САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА МЕХАНИКИ УПРАВЛЯЕМОГО ДВИЖЕНИЯ

Поляков Юрий Андреевич

Выпускная квалификационная работа бакалавра

Управление движением инструмента робота-манипулятора

Направление 01.03.02

Прикладная математика, фундаментальная информатика и программирование

Научный руководитель:
кандидат физ.-мат. наук,
доцент
Шиманчук Д.В.

Рецензент:
кандидат физ.-мат. наук,
доцент
Шмыров В.А.

Санкт-Петербург

2020

Оглавление

Введение

Стационарные роботы-манипуляторы составляют большую часть коммерческих робототехнических систем. Роботы-манипуляторы традиционно применяются, например, для точечной сварки и окраски методом распыления в автомобильной промышленности, для фасовки и упаковки продукции в химической и фармацевтической промышленности. Они могут выполнять самую разную работу: от простых повторяющихся действий до сборки сложных сенсорных устройств.

Ведущие производители промышленных роботов [6]:

1. FANUC Robotics, Япония. Всего в мире можно насчитать свыше 200 000 роботов FANUC, 30 000 из которых находятся в Европе и России.
2. KUKA (Keller und Knappich Augsburg), Германия. Роботов KUKA используют во всем мире на заводах: для операций по сварке, погрузке, паллетизации, упаковке, обработке, сборке и др.
3. ABB (Asea Brown Boveri Ltd.), Швеция, Швейцария. Производит промышленных роботов, специальное оборудование, инструменты и программное обеспечение.
4. Kawasaki, Япония. В линейку входят манипуляторы специального взрывобезопасного исполнения, роботы, трудящиеся в агрессивных средах, конструкции для металлургических производств, а также паллетайзеры.
5. Motoman (Yaskawa), Япония, США. Модельный ряд состоит из 175 роботизированных моделей и 40 полностью интегрированных готовых решений, применимых для специфических задач (в том числе оборудование для безопасности).
6. OTC Daihen, Япония. Роботы используются для разных видов сварки и плазменной резки (в частности мягкой и нержавеющей стали, алюминия, титана, других экзотических металлов).
7. Panasonic, Япония. Компания выпускает универсальные манипуляторы для многих видов производственных задач.

8. KC Robotics, Inc, США. Предприятие обслуживает все отрасли использования промышленных роботов, а также занимается производством и обработкой материалов, включая пакетирование и сварочные работы.

9. Triton Manufacturing, США. Сфера деятельности — гибкие системы питания, а также пользовательские обработанные шины и паяные электрические компоненты.

10. Kaman Corporation, США. Компания производит подшипники, механические и электрические устройства для электропередачи и управления движением, обработки материалов и жидкостей, а также другие устройства, применяемые в промышленной и военной робототехнике.

Рис. 1. Робот-манипулятор OWI-535 и робот-сварщик ТВ-1400 [5]

За счет применения промышленных роботов работа становится эффективнее, качественнее и быстрее. Примеры подобных роботов представлены на рис. 1.

Зачастую важным вопросом при принятии решения о внедрении робота на производство становится оценка времени, затрачиваемого роботом на выполнение поставленной задачи, по сравнению с временем, которое тратит человек.

Обзор источников

[1-4] В основе конструирования роботов-манипуляторов лежат кинематические законы, неотъемлемой частью которых являются однородные координаты.

В традиционной трёхмерной геометрии операции перемещения или параллельного переноса (*Trans*) описываются сложением векторов размерности 3x1, а операции поворота (*Rot*) – умножением матриц размерности 3x3. Этого достаточно для решения простых задач, но при конструировании манипулятора нам приходится оперировать более длинными цепочками преобразований, в результате которых, например, начальная точка p перейдёт в точку

$$p' = \Big(\Big(\big(p + Trans(x_1, y_1, z_1)\big)\, Rot(x, 90)\Big) + Trans(x_2, y_2, z_2)\Big)\, Rot(x, -45).$$

С помощью стандартных операций трёхмерной геометрии мы не можем упростить это выражение так, чтобы можно было перевести точку p в p', применив к ней единственную операцию. А если таких точек достаточно много, то вычислительные затраты могут оказаться слишком большими.

Проблема решается с помощью однородных координат [Möbius, 1827]. Однородные координаты дополняют операции перемещения и вращения четвёртой координатой (масштабным коэффициентом), значение которой можно принять равной 1. При этом формат преобразований для обеих операций сводится к матрицам размерности 4x4.

Каждое из таких однородных преобразований состоит из вращательной и трансляционной частей. Любая часть может быть эквивалентной тождественному преобразованию. Матрицы стандартных операций выглядят следующим образом [4]:

$$Trans(v_x, v_y, v_z) = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, Rot(x, \theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$Rot(y,\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, Rot(z,\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Любая цепочка преобразований описывается произведением таких матриц. Кроме того, матрицами 4x4 удобно описывать положение и ориентацию объекта (например, руки робота) в трёхмерном пространстве.

Любой манипулятор состоит из нескольких *звеньев* и *сочленений* между ними, приводимых в движение силовыми приводами. Типичный манипулятор имеет основание, рабочий орган и 6 соединительных элементов между ними.

Каждое сочленение называется *степенью свободы* [1], так что такой манипулятор имеет 6 степеней свободы. Такая конструкция позволяет манипулятору принимать любое возможное положение и ориентацию в трёхмерном пространстве.

Существует два базовых вида сочленений – телескопическое и вращательное. Более сложные сочленения (например, шаровой шарнир) можно представить как комбинацию двух или более базовых сочленений. [3]

Кинематика манипулятора отвечает на два основных вопроса, связанных с особенностями геометрии манипулятора [1-4]:

1. Какими будут конечные положение и ориентация инструмента манипулятора при заданных углах в сочленениях? Ответ на этот вопрос даёт решение *прямой задачи кинематики*.
2. Какими должны быть углы в сочленениях, чтобы рабочий орган принял требуемые положение и ориентацию? Ответ на этот вопрос даёт решение *обратной задачи кинематики*.

Прямая задача кинематики описывает преобразования манипулятора, начинающиеся с его основания и проходящие через его сочленения к рабочему органу. Каждое преобразование от сочленения к сочленению можно описать однородной матрицей размерности 4x4. Далее все матрицы (для 6-звенного манипулятора – 6 матриц) перемножаются, чтобы в результате получить *одну*

матрицу 4x4, описывающую полное преобразование манипулятора от основания к рабочему органу.

Каждый переход от одного сочленения к другому можно описать своей матрицей 4x4. Но Денавит и Хартенберг разработали систему обозначений, которая позволяет стандартным способом описать *любую* конфигурацию сочленений с помощью 4 преобразований – 2 перемещений и 2 поворотов [2].

