

УДК 681.518

Моделирование роботов - манипуляторов с открытой и замкнутой кинематикой

Гудым Антон Васильевич

anton.gudym@gmail.com

Соколов Александр Павлович

alsokolo@bmstu.ru

МГТУ им. Н.Э. Баумана, Москва, Россия

Научные производственные технологии во многом зависят от использования робототехники. Фундаментом автоматизации производственных процессов являются многозвенные роботы-манипуляторы. Широкий спектр решаемых задач обуславливает значительное разнообразие моделей роботов и условий их применения. Авторы предлагают библиотеку с открытым исходным кодом (на языке Python), которая содержит алгоритмы для эффективного и наглядного моделирования различных конфигураций манипуляторов, а также решения обратной задачи кинематики с учетом ограничений, например, для замкнутых кинематических цепей, таких как гексаподы.

Ключевые слова: робот-манипулятор, моделирование, обратная кинематика, кинематическая цепь, методы оптимизации

Введение. Промышленное производство опирается на технологии многозвенных роботов-манипуляторов, рис. 1. Эти механизмы применяются для задач перемещения деталей или различных механических операций, таких как фрезерование, сверление или сварка в автоиндустрии или аэрокосмической отрасли. Качество подобных операций зависит от способности робота задавать положение выходного звена, устройства захвата, детали или инструмента. Положение манипулируемого тела относительно базы робота (или иной системы координат) задает кинематика робота. Позиция как самого тела, так и звеньев робота определяются шестью параметрами: тремя для вращения и тремя для позиции. В данной работе для представления вращения используется экспоненциальные координаты или элементы алгебры Ли [1].

В широко известных программных пакетах ROS [2] или GAZEBO [3] можно моделировать разнообразные робототехнические системы. Задание конфигурации выполняется с помощью XML-файлов (часто, достаточно объемных), но ввиду громоздкости данных комплексов, очень сложно применять различные ограничения, как например, замыкание кинематической цепи или ограничение рабочей зоны для обхода препятствий. Данная работа предлагает простой и эффективный программный интерфейс, а также комплекс алгоритмов, для задания конфигурации робота средствами языка программирования Python, и решения обратной кинематической задачи с ограничениями, например, для определения положения и управления гексаподом.

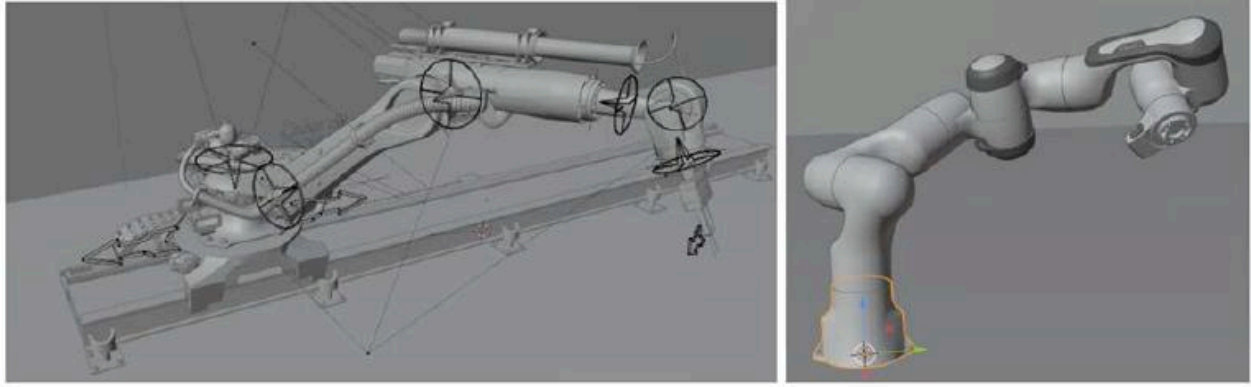


Рис. 1. KUKA KR210: шесть вращательных звеньев, два призматических (слева); Franka-Emika Panda: семь вращательных звеньев, коллаборативный робот (справа)

Материалы и методы. Удобным и эффективным способом описания положения тела в пространстве является матрица трансформации ${}^wT_o \in SE3 \subset R^{4 \times 4}$ вида

$${}^wT_o = \begin{bmatrix} r_{11} & r_{12} & r_{13} & tx \\ r_{21} & r_{22} & r_{23} & ty \\ r_{31} & r_{32} & r_{33} & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} {}^wR_o & {}^wt_o \\ 0 & 1 \end{bmatrix}, \quad (1)$$

где множество $SE3$ — евклидова группа.

