Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ”

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Разработка системы управления для робота, обслуживающего посетителей заведений общественного питания

Автор ____Камнев Юрий Дмитриевич_____ _____
(Фамилия, Имя, Отчество) (Подпись)

Направление подготовки 15.03.06 Мехатроника и робототехника

Квалификация бакалавр

Руководитель ___Куприянов Д.В_____ _____
(Фамилия, И., О., ученое звание, степень) (Подпись)

К защите допустить

Зав. кафедрой Мехатроники Колюбин С.А., к.т.н. _____
(Подпись)

“_____”_____ 2018 г.

Санкт-Петербург, 2018 г.

Студент ___Камнев Ю.Д___ Группа Р3425 Кафедра Мехатроники Факультет СУиР
(Фамилия, И,О.)

Направленность, специализация 15.03.06. Мехатроника

Консультант (ы):
а) _____ _____
(Фамилия, И., О., ученое звание, степень) (Подпись)

б) _____ _____
(Фамилия, И., О., ученое звание, степень) (Подпись)

ВКР принята “____”_____2018 г.

Оригинальность ВКР _____%

ВКР выполнена с оценкой _____

Дата защиты “____”_____2018 г.

Секретарь ГЭК Монахова Наталья Владимировна _____
(подпись)

Листов хранения _____

Демонстрационных материалов/Чертежей хранения _____

Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ”

УТВЕРЖДАЮ

Зав. кафедрой Мехатроники

Колюбин С.А. _____
(подпись)

«____» «_____» 20__ г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Студенту Камневу Ю.Д. **Группа** P3425 **Кафедра** Мехатроники **Факультет** СУиР

Руководитель Куприянов Д.В., ассистент кафедры мехатроники

(ФИО, ученое звание, степень, место работы, должность)

1 Наименование темы: Разработка роботизированной системы для обслуживания посетителей заведений общественного питания

Направление подготовки 15.03.06 Мехатроника и робототехника

Направленность 15.03.06 Мехатроника

Квалификация бакалавр

2 Срок сдачи студентом законченной работы «21» « мая » 2018 г.

3 Техническое задание и исходные данные к работе

Разработать алгоритм управления для манипулятора, способного формировать и выдавать заказы в заведениях общественного питания с помощью программного фреймворка ROS.

Технические требования, предъявляемые к системе:

Автоматизированное построение стандартизированных карт помещений, в которых может оперировать робот, в среде моделирования Gazebo;

Безопасный перенос напитков и других продуктов питания на подносе;

4 Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов)

Описание решения задачи прямой кинематики;
Описание существующих методов решения задачи обратной кинематики;
Описание существующих алгоритмов планирования путей;

Описание существующих алгоритмов планирования траекторий;
Реализация отдельных алгоритмов.
Описание того, как эти алгоритмы реализованы в ROS, выбор оптимальных.
Реализация алгоритма генерирования стандартизированных карт помещений.
Моделирование системы в Gazebo с использованием ROS на роботе UR10.

5 Перечень графического материала (с указанием обязательного материала)

Графики полученных траекторий

Схема манипулятора

Изображения моделей помещения в котором оперирует робот

6 Исходные материалы и пособия

1.Томас Кормен, Ч. Э. Л. Алгоритмы: построение и анализ / Ч. Э. Л. Томас Кормен. — 2-е изд. — MIT Press, 2007.
2. Coursera online course "Robotics: Computational Motion Planning".
3. James Cuffner, S. L. V. RRT-Connect: An Efficient Approach to Single-Query Path Planning. / S. L. V. James Cuffner. — 2000.
4. Luigi Biagiotti, C. M. Trajectory Planning for Automatic Machines and Robots / C. M. Luigi Biagiotti. — Springer, 2008.
5. Mark W. Spong. Robot modelling and control. / Mark W. Spong. — Wiley, 2005.

7 Дата выдачи задания «05» «февраля» 2018 г.

Руководитель ВКР _____
(подпись)
Задание принял к исполнению _____ «____» «_____» 20____г.
(подпись)

Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ”

АННОТАЦИЯ

ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Студент Камнев Юрий Дмитриевич
(ФИО)

Наименование темы ВКР: Разработка системы управления для робота, обслуживающего посетителей заведений общественного питания

Наименование организации, где выполнена ВКР Университет ИТМО

ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

1 Цель исследования Исследование алгоритмов, использующихся при управлении манипуляторами, и разработка, основанной на них, системы управления

2 Задачи, решаемые в ВКР Исследование алгоритмов решения прямой и обратной кинематики, а также алгоритмов планирования путей и траекторий, их реализация и применение к модели робота UR10.