При переходе от (i-1)-ого сочленения к i-ому предполагается, что ось вращательного сочленения совпадает с локальной осью Z. Тогда два перемещения и два поворота следует осуществлять так:

1. $Rot(z_{i-1}, q_i)$ – поворот вокруг $O_{i-1}Z_{i-1}$ на угол q_i => $O_{i-1}X_{i-1} || O_iX_i$.
2. $\mathrm{T}rans(0,0,d_i)$ –по $O_{i-1}Z_{i-1}$ на d_i => $O_{i-1}X_{i-1}$ и $O_iX_i \in$ одной прямой.
3. $\mathrm{T}rans(a_i,0,0)$ – по $O_{i-1}X_{i-1}$ на a_i => $O_{i-1} = O_i$.
4. $Rot(x_{i-1}, \alpha_i)$ – поворот вокруг $O_{i-1}X_{i-1}$ на угол α_i => $O_{i-1}Y_{i-1} = O_iY_i$.

Таким образом, переход от одного сочленения к следующему соответствует произведению четырёх отдельных базовых преобразований и может быть описан одной матрицей 4x4. Преобразование T, переводящее сочленение $i-1$ в сочленение i, выглядит следующим образом [4]:

$$
{}^{i-1}_{i}T = Rot(z_{i-1}, q_i) * Trans(0,0,d_i) * Trans(a_i,0,0) * Rot(x_{i-1}, \alpha_i) =
$$

$$
= \begin{bmatrix} \cos(q_i) & -\cos(\alpha_i)\sin(q_i) & \sin(\alpha_i)\sin(q_i) & a_i\cos(q_i) \\ \sin(q_i) & \cos(\alpha_i)\cos(q_i) & -\sin(\alpha_i)\cos(q_i) & a_i\sin(q_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}.
$$

Обратная задача кинематики описывает преобразования манипулятора, начинающиеся от его рабочего органа и проходящие через его сочленения к основанию. Другими словами, требуется найти обобщённые координаты, приводящие инструмент в заданное положение и ориентацию.

Решение обратной задачи для N-звенного робота-манипулятора сводится к решению нелинейной тригонометрической системы уравнений.

Обозначим $Q = q_1 \times q_2 \times q_3$ – пространство состояний системы. Пусть в момент времени t_0 манипулятор находится в точке $\boldsymbol{q}_0$. Необходимо определить функцию $\boldsymbol{q}(t)$: $\boldsymbol{q} = \boldsymbol{q}(t), t \in [t_0, t_1], \boldsymbol{q}(t_0) = \boldsymbol{q}_0, \boldsymbol{q}(t_1) = \boldsymbol{q}_1, q \in Q$. [3]

Задача имеет бесконечное множество решений, если не ввести дополнительных ограничений.

Рассмотрим в пространстве Q движение по прямой [4]:

$$\frac{\boldsymbol{q} - \boldsymbol{q}_0}{t - t_0} = \frac{\boldsymbol{q}_1 - \boldsymbol{q}_0}{t_1 - t_0}$$

или

$$\boldsymbol{q} = \boldsymbol{q}(t) = \boldsymbol{q}_0 + \boldsymbol{v}(t - t_0), \text{где } \boldsymbol{v} = \frac{\boldsymbol{q}_1 - \boldsymbol{q}_0}{t_1 - t_0}.$$

Естественно, $|v_i| \leq (\dot{q}_i)_{max}$, и если это не выполняется, то промежуток времени $t_1 - t_0$ увеличивают. Эту особенность решения задач планирования обходят добавлением условия $\max\limits_{t \in [t_0, t_1]} |\dot{q}_i| = c_i, i = 1, 2, \ldots, N$, где **c** – постоянный вектор, а момент времени t_1 является дополнительной неизвестной, определяемой в процессе решения задачи [2].

При равномерной параметризации перемещения в точках $\boldsymbol{q}_0$ и $\boldsymbol{q}_1$ скорость и ускорение терпят разрывы первого и второго рода соответственно, что предполагает необходимость прикладывать к сочленениям неограниченные силы или моменты [3].

Постановка задачи

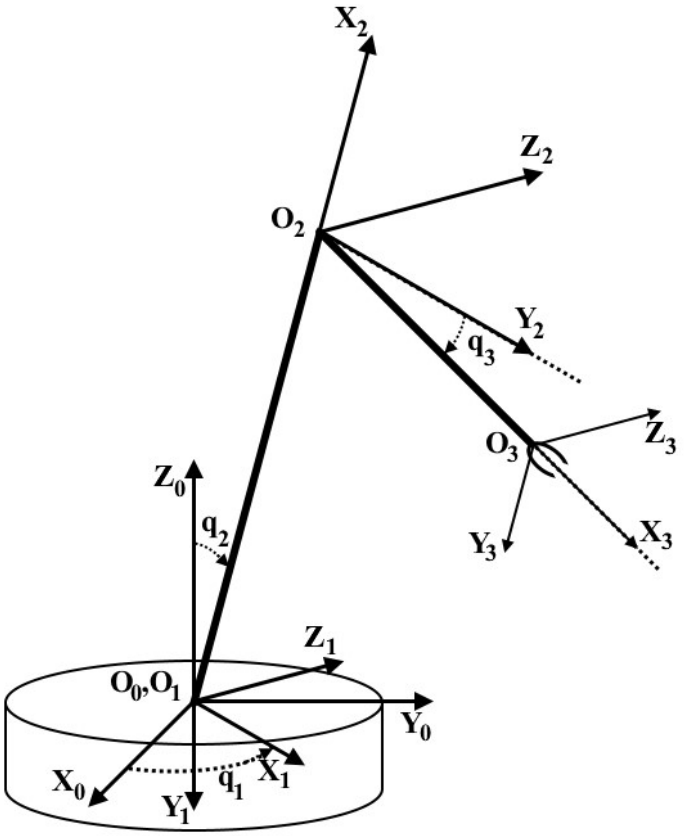

Рис. 2. Кинематическая схема робота-манипулятора

Рассмотрим робот-манипулятор, схема которого представлена на рис. 2. Система $O_0X_0Y_0Z_0$ – система координат основания. Звено O_1O_2 связано с основанием шаровым шарниром, способным вращаться по двум осям: вращение вокруг вертикальной оси Z_0 и наклон в сторону от неё. На схеме данные вращения заданы как два «сочленения», центры вращения которых – точки O_0 и O_1 – совпадают.