Последовательность или цепочка матриц трансформаций (1) вместе с операцией умножения — универсальный и удобный способ задания положения звеньев или кинематической модели робота [4] (рис. 2):

$${}^{l_0}T_{l_n}(q) = {}^{l_0}T_{l_1}(q_1) {}^{l_1}T_{l_2}(q_2) \dots {}^{l_{n-1}}T_{l_n}(q_n), \quad (2)$$

где $q \in R^n$ — степени свободы механизма; ${}^{l_0}T_{l_n}(q)$ — позиция выходного (последнего) звена l_n относительно неподвижной базы l_0 , определяется умножением локальных трансформаций (приращений) ${}^{l_{i-1}}T_{l_i}(q_i)$ — положения звена l_i относительно l_{i-1} . В (2), для простоты, предполагается, что каждое звено робота обладает одной степенью свободы, если это не так, то всегда можно разложить звено на два и более «виртуальных звеньев» с одной степенью свободы.

Трансформации звеньев робота относительно друг друга ${}^{l_{i-1}}T_{l_i}(q_i)$, в свою очередь, раскладываются на цепочки элементарных преобразований сдвига или поворота (часто определяется ДН-параметрами [5]). К примеру, на рис. 2 величина q_i задает угол поворота вокруг оси OZ звена l_{i-1} , а $c = 5$ — статический параметр, определяет сдвиг

$${}^{l_{i-1}}T_{l_i}(q_i) = \begin{bmatrix} \cos(q_i) & -\sin(q_i) & 0 & 0 \\ \sin(q_i) & \cos(q_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & c \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

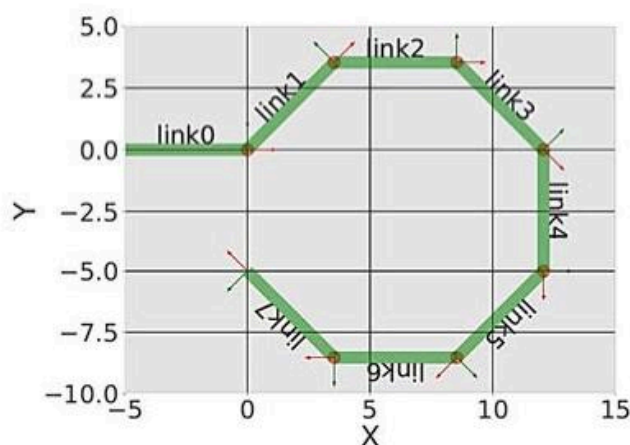


Рис. 2. Схематичный 2D-робот с открытой кинематической цепью на плоскости XY

Результаты. На рис. 4 представлен краткий пример кода необходимого для построения и отображения гексапода. Реализованный в данной работе программно-алгоритмический комплекс позволяет средствами объектно-ориентированного языка программирования Python решать следующие задачи:

- формулировать конфигурацию механизма средствами ООП;
- выполнять визуализацию моделируемого робота в программных средах (Bledner3D рис. 3, matplotlib — рис. 2);
- решать оптимизационную задачу обратной кинематики с ограничениями (например, учитывая замыкание кинематической цепи (2)) в реальном времени (свыше 100 Гц).

