10^{-10}
6	0.250	0.316	4.197	2.720×10^{-10}
7	0.254	0.314	4.198	2.588×10^{-10}
8	0.252	0.307	4.183	2.653×10^{-10}
9	0.254	0.309	4.186	2.773×10^{-10}
10	0.253	0.307	4.181	2.755×10^{-10}
Average	0.250	0.309	4.185	2.700×10^{-10}

4.3 The Result with the Univariate Numerical Solver (`roots`) with successive substitutions

Table 4 shows the result of experiments with the univariate numerical solver (`numpy.roots`) [21]. The columns ‘Test’, T_{GB} , T_{Solve} , T_{Total} , ‘Error’, and the bottom row ‘Average’ are the same as those in Tables 2 and 3. The result shows that the method successfully converged and found appropriate solutions with sufficiently small errors in all the tests.

5 Concluding Remarks

In this paper, we have presented a portable implementation of the inverse kinematics computation of a 3 DOF robot manipulator using Gröbner bases. The implementation is made on the top of Python and SymPy with using Risa/Asir for efficient computation of Gröbner bases and symbolic and/or numerical solvers called within Python for solving a system of algebraic equations. Risa/Asir can easily be called from Python via OpenXM infrastructure.

The experiments have shown the following features of solvers used in solving the system of algebraic equations used in the present computation:

1. Symbolic solver can be used for inverse kinematics computation with high accuracy with the certification of real roots, although the computing time increases.
2. Multivariate numerical solver is often unstable, although it can be used to solve the inverse kinematics problem with high efficiency and accuracy in stable cases.
3. Univariate numerical solver with successive substitutions is stable with high efficiency and accuracy for all the tests in the present paper.

Thus, we see that univariate numerical solver with successive substitutions is effective for solving the inverse kinematics problem in the present case, although certification of real roots may be needed.

For future research, we need to improve implementation for calling Risa/Asir from Python via OpenXM in a more sophisticated way for more efficient computation.⁷ Furthermore, we intend to extend our implementation for embedding our solver in robotics simulators such as ROS and/or controlling the actual EV3 manipulators including the one we have built in the present paper.

References

1. Cox, D.A., Little, J., O’Shea, D.: Using algebraic geometry (2005). <https://doi.org/10.1007/978-1-4757-6911-1>
2. Faugère, J.C., Merlet, J.P., Rouillier, F.: On solving the direct kinematics problem for parallel robots. Research report RR-5923, INRIA (2006). <https://hal.inria.fr/inria-00072366>
3. Gao, T., Li, T.Y., Wu, M.: Algorithm 846: mixedvol: a software package for mixed-volume computation. *ACM Trans. Math. Softw.* **31**(4), 555–560 (2005). <https://doi.org/10.1145/1114268.1114274>
4. Huber, B., Thomas, R.R.: Computing Gröbner fans of toric ideals. *Exp. Math.* **9**(3), 321–331 (2000). <https://doi.org/10.1080/10586458.2000.10504409>

⁷ At this time Risa/Asir is invoked by sending the command in the text form, with the waiting time (approximately 1.5s) for synchronizing output data is set. We expect that this process becomes more efficient by using appropriate API for sending/receiving commands and data.