Обобщёнными координатами перехода из 0-й в 1-ю и из 1-ой во 2-ую систему координат становятся углы q_1 поворота вокруг оси O_0Z_0 и q_2 – вокруг оси O_1Z_1, лежащей в плоскости $O_0X_0Y_0$. Обобщённая координата перехода из 2-ой системы координат в 3-ю – угол q_3 поворота звена O_2O_3 вокруг оси O_2Z_2, перпендикулярной плоскости $O_1O_2O_3$. При $q_3 = 0$ звенья O_1O_2 и O_2O_3 перпендикулярны. Длины звеньев O_1O_2 и O_2O_3 заданы, постоянны и равны соответственно l_1 и l_2.

Требуется решить прямую и обратную задачи кинематики для данного манипулятора: найти выражения для координат (p_1, p_2, p_3) инструмента в системе $O_0X_0Y_0Z_0$ через обобщённые координаты (q_1, q_2, q_3) и выражения обобщённых координат через координаты положения инструмента. Считается, что решение обратной задачи существует для всех целевых точек, входящих в рабочую область.

Наличие только трёх звеньев позволяет управлять только положением инструмента, повлиять на положение и ориентацию в пространстве одновременно возможность отсутствует.

Ставятся два класса задач:

1. Задача «сбора». Имея координаты начальной точки, рассматриваемой как контейнер, и набор координат целевых точек, рассматриваемых как объекты, требуется определить максимальное количество объектов, которые можно подобрать **и** сложить в контейнер за заданное время.
2. Задача «посещения». Имея координаты начальной точки и набор координат целевых точек, требуется определить, через какое максимальное количество целевых точек можно успеть провести инструмент за заданное время.

Целью данной работы является написание пакета программного обеспечения, способного выполнить численную оценку результатов выполнения данных задач.

Решение задач кинематики робота-манипулятора

Решение прямой задачи о положении

Для начала по алгоритму Денавита-Хартенберга найдём матрицы $A_i = T_i^1 T_i^2 T_i^3 T_i^4$ перехода от $i-1$ к i системе координат, где

$$T_i^1 = \begin{bmatrix} \cos(q_i) & -\sin(q_i) & 0 & 0 \\ \sin(q_i) & \cos(q_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, T_i^2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$T_i^3 = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, T_i^4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha_i) & -\sin(\alpha_i) & 0 \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Исходя из кинематической схемы, $d_1 = d_2 = d_3 = a_1 = \alpha_2 = \alpha_3 = 0$, поэтому $T_i^2 = T_1^3 = T_2^4 = T_3^4 = E$ и матрицы A_i принимают следующий вид:

$$A_1 = T_1^1 T_1^4 = \begin{bmatrix} \cos(q_1) & -\sin(q_1) & 0 & 0 \\ \sin(q_1) & \cos(q_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha_1) & -\sin(\alpha_1) & 0 \\ 0 & \sin(\alpha_1) & \cos(\alpha_1) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$A_2 = T_2^1 T_2^3 = \begin{bmatrix} \cos(q_2) & -\sin(q_2) & 0 & 0 \\ \sin(q_2) & \cos(q_2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$A_3 = T_3^1 T_3^3 = \begin{bmatrix} \cos(q_3) & -\sin(q_3) & 0 & 0 \\ \sin(q_3) & \cos(q_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

и, учитывая $\alpha_1 = \frac{\pi}{2}, q_2 = q_2 - \frac{\pi}{2}, a_2 = l_1, q_3 = q_3 + \frac{\pi}{2}, a_3 = l_2$, получаем

$$A_1 = \begin{bmatrix} \cos(q_1) & 0 & -\sin(q_1) & 0 \\ \sin(q_1) & 0 & \cos(q_1) & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$A_2 = \begin{bmatrix} \sin(q_2) & \cos(q_2) & 0 & l_1 \sin(q_2) \\ -\cos(q_2) & \sin(q_2) & 0 & -l_1 \cos(q_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$A_3 = \begin{bmatrix} -\sin(q_3) & -\cos(q_3) & 0 & -l_2 \sin(q_3) \\ \cos(q_3) & -\sin(q_3) & 0 & l_2 \cos(q_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Для решения прямой задачи достаточно найти матрицу

$$T_3 = A_1 A_2 A_3 = \begin{bmatrix} R^* & p^* \\ 0 & 1 \end{bmatrix}.$$

Получаем

$$R^* = \begin{bmatrix} \cos(q_1)\cos(q_2+q_3) & -\cos(q_1)\sin(q_2+q_3) & -\sin(q_1) \\ \sin(q_1)\cos(q_2+q_3) & -\sin(q_1)\sin(q_2+q_3) & \cos(q_1) \\ -\sin(q_2+q_3) & -\cos(q_2+q_3) & 0 \end{bmatrix},$$

$$p^* = \begin{bmatrix} \cos(q_1)\,(l_1\sin(q_2) + l_2\cos(q_2+q_3)) \\ \sin(q_1)\,(l_1\sin(q_2) + l_2\cos(q_2+\ q_3)) \\ l_1\cos(q_2) - l_2\sin(q_2+q_3) \end{bmatrix}.$$

Вектор p^* содержит решение прямой задачи о положении.

Решение обратной задачи о положении

Для дальнейших вычислений нам понадобятся матрицы, обратные к матрицам A_i, имеющие следующий вид [4]:

$$A^{-1} = \begin{bmatrix} R^T & -R^T\boldsymbol{p} \\ 0 & 1 \end{bmatrix}, \text{где } R = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}, \boldsymbol{p} = \begin{bmatrix} A_{14} \\ A_{24} \\ A_{34} \end{bmatrix}.$$

Откуда получаем:

$$A_1^{-1} = \begin{bmatrix} \cos(q_1) & \sin(q_1) & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -\sin(q_1) & \cos(q_1) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$A_2^{-1} = \begin{bmatrix} \sin(q_2) & -\cos(q_2) & 0 & -l_1 \\ \cos(q_2) & \sin(q_2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$A_3^{-1} = \begin{bmatrix} -\sin(q_3) & \cos(q_3) & 0 & -l_2 \\ -\cos(q_3) & -\sin(q_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Составим уравнение $A_1^{-1}T_3 = A_2A_3$:

$$A_2A_3 = \begin{bmatrix} \cos(q_2+q_3) & -\sin(q_2+q_3) & 0 & l_2\cos(q_2+q_3) + l_1\sin(q_2) \\ \sin(q_2+q_3) & \cos(q_2+q_3) & 0 & l_2\sin(q_2+q_3) - l_1\cos(q_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

«Развернём» матрицу A_2A_3 так, чтобы ось Z была направлена вверх:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} A_2A_3 = \begin{bmatrix} \cos(q_2+q_3) & -\sin(q_2+q_3) & 0 & l_1\sin(q_2) + l_2\cos(q_2+q_3) \\ 0 & 0 & 1 & 0 \\ -\sin(q_2+q_3) & -\cos(q_2+q_3) & 0 & l_1\cos(q_2) - l_2\sin(q_2+q_3) \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Результат интересен не только тем, что наглядно показывает зависимость координат X и Z схвата в абсолютной системе координат от углов q_2 и q_3 при $q_1 = 0$, но и совпадает с матрицей $T_3(0, q_2, q_3)$, что подтверждает, что она посчитана правильно.