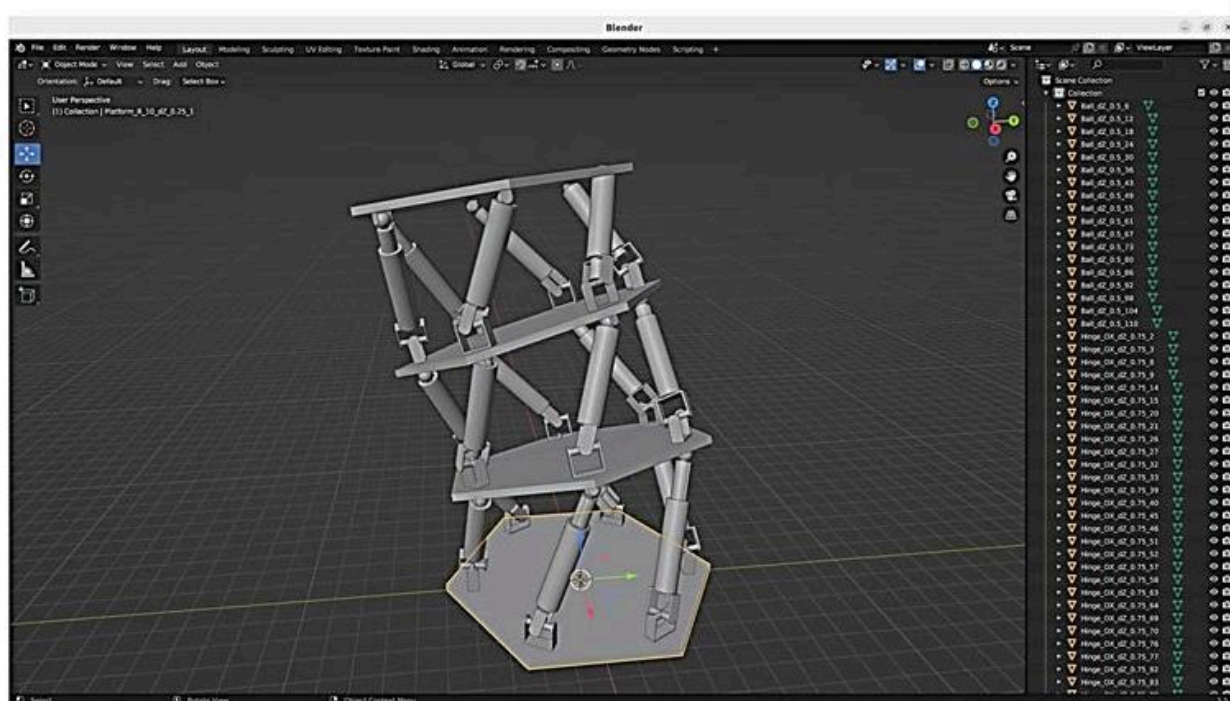


Рис. 3. Гексапод с замкнутой кинематической цепью, с динамической визуализацией в открытом ПО для 3D-дизайна и моделирования Blender3D