5. Johansson, F.: mpmath developers: mpmath 1.1.0 documentation: Root-finding and optimization (2018). <http://mpmath.org/doc/current/calculus/optimization.html>
6. Johansson, F.: mpmath developers: mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 1.1.0) (2018). <http://mpmath.org/>. Accessed 20 Mar 2020
7. Kalker-Kalkman, C.M.: An implementation of Buchbergers' algorithm with applications to robotics. *Mech. Mach. Theory* **28**(4), 523–537 (1993). [https://doi.org/10.1016/0094-114X\(93\)90033-R](https://doi.org/10.1016/0094-114X(93)90033-R)
8. Kapur, D., Sun, Y., Wang, D.: An efficient method for computing comprehensive gröbner bases. *J. Symbolic Comput.* **52**, 124–142 (2013). <https://doi.org/10.1016/j.jsc.2012.05.015>
9. Koubaa, A. (ed.): Robot Operating System (ROS). SCI, vol. 625. Springer, Cham (2016). <https://doi.org/10.1007/978-3-319-26054-9>
10. Maekawa, M., Noro, M., Ohara, K., Takayama, N., Tamura, K.: The design and implementation of OpenXM-RFC 100 and 101. In: Shirayanagi, K., Yokoyama, K. (eds.) *Computer Mathematics: Proceedings of the Fifth Asian Symposium on Computer Mathematics (ASCM 2001)*, pp. 102–111. World Scientific (2001). https://doi.org/10.1142/9789812799661_0011
11. Maplesoft, a division of Waterloo Maple Inc.: Maple 2020 [computer software] (2020). <https://www.maplesoft.com/products/maple/>. Accessed 17 Mar 2020
12. Meurer, A., et al.: SymPy: symbolic computing in Python. *PeerJ Comput. Sci.* **3** (2017). <https://doi.org/10.7717/peerj-cs.103>
13. Nabeshima, K.: An implementation of GCS algorithm for Risa/Asir. Private communication (2012)
14. Noro, M.: A computer algebra system: Risa/Asir. In: Joswig, M., Takayama, N. (eds.) *Algebra, Geometry and Software Systems*, pp. 147–162. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-662-05148-1_8
15. Python Software Foundation: The Python Standard Library. Python Software Foundation (2020). <https://docs.python.org/3/library/>. Accessed 15 Mar 2020
16. Shoup, V.: NTL: a library for doing number theory [computer software] (version 11.4.3) (2020). <http://www.shoup.net/ntl/>. Accessed 18 Mar 2020
17. Siciliano, B., Khatib, O. (eds.): *Springer Handbook of Robotics*. SHB. Springer, Cham (2016). <https://doi.org/10.1007/978-3-319-32552-1>
18. Siciliano, B., Sciavicco, L., Villani, L., Oriolo, G.: *Springer. Robotics: Modelling, Planning and Control* (2008). <https://doi.org/10.1007/978-1-84628-642-1>
19. SymPy Development Team: Sympy 1.5.1 documentation: Solvers (2019). <https://docs.sympy.org/latest/modules/solvers/solvers.html>
20. Takayama, N.: Kan/sml: a system for computing in the ring of differential operators [computer software] (1991–2003), <http://www.math.kobe-u.ac.jp/KAN/>. Accessed 17 Mar 2020
21. The SciPy community: Numpy v1.18 manual: numpy.roots (2019). <https://numpy.org/doc/stable/reference/generated/numpy.roots.html>
22. Uchida, T., McPhee, J.: Triangularizing kinematic constraint equations using Gröbner bases for real-time dynamic simulation. *Multibody Syst. Dyn.* **25**, 335–356 (2011). <https://doi.org/10.1007/s11044-010-9241-8>
23. Uchida, T., McPhee, J.: Using Gröbner bases to generate efficient kinematic solutions for the dynamic simulation of multi-loop mechanisms. *Mech. Mach. Theory* **52**, 144–157 (2012). <https://doi.org/10.1016/j.mechmachtheory.2012.01.015>

24. van der Walt, S., Colbert, S.C., Varoquaux, G.: The numpy array: a structure for efficient numerical computation. *Comput. Sci. Eng.* **13**(2), 22–30 (2011). <https://doi.org/10.1109/MCSE.2011.37>
25. Verschelde, J.: Algorithm 795: PHCpack: a general-purpose solver for polynomial systems by homotopy continuation. *ACM Trans. Math. Softw.* **25**(2), 251–276 (1999). <https://doi.org/10.1145/317275.317286>
26. Wolfram Research Inc: Mathematica, Version 12.1 [computer software] (2020). <https://www.wolfram.com/mathematica>. Accessed 17 Mar 2020