Пусть задана матрица $T_3 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_1 \\ r_{21} & r_{22} & r_{23} & p_2 \\ r_{31} & r_{32} & r_{33} & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, описывающая желаемые положение и ориентацию инструмента. Домножим её слева на A_1^{-1}:

$$A_1^{-1}T_3 = \begin{bmatrix} \cos(q_1) & \sin(q_1) & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -\sin(q_1) & \cos(q_1) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_1 \\ r_{21} & r_{22} & r_{23} & p_2 \\ r_{31} & r_{32} & r_{33} & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Приравняв элементы $(4,1)$, $(4,2)$, $(4,3)$ последнего матричного равенства, получаем уравнения:

$$\begin{cases} l_2 \cos(q_2 + q_3) + l_1 \sin(q_2) = p_2 \sin(q_1) + p_1 \cos(q_1), \\ l_2 \sin(q_2 + q_3) - l_1 \cos(q_2) = -p_3, \\ 0 = p_2 \cos(q_1) - p_1 \sin(q_1). \end{cases}$$

Из третьего уравнения можно выразить $\tan(q_1) = \frac{p_2}{p_1}$, и из первого найти

$$p_2 \sin(q_1) + p_1 \cos(q_1) \tan(q_1) \frac{p_1}{p_2} = \left(p_2 + \frac{p_1^2}{p_2} \right) \sin(q_1),$$

$$\sin(q_1) = \frac{p_2 (l_2 \cos(q_2 + q_3) + l_1 \sin(q_2))}{p_1^2 + p_2^2}.$$

Из геометрии задачи следует, что

$$\sin(q_1) = \frac{p_2}{\sqrt{p_1^2 + p_2^2}},$$

Отсюда

$$l_1 \sin(q_2) + l_2 \cos(q_2 + q_3) = \sqrt{p_1^2 + p_2^2}.$$

Получаем систему уравнений:

$$\begin{cases} l_1 \sin(q_2) + l_2 \cos(q_2 + q_3) = \sqrt{p_1^2 + p_2^2}, \\ l_1 \cos(q_2) - l_2 \sin(q_2 + q_3) = p_3, \\ \tan(q_1) = \frac{p_2}{p_1}. \end{cases}$$

Теперь похожим образом выразим элементы четвёртого столбца из $A_3 = A_2^{-1}A_1^{-1}T_3$. Получим следующую систему уравнений:

$$\begin{cases} l_1 - p_1\cos(q_1)\sin(q_2) - p_2\sin(q_1)\sin(q_2) - p_3\cos(q_2) = l_2\sin(q_3), \\ p_1\cos(q_1)\cos(q_2) + p_2\sin(q_1)\cos(q_2) - p_3\sin(q_2) = l_2\cos(q_3), \\ p_2\cos(q_1) - p_1\sin(q_1) = 0. \end{cases}$$

Подставим третье уравнение этой системы (совпадающее с третьим уравнением предыдущей) в первое уравнение:

$$p_1\cos(q_1)\sin(q_2)\tan(q_1)\frac{p_1}{p_2} = \frac{p_1^2}{p_2}\sin(q_1)\sin(q_2),$$

$$p_2\sin(q_1)\sin(q_2)\cot(q_1)\frac{p_2}{p_1} = \frac{p_2^2}{p_1}\cos(q_1)\sin(q_2),$$

$$\sin(q_2)\left(\frac{p_1^2}{p_2}\sin(q_1) + \frac{p_2^2}{p_1}\cos(q_1)\right) = [p_2\cos(q_1) = p_1\sin(q_1)] =$$

$$= \sin(q_2)\sin(q_1)\left(\frac{p_1^2}{p_2} + p_2\right) = \left[\sin(q_1) = \frac{p_2}{\sqrt{p_1^2 + p_2^2}}\right] =$$

$$= \sin(q_2)\sqrt{p_1^2 + p_2^2}\,\frac{p_2}{p_1^2 + p_2^2}\,\frac{p_1^2 + p_2^2}{p_2} = \sin(q_2)\sqrt{p_1^2 + p_2^2},$$

и во второе:

$$p_1\cos(q_1)\cos(q_2) + p_2\sin(q_1)\cos(q_2) = \cos(q_2)\sqrt{p_1^2 + p_2^2}.$$

Система приобретает следующий вид:

$$\begin{cases} l_1 - \sin(q_2)\sqrt{p_1^2 + p_2^2} - p_3\cos(q_2) = l_2\sin(q_3), \\ \cos(q_2)\sqrt{p_1^2 + p_2^2} - p_3\sin(q_2) = l_2\cos(q_3), \\ \tan(q_1) = \dfrac{p_2}{p_1}. \end{cases}$$

Из геометрических соображений (тех же, что позволили выяснить значение $\sin(q_1)$), можно выразить q_3. Для этого воспользуемся теоремой косинусов, треугольник $O_1O_2O_3$ имеет стороны l_1, l_2 и $\sqrt{p_1^2+p_2^2+p_3^2}$, а угол $O_1O_2O_3$ равен $\frac{\pi}{2}-q_3$:

$$\cos\left(\frac{\pi}{2}-q_3\right)=\sin(q_3)=\frac{l_1^2+l_2^2-p_1^2-p_2^2-p_3^2}{2l_1l_2}.$$

Похожим образом можно выразить q_2. Пусть β – угол $O_2O_1O_3$. По теореме косинусов:

$$\cos(\beta)=\frac{l_1^2+p_1^2+p_2^2+p_3^2-l_2^2}{2l_1\sqrt{p_1^2+p_2^2+p_3^2}}.$$

Далее можно увидеть, что

$$O_1O_3\sin(q_2+\beta)=\sqrt{p_1^2+p_2^2},$$
$$O_1O_3\cos(q_2+\beta)=p_3$$

откуда $q_2+\beta=\operatorname{atan2}\left(\sqrt{p_1^2+p_2^2},p_3\right)$, а значит

$$q_2=\operatorname{atan2}\left(\sqrt{p_1^2+p_2^2},p_3\right)-\arccos\left(\frac{l_1^2+p_1^2+p_2^2+p_3^2-l_2^2}{2l_1\sqrt{p_1^2+p_2^2+p_3^2}}\right).$$