Эффективное решение приведенных задач возможно благодаря использованию Python-библиотеки для машинного обучения и оптимизации JAX с возможностью выполнения вычислений на GPU.

```

1 # os.path.abspath is the path to the current folder.
2 mesh_root = os.path.abspath("data/meshes/hexapod/").
3
4 # Главные размеры основных деталей - платформы, шарниров, поршней и пр.
5 base_height = 0.25
6 hinge_height = 0.75
7 piston_height = 3
8 hinge_ball_height = 0.5
9
10 # Полигональные сетки соответствующих деталей.
11 platform_path = os.path.join(mesh_root, "Platform_R_10_dZ_0.25.dae")
12 hinge_path = os.path.join(mesh_root, "Hinge_0X_dZ_0.75.dae")
13 sleeve_path = os.path.join(mesh_root, "Sleeve_0Z_3.dae")
14 piston_path = os.path.join(mesh_root, "Piston_0Z_3.dae")
15 ball_path = os.path.join(mesh_root, "Ball_dZ_0.5.dae")
16 hinge_ball_path = os.path.join(mesh_root, "HingeBall_0X_dZ_0.5.dae")
17
18 # Функция создания призматического шарнира
19 def create_piston_chain(link_parents, ets_parents):
20     l = LinkBlenderDrawing3D(hinge_path, link_parents=link_parents,
21                               ets_l2ps=ets_parents)
22     l = LinkBlenderDrawing3D(None, link_parents=l, #q0_l2ps=pi/4,
23                               ets_l2ps=ET.tz(hinge_height) * ET.Rx())
24     l = LinkBlenderDrawing3D(None, link_parents=l, #q0_l2ps=pi/4,
25                               ets_l2ps=ET.Rz(pi/2) * ET.Rx())
26     l = LinkBlenderDrawing3D(hinge_path, link_parents=l,
27                               ets_l2ps=ET.tz(hinge_height) * ET.Rx(pi))
28     l = LinkBlenderDrawing3D(sleeve_path, link_parents=l,
29                               ets_l2ps=ET.Rx(pi))
30     l = LinkBlenderDrawing3D(piston_path, link_parents=l, #q0_l2ps=0.9 * piston_height,
31                               ets_l2ps=ET.tz())
32     l = LinkBlenderDrawing3D(ball_path, link_parents=l,
33                               ets_l2ps=ET.tz(piston_height + hinge_ball_height) * ET.Rx(pi))
34     l = LinkBlenderDrawing3D(hinge_ball_path, link_parents=l, #q0_l2ps=[pi/4, pi/4, 0],
35                               ets_l2ps=ET.tz(hinge_ball_height/2) * ET.Rx() * ET.Ry() * ET.Rz() * ET.tz(-hinge_ball_height/2))
36     return l
37
38 # Функция задания одной платформы и прикрепленных шарниров (секция)
39 def create_hex_section(link_platform: LinkBlenderDrawing3D, num_piston_chains: int):
40     pistons_placement_radius = 4
41     angular_step = 2 * pi / num_piston_chains
42
43     links_out = []
44     for i in range(num_piston_chains):
45         ets = ET.tz(base_height) * ET.Rz(angular_step * i) * ET.ty(pistons_placement_radius)
46         links_out.append(create_piston_chain(link_platform, ets))
47     return links_out
48
49 # Создание полного механизма - соединение нескольких секций
50 num_sections = 3
51 # количество призматических шарниров в секции
52 num_piston_chains = 6
53 link_platform = LinkBlenderDrawing3D(platform_path)
54 for section_id in range(num_sections):
55     links_out = create_hex_section(link_platform, num_piston_chains)
56
57     ets_platform_to_chains = []
58     for i in range(num_piston_chains):
59         ets_parent_to_world = links_out[i].get_orientation()
60         if i == 0:
61             # Use first chain for reference orientation (Z-height)
62             ets_platform_to_world = ET.tz(ets_parent_to_world[2,3])
63
64         ets_platform_to_chains.append(
65             ET.SE3(inv(ets_parent_to_world) * ets_platform_to_world))
66
67     link_platform = LinkBlenderDrawing3D(platform_path, link_parents=links_out,
68                                           ets_l2ps=ets_platform_to_chains)
69
70 link_platform.init_transform()

```

Рис. 4. Листинг Python-кода для построения гексапода

Заключение. В рамках данной работы был реализован комплекс алгоритмов, позволяющих эффективно моделировать робототехнические системы, а также выполнять удобную визуализацию полученных результатов.

В дальнейшем планируется использование данной системы для исследований в области высокоточной кинематической калибровки манипуляторов.

СПИСОК ИСТОЧНИКОВ

- [1] Solà J., Deray J., Atchuthan D. A micro Lie theory for state estimation in robotics. Retrieved from (2018). URL: <https://arxiv.org/abs/1812.01537v4> (accessed October 9, 2024).
- [2] ROS — Robot Operating System. URL: <https://ros.org> (accessed October 9, 2024).
- [3] Koenig N., Howard A. Design and use paradigms for Gazebo, an open-source multi-robot simulator. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2004, vol. 3, pp. 2149–2154. <https://doi.org/10.1109/IROS.2004.1389727>
- [4] Haviland J., Corke P. Manipulator Differential Kinematics: Part I: Kinematics, Velocity, and Applications. IEEE Robotics and Automation Magazine, 2023, pp. 2–11. <https://doi.org/10.1109/MRA.2023.3270228>
- [5] Corke P. A Simple and Systematic Approach to Assigning Denavit — Hartenberg Parameters. IEEE Transactions on Robotics, 2007, vol. 23, pp. 590–594. <https://doi.org/10.1109/TRO.2007.896765>

Algorithms for metrological photogrammetry

Gudym Anton V. anton.gudym@gmail.com

Sokolov Alexandr P. alsokolo@bmstu.ru

BMSTU, Moscow, Russia

Hightech industrial technologies largely rely on the use of robotics. The foundation of automation in production processes lies in multi-link robotic manipulators. The wide range of tasks addressed leads to significant diversity in robot models and their application conditions. The authors present an open-source library (written in Python) that includes algorithms for efficient and intuitive modeling of various manipulator configurations, as well as solving the inverse kinematics problem with constraints, for example, for closed kinematic chains such as hexapods.

Keywords: robotic arm, kinematics, computer vision, photogrammetry, non-linear optimization