В итоге, решение обратной задачи о положении принимает вид:

$$q_1=\operatorname{atan2}(p_2,p_1),$$
$$q_2=\operatorname{atan2}\left(\sqrt{p_1^2+p_2^2},p_3\right)-\arccos\left(\frac{l_1^2+p_1^2+p_2^2+p_3^2-l_2^2}{2l_1\sqrt{p_1^2+p_2^2+p_3^2}}\right),$$
$$q_3=\arcsin\left(\frac{l_1^2+l_2^2-p_1^2-p_2^2-p_3^2}{2l_1l_2}\right).$$

Определение границ рабочей области

Внешние границы рабочей области определяются двумя простыми условиями:

1. Робот должен быть способен дотянуться до целевой точки, полностью выпрямив манипулятор.
2. Робот не должен повредить покрытие на полу.

Соответственно,

1. $p_1^2 + p_2^2 + p_3^2 \leq (l_1 + l_2)^2$
   $p_3 \geq 0$
2. $l_1 \cos(q_2) \geq 0 \overset{l_1 \geq 0}{\Longrightarrow} \cos(q_2) \geq 0$

Однако этих условий достаточно только при $l_1 = l_2$.

В случае, когда второе звено короче – $l_1 > l_2$ – робот не может достать до точки O_0, даже если полностью сложится – то есть даже при $q_3 = \frac{\pi}{2}$ – инструмент окажется от этой точки на расстоянии $l_1 - l_2$, а значит, $p_1^2 + p_2^2 + p_3^2 \geq (l_1 - l_2)^2$.

Если же второе звено длиннее первого – $l_2 > l_1$ – то без ограничения $\cos(q_2) \geq 0$ внутренняя граница имела бы форму такой же сферы радиусом $l_2 - l_1$ с центром в точке O_0. При введении этого ограничения точка O_2 – точка соединения звеньев – не может опуститься ниже поверхности пола – иначе угол q_2 будет больше $\frac{\pi}{2}$, т.е. $\cos(q_2) < 0$.

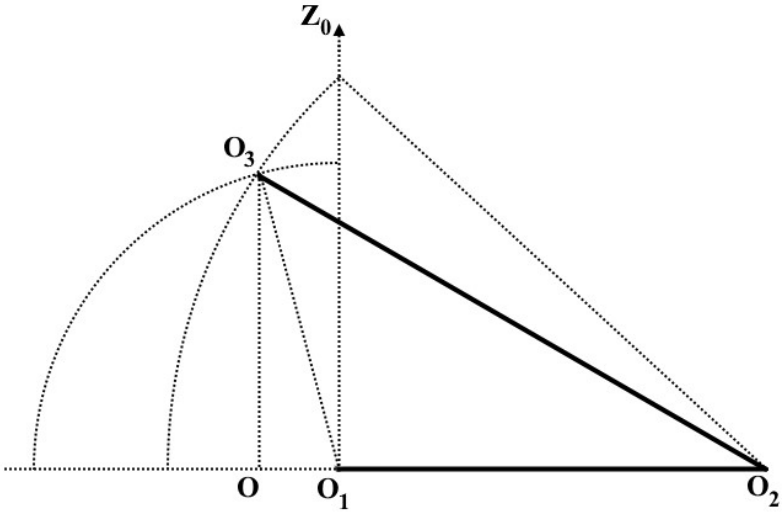

Рис. 3. Иллюстрация к описанию внутренней границы рабочей области

В этом случае, рассматривая ситуацию в плоскости звеньев робота, имеем картину, представленную на рис. 3.

Робот вращается вокруг точки O_1. При повороте звена O_2O_3 вокруг зафиксированной точки O_2, точка O_3 будет двигаться вдоль большей окружности. При фиксации угла $O_1O_2O_3$ и вращении вокруг точки O_1, точка O_3 будет двигаться вдоль малой окружности. Обе окружности рассматриваем до их пересечения с горизонталью (нижняя граница области) и осью Z_0, являющейся осью симметрии рабочей области.

При фиксированном расстоянии $O_1O_3 = \sqrt{p_1^2 + p_2^2 + p_3^2}$, вращение вокруг точки O_1 влияет только на угол OO_1O_3, который в сумме с углом $O_2O_1O_3$ не должен превысить π.

Таким образом, $O_2O_1O_3 + OO_1O_3 \leq \pi$. С одной стороны, мы можем вычислить

$$\cos(OO_1O_3) = \frac{\sqrt{p_1^2 + p_2^2}}{\sqrt{p_1^2 + p_2^2 + p_3^2}},$$

с другой – по теореме косинусов

$$\cos(O_2O_1O_3) = \frac{p_1^2 + p_2^2 + p_3^2 + l_1^2 - l_2^2}{2l_1\sqrt{p_1^2 + p_2^2 + p_3^2}}.$$

Пограничное значение достигается при $\cos(OO_1O_3) = \cos(\pi - O_2O_1O_3) = -\cos(O_2O_1O_3)$, при возрастании угла его косинус уменьшается, из этого получаем неравенство:

$$\frac{\sqrt{p_1^2 + p_2^2}}{\sqrt{p_1^2 + p_2^2 + p_3^2}} \geq \frac{l_2^2 - l_1^2 - p_1^2 - p_2^2 - p_3^2}{2l_1\sqrt{p_1^2 + p_2^2 + p_3^2}},$$

$$\sqrt{p_1^2 + p_2^2} \geq \frac{l_2^2 - l_1^2 - p_1^2 - p_2^2 - p_3^2}{2l_1},$$

$$2l_1\sqrt{p_1^2 + p_2^2} \geq l_2^2 - l_1^2 - (p_1^2 + p_2^2 + p_3^2).$$

Данное неравенство имеет смысл только при $l_2 > l_1$, т.к. при $l_1 \geq l_2$ оно выполняется при любых p_i.

Его можно легко проверить на примере. Пусть $l_1 = 4, l_2 = 5, p_1 = 0, p_2 = = p_3 = \frac{\sqrt{2}}{2}$. Тогда $p_1^2 + p_2^2 + p_3^2 = 1 = O_1O = l_2 - l_1$, то есть такая точка недостижима, и неравенство не должно выполняться. Подставим:

$$2 * 4 * \left(\frac{\sqrt{2}}{2}\right) \geq (5)^2 - (4)^2 - 1 \Rightarrow 4\sqrt{2} \geq 8.$$

Итак, рабочая область ограничивается следующей системой неравенств:

$$\begin{cases} p_1^2 + p_2^2 + p_3^2 \leq (l_1 + l_2)^2, \\ p_3 \geq 0, \\ \cos(q_2) \geq 0, \\ p_1^2 + p_2^2 + p_3^2 \geq (l_1 - l_2)^2, & l_1 > l_2, \\ p_1^2 + p_2^2 + p_3^2 \geq l_2^2 - l_1^2 - 2l_1\sqrt{p_1^2 + p_2^2}, & l_1 < l_2. \end{cases}$$

Для однозначности можно сказать, что

$$-\pi \leq q_1 \leq \pi, \qquad -\frac{\pi}{2} \leq q_2 \leq \frac{\pi}{2}, \qquad -\frac{\pi}{2} \leq q_3 \leq \frac{\pi}{2}.$$

Координата q_2 может изменяться от $-\frac{\pi}{2}$ до $\frac{\pi}{2}$, а не от 0 до $\frac{\pi}{2}$, иначе вокруг оси Z_0 образуется зона, в которую инструмент не сможет попасть, если робот не сделает угол q_3 меньше, чем $-\frac{\pi}{2}$.

Нахождение времени, необходимого для обхода или «сбора» набора точек

Программа принимает следующие входные данные:

- l_1, l_2 (м) – длины звеньев робота,
- T (с) – максимальное время выполнения задачи,
- q' (рад/с) – вектор максимальных скоростей вращения сочленений,
- p_0 – вектор координат (м) начальной точки в системе $O_0X_0Y_0Z_0$,
- p_i – массив векторов координат (м) целевых точек в системе $O_0X_0Y_0Z_0$,
- $task \in \{"visit", "collect"\}$ – тип выполняемой задачи:
  - $"visit"$ – задача «посещения»,
  - $"collect"$ – задача «сбора».

На пространстве состояний системы Q введём функцию $t(p_1, p_2)$ «минимального времени перемещения от точки до точки»:

$$t(p_1, p_2) = \max\left(\frac{|q_{1_1} - q_{2_1}|}{q'_1}, \frac{|q_{1_2} - q_{2_2}|}{q'_2}, \frac{|q_{1_3} - q_{2_3}|}{q'_3}\right),$$

$$p_1, p_2 \in Q, \qquad p_1 = (q_{1_1}, q_{1_2}, q_{1_3}), p_2 = (q_{2_1}, q_{2_2}, q_{2_3}),$$

q'_1, q'_2, q'_3 – максимальные скорости вращения сочленений. В дальнейшем, речь о «расстоянии» между точками идёт в терминах этой функции.

Обычно при равномерной параметризации скорости вращения более быстрых сочленений занижаются, чтобы вращение всех трёх сочленений заканчивалось одновременно. Однако этот факт не влияет на минимально возможное время перемещения инструмента от точки p_1 в точку p_2.

При таком определении функции расстояния ограничение внутренней границы рабочей области при $l_1 < l_2$ не влияет на её значение.

Выполняемый алгоритм состоит из 3 этапов:

1. Подготовка:
   - Считать в память содержимое файлов с расширением «.json», находящихся в папке «input» рядом с исполняемым файлом. Выдать соответствующее сообщение, если папка не найдена или пуста.
   - Для каждого файла проверить наличие в нём требуемых параметров. При отсутствии любого поля – выдать сообщение об ошибке и завершить обсчёт данного входного файла.
   - Проверить типы данных, находящихся в переменных – например, в качестве T не может выступать строка или отрицательное число. При несоответствии типа данных – выдать сообщение об ошибке и завершить обсчёт данного входного файла.
2. Проверка входных значений:
   - Проверить точки p_0 и p_i на то, что все они находятся внутри рабочей области робота – иначе задача становится невыполнимой. В этом случае – выдать сообщение об ошибке и завершить обсчёт данного входного файла.
   - Решить обратную задачу кинематики для точек p_0 и p_i по полученным выше формулам и проверить, что полученные значения обобщённых координат удовлетворяют матричным уравнениям. Если одно из значений не может быть вычислено из-за математической ошибки, или не удовлетворяет какому-либо матричному уравнению – выдать сообщение об ошибке и завершить обсчёт данного входного файла.

3. Выполнение задачи:

- Для задачи «сбора»:
  - для каждой точки p_i подсчитать расстояние от неё до начальной точки и умножить результат на 2, поскольку время затрачивается на достижение точки и возврат на исходную позицию,
  - отсортировать массив значений времени по возрастанию, отслеживая индексы,
  - выбрать из него максимально возможное число элементов, выбирая с начала массива, сумма которых не превышает *T*,
  - вывести список «собранных» целевых точек и затраченное время.
- Для задачи «посещения» использован «жадный» алгоритм поиска пути:
  - взять начальную точку (на первом шаге это p_0) и подсчитать расстояния от неё до всех точек p_i, которые ещё не были посещены,
  - выбрать точку с наименьшим расстоянием от начальной, записать её в массив посещённых и взять в качестве начальной,
  - повторять два предыдущих пункта, пока либо не кончатся непосещённые точки, либо суммарное время не превысит T,
  - вывести список «посещённых» целевых точек и затраченное время.

Для проверки программы взяты три задачи:

1. $l_1 = 4,\ l_2 = 5,\ T = 5,\ q' = (0.02, 0.02, 0.02),\ task = "collect",$
   $p_0 = (0, 0.7071, 0.7071),\qquad p_i = \{(2, 3, 4), (1, 3, 5)\}$

– пример, взятый выше для проверки ограничения рабочей области при $l_1 < l_2$.

В этой задаче обнаруживается проблема с точкой p_0, которая, как было показано выше, лежит за рабочей областью робота, поэтому выполнение программы останавливается заранее.

2. $l_1 = 5,\ l_2 = 4,\ T = 10,\ q' = (0.2, 0.2, 0.2),\ task = "collect",$
   $p_0 = (1, 1, 1),\qquad p_i = \{(2, 3, 2), (1, 3, 3)\}$

В этой задаче проблем не обнаружено, робот выполняет задачу «сбора».

За отведённые 10 секунд робот успевает забрать только один целевой объект из точки $(2, 3, 2)$ и затрачивает на это около 6 секунд.

3. $l_1 = 5,\ l_2 = 4,\ T = 10,\ q' = (0.2, 0.2, 0.2),\ task = "visit",$
   $p_0 = (1, 1, 1),\qquad p_i = \{(2, 3, 2), (1, 3, 3)\}$

– входные данные совпадают с предыдущими, за исключением типа задачи – робот выполняет задачу «посещения».

За отведённые 10 секунд робот успевает обойти обе целевые точки и тратит на это примерно 4.4 секунды.

Заключение

В рамках работы были решены аналитически прямая и обратная задачи по положению инструмента трёхзвенного робота-манипулятора – получены выражения для вычисления координат точки положения инструмента робота при заданных значениях обобщённых координат и нахождения обобщённых координат – углов между звеньями робота – приводящих инструмент в заданную точку в пространстве. В процессе использован алгоритм Денавита-Хартенберга, позволяющий найти матрицы однородного преобразования между локальными системами координат звеньев робота.

Полученные результаты использованы для численной оценки количества подзадач, которое робот способен выполнить за заданное время, если подзадачей считать посещение (возможно, с возвратом обратно) указанной точки. Для нахождения этой оценки используется программа, написанная на языке Python 3.6.

Такая оценка полезна при анализе поставленной задачи управления инструментом робота-манипулятора. Например, при её выполнении для робота-упаковщика может выясниться, что одного робота недостаточно, чтобы при данной скорости конвейера успеть забрать с него все объекты, и требуется либо поставить второго робота, либо замедлить конвейер.

Список источников

1. Герман-Галкин С.Г. «Моделирование в мехатронике».
2. Зенкевич С. Л., Ющенко А. С. «Основы управления манипуляционными роботами: Учебник для вузов» – 2-е изд., исправ. и доп. М.: Изд-во МГТУ им. Н. Э. Баумана, 2004. – 480 с.: ил. (Робототехника / Под. ред. С.Л. Зенкевича, А.С. Ющенко).
3. Климчик А.С., Гомолицкий Р.И., Фурман Ф.В., Сёмкин К.И. «Разработка управляющих программ промышленных роботов». – Минск, 2008. – 131 с.
4. Шиманчук Д.В. «Введение в современную робототехнику». – Санкт-Петербург, 2018. – 203 с.
5. Терентьева Е.И. «Анализ современного состояния применения роботов в промышленности». – Nauka-Rastudent.ru, 2015 г. № 10. - С. 20.
6. Интернет-ресурс: https://robo-hunter.com/news/10-vedushih-proizvoditelei-promishlennih-robotov
7. Бабоченко Н.В. «Компьютерное построение зоны действия шарнирно-стержневого робота манипулятора». – Агротехника и энергообеспечение, 2015г. № 1 (5). – С. 50-55.
8. Гендик Д.С., Ткачев Е.С. «Робототехника в космосе. Проектирование промышленного робота-манипулятора». – Актуальные проблемы авиации и космонавтики, 2016 г. Т. 1. № 12. – С. 244-246.
9. Богданова Ю.В., Гуськов А.М. «Численное моделирование задачи позиционирования инструмента хирургического робота-манипулятора при движении по заданной траектории». – Наука и образование: научное издание МГТУ им. Н.Э. Баумана, 2013 г. № 6. С. 181-210.
10. Гурский Н.Н., Скудняков Ю.А., Артющик В.С., Безручко А.Н. «Управление мехатронной системой на базе многозвенных роботов-манипуляторов». – Наука и техника, 2019 г. Т. 18. № 4. – С. 350-354.

Приложения

Приложение 1. Исходный код программы

```
import codecs
import glob
import json
import logging
import math
import pprint
from operator import itemgetter
from pathlib import Path


def setup_logging():
    LOG_LEVEL_LOG = logging.DEBUG
    LOG_LEVEL_CONSOLE = logging.INFO
    logging.basicConfig(level=LOG_LEVEL_LOG,
                        format='%(levelname)s: %(message)s',
                        filename='output.log',
                        filemode='w')
    console = logging.StreamHandler()
    console.setLevel(LOG_LEVEL_CONSOLE)
    formatter = logging.Formatter(
        '%(levelname)s: %(message)s')
    console.setFormatter(formatter)
    logging.getLogger('').addHandler(console)


def load_inputs():
    input_paths = glob.glob('./input/*.json')
    if not input_paths:
        logging.info('no tasks found in ./input/ directory')
        return
    inputs = {}
    for p in input_paths:
        with codecs.open(p, 'rb') as f:
            try:
                task_data = json.load(f)
            except ValueError as e:
                logging.error(f'Error while loading file {p}')
                logging.error(e)
                continue
            inputs[Path(p).name] = task_data
    return inputs


def iter_and_clean(inputs, func):
    for p, task_data in list(inputs.items()):
        if not func(p, task_data):
            logging.info(f'Task {p} skipped')
```

```
            del inputs[p]


def sanitize_input(p, task_data):
    missing = [key for key in
               ('L1', 'L2', 'T', "q'", 'task', 'p0', 'pi')
               if key not in task_data]
    if missing:
        logging.error(
            f'task {p} does not have necessary data: {missing}')
        return False
    wrong_type = {
        'positive int or float': [
            k for k in ('L1', 'L2', 'T')
            if not isinstance(task_data[k], (int, float))
               or task_data[k] < 0],
        'either "visit" or "collect"': [
            k for k in ('task',)
            if task_data[k] not in ('visit', 'collect')],
        'array of 3 int or float': [
            k for k in ('p0', "q'")
            if not isinstance(task_data[k], list)
               or len(task_data[k]) != 3
               or any(not isinstance(v, (int, float))
                      for v in task_data[k])],
        'non-empty array of arrays of 3 int or float': [
            k for k in ('pi',)
            if not isinstance(task_data[k], list)
               or not task_data[k]
               or any(len(v) != 3
                      or any(not isinstance(_v, (int, float))
                             for _v in v) for v in task_data[k])]
    }
    if any(v for v in wrong_type.values()):
        logging.error(
            f'Task {p}: wrong type of input data:\n'
            + '\n'.join(
                f'{k} must contain {t}'
                for t, ks in wrong_type.items() for k in ks))
        return False
    return True


def check_input(t, task):
    success = True
    L1, L2 = (float(task[k]) for k in ('L1', 'L2'))
    p0, pi = (task[k] for k in ('p0', 'pi'))
    for i, p in enumerate([p0] + pi):
        p = [float(_p) for _p in p]
        p_sq = [math.pow(_p, 2) for _p in p]
        p_hypot_sq = p_sq[0] + p_sq[1] + p_sq[2]
        if p_hypot_sq > math.pow((L1 + L2), 2):
```

```
            logging.info(
                f'Task {t}: point {i} ({p}) is too far')
            success = False
        if p[2] < 0:
            logging.info(
                f'Task {t}: point {i} ({p}) is under floor')
            success = False
        if L1 > L2 and p_hypot_sq < math.pow((L1 - L2), 2):
            logging.info(
                f'Task {t}: point {i} ({p}) too close to origin')
            success = False
        if L1 < L2 and p_hypot_sq < (
                math.pow(L2, 2) - math.pow(L1, 2)
                - 2 * L1 * math.sqrt(p_sq[0] + p_sq[1])):
            logging.info(
                f'Task {t}: point {i} ({p}) too close to origin')
            success = False
    return success


def calculate(path, task):
    logging.info(f'Task {path} passed all checks, starting')
    success = True
    L1, L2, T = (
        float(task[k]) for k in ('L1', 'L2', 'T'))
    p0, pi, q_p, task_type = (
        task[k] for k in ('p0', 'pi', "q'", 'task'))
    q_p1, q_p2, q_p3 = (float(_q_p) for _q_p in q_p)
    qi = []
    for i, p in enumerate([p0] + pi):
        p = [float(_p) for _p in p]
        p_sq = [math.pow(_p, 2) for _p in p]
        p_hypot_sq = p_sq[0] + p_sq[1] + p_sq[2]
        acos_arg = (math.pow(L1, 2) + p_hypot_sq - math.pow(L2, 2)
                    ) / (2 * L1 * math.sqrt(p_hypot_sq))
        asin_arg = (math.pow(L1, 2) - p_hypot_sq + math.pow(L2, 2)
                    ) / (2 * L1 * L2)
        try:
            q1 = math.atan2(p[1], p[0])
            q2 = (math.atan2(math.sqrt(p_sq[0] + p_sq[1]), p[2])
                  - math.acos(acos_arg))
            q3 = (math.asin(asin_arg))
            qi.append([q1, q2, q3])
        except ValueError:
            logging.info(
                f'Task {path}: reverse calculation for point ' +
                f'{i} ({p}) failed: invalid arg for acos or asin')
            logging.info(f'acos arg: {acos_arg}')
            logging.info(f'asin arg: {asin_arg}')
            success = False
    if not success:
        logging.info(
```

```
            f'Task {path}: rotation angles calculation failed')
        return success
    epsilon = 10 ** -6
    pi_2 = math.pi / 2
    q0, qi = qi[0], qi[1:]
    logging.debug(f'{q0}, {qi}')
    for i, (q, p) in enumerate(zip([q0] + qi, [p0] + pi)):
        p = [float(_p) for _p in p]
        p_sq = [math.pow(_p, 2) for _p in p]
        q1, q2, q3 = q
        p_12_sqrt = math.sqrt(p_sq[0] + p_sq[1])
        checks = [
            -math.pi <= q1 <= math.pi,
            -pi_2 <= q2 <= pi_2,
            -pi_2 <= q3 <= pi_2,
            L1 * math.sin(q2) + L2 * math.cos(
                q2 + q3) - p_12_sqrt <= epsilon,
            L1 * math.cos(q2) - L2 * math.sin(q2 + q3) - p[
                2] <= epsilon,
            L1 - p_12_sqrt * math.sin(q2) - p[2] * math.cos(
                q2) - L2 * math.sin(q3) <= epsilon,
            p_12_sqrt * math.cos(q2) - p[2] * math.sin(
                q2) - L2 * math.cos(q3) <= epsilon
        ]
        if not all(checks):
            logging.warning(
                f'Task {path}: point {i} {tuple(p)}: ' +
                f'some matrix check failed: {checks}')
            success = False
    if not success:
        logging.info(f'Task {path}: matrix check(s) failed')
        return success
    if task_type == 'collect':
        q01, q02, q03 = q0
        times = []
        for q in qi:
            qi1, qi2, qi3 = q
            t1, t2, t3 = abs(qi1 - q01) / q_p1, abs(
                qi2 - q02) / q_p2, abs(qi3 - q03) / q_p3
            logging.debug(f'{t1, t2, t3}')
            t = max(t1, t2, t3)
            times.append(t * 2)  # there and back
        indices, times_sorted = zip(*sorted(
            enumerate(times), key=itemgetter(1)))
        logging.debug(f'{indices}, {times_sorted}')
        total_time = 0
        idx = None
        for i in range(len(indices)):
            total_time += times_sorted[i]
            if total_time <= T:
                idx = i
            else:
```

```
            total_time -= times_sorted[i]
            break
    if idx is None:
        logging.info(
            f'Task {path}: could not collect any points, ' +
            f'not enough time')
    else:  # never throws IndexErrors, even if got all points
        collected = indices[:idx + 1]
        logging.info(
            f'Task {path}: collected points:\n\t'
            + '\n\t'.join(
                f'{i} {tuple(pi[i])}' for i in collected)
            + f'\nTime elapsed: {total_time}')
elif task_type == 'visit':
    visited = [-1]
    total_time = 0
    while total_time <= T:
        cur_idx = visited[-1]
        if cur_idx == -1:
            q1, q2, q3 = q0
        else:
            q1, q2, q3 = qi[cur_idx]
        min_time = -1
        min_idx = -1
        for i, q in enumerate(qi):
            if i in visited:
                continue
            qi1, qi2, qi3 = q
            t1, t2, t3 = abs(qi1 - q1) / q_p1, abs(
                qi2 - q2) / q_p2, abs(qi3 - q3) / q_p3
            logging.debug(f'{t1, t2, t3}')
            t = max(t1, t2, t3)  # the first encountered point
            if min_time == -1 or min_time > t:  # is worthy
                min_idx = i
                min_time = t
        if min_idx == -1:  # we visited all points
            break
        total_time += min_time
        if total_time <= T:
            visited.append(min_idx)
        else:
            total_time -= min_time
    visited = visited[1:]
    if not visited:
        logging.info(
            f'Task {path}: could not visit any points, ' +
            f'not enough time')
    else:
        logging.info(
            f'Task {path}: visited points:\n\t'
            + '\n\t'.join(
                f'{i} {tuple(pi[i])}' for i in visited)
```

```
                + f'\nTime elapsed: {total_time}')
    else:
        logging.info(
            f'Task {path}: unrecognized task type: ' +
            f'{repr(task_type)}')
        return False
    logging.info(f'Task {path} finished successfully')
    return success


def main():
    setup_logging()
    inputs = load_inputs()
    logging.debug(pprint.pformat(inputs))
    iter_and_clean(inputs, sanitize_input)
    logging.debug(pprint.pformat(inputs))
    iter_and_clean(inputs, check_input)
    logging.debug(pprint.pformat(inputs))
    iter_and_clean(inputs, calculate)


if __name__ == '__main__':
    main()
    input('Press Enter to continue...')
```