3 Число источников, использованных при составлении обзора 11

4 Полное число источников, использованных в работе 11

5 В том числе источников по годам

Отечественных			**Иностранных**		
Последние 5 лет	От 5 до 10 лет	Более 10 лет	Последние 5 лет	От 5 до 10 лет	Более 10 лет
				3	1

6 Использование информационных ресурсов Internet да - 6
(Да, нет, число ссылок в списке литературы)

7 Использование современных пакетов компьютерных программ и технологий (Указать, какие именно, и в каком разделе работы)

Пакеты компьютерных программ и технологий	**Параграф работы**
ROS	6
Python	6
VREP	2
Gazebo	6
Latex	

8 Краткая характеристика полученных результатов

Исследованы и реализованы алгоритмы решения задач обратной и прямой кинематики. Исследованы и реализованы алгоритмы нахождения путей и построения траекторий с заданными ограниениями. Реализована автоматической построение стандартизированных карт. Промоделирована работа системы в среде моделирования Gazebo на роботе UR10.

9 Полученные гранты, при выполнении работы _____

(Название гранта)

10 Наличие публикаций и выступлений на конференциях по теме выпускной работы__Нет__

(Да, нет)

а) 1 _____

(Библиографическое описание публикаций)

2_____

3_____

б) 1_____

(Библиографическое описание выступлений на конференциях)

2_____

3_____

Студент _____ _____

(ФИО) (подпись)

Руководитель _____ _____

(ФИО) (подпись)

“_____”_____20___г.

Оглавление

Стр.

Стр.

Введение

Использование роботизированных систем в производстве позволило многократно увеличить скорость производства, а также сэкономить на содержании штата. Аналогичные преимущества можно было бы получить, используя роботов для обслуживания клиентов заведений общественного питания. На текущий момент эта сфера еще полностью, за исключением двух случаев, свободна.

При разработке алгоритмической части можно пойти двумя путями: реализовать все алгоритмы с нуля самому или же использовать готовые фреймфорки, такие как *ROS*. В первом случае вы получаете глубокое понимание того, что происходит, но маловероятно, что алгоритмы будут реализованы эффективнее, чем аналогичные во фреймворках. Второй же подход избавляет вас от необходимости изучать, как устроены алгоритмы, и позволяет вам оперировать более высокоуровневыми операциями(наподобие "переместить рабочий элемент в точку $\{x, y, z\}^T$") с самого начала работы. Оба навыка очень полезны. Имея понимание алгоритмов, вы можете оптимизировать их под вашу конкретную задачу, что делает ваш продукт уникальным. Умея же работать с современными фреймворками дает возможность быстро и безболезненно решать поставленные задачи. Обладая же умениями в обоих сферах, вы можете оптимизировать лишь отдельные алгоритмы фреймворка под себя, тем самым находя золотую середину между скоростью разработки и эффективностью алгоритмов. Поэтому все используемые алгоритмы будут реализованы и проверены в среде *VREP*, а аналогичная им система будет промоделирована в *Gazebo* с использованием *ROS*. В качестве управляемого манипулятора взят робот *UR10* фирмы *Universal Robotics*.

Поскольку система должна иметь возможность встраиваться в любые помещения, предлагается разработать механизм генерирования стандартизированных карт помещений. Они будут включать в себя столы и ячейки стандартизированного размера. Таким образом работоспособность системы не будет нарушена при изменении помещения, а также отпадает надобность в использовании дополнительных сенсоров для построения карты. Архитектура системы управления сильно зависит от поставленной задачи. В случае робота, способного формировать заказы и выдавать их клиенту существует ряд ограничений:

- Окружающее пространство содержит препятствия - мебель. Это означает, что должен быть механизм построения геометрического пути с обходом препятствий. Эта задача решается алгоритмами поиска пути.
- При переносе напитков есть риск перевернуть емкость или выплеснуть часть жидкости из нее. Таким образом требуется ограничить максимальные значения ускорений вдоль траектории переноса. Это решается интерполяцией пути, а также параметризацией траекторий.
- Также при переносе подноса с заказом должны сохраняться постоянными углы крена и тангажа, чтобы не перевернуть содержимое. Таким образом решение задачи обратной кинематики требует выбора одного конкретного решения с фиксированными углами, а задача поиска пути должна учитывать ориентацию на каждой итерации.

Еще остается выбор пространства, в котором будет происходить управления: joint space или же operational space. Первый основывается на решении задачи обратной кинематики, а второй - на свойстве Якобиана $Q = JF$. Первый подход несколько проще в разработке, поэтому решено выбрать его.

Таким образом система управления должна включать в себя следующие элементы:

- Решение прямой кинематики. Требуется для вычисления положения рабочего элемента и проверки на коллизии. См. 2.
- Решение обратной кинематики. Требуется для того, чтобы знать в какие точки в пространстве обощенных координат требуется поместить джойнты. См. 3.
- Планировщик геометрических путей. Требуется для построения пути избегающего препятствия, а также выполнения ограничений на ориентацию рабочего элемента. См. 4.
- Планировщик траекторий. Необходим для выполнения ограничений на плавность передвижения. См. 6.

Глава 1. Обзор аналогов

На сегодняшний день мне известны лишь два случая обслуживающих манипуляторов. Первый в заведении *Cafe X*, расположившемся в Сан-Франциско(см. рис. 1.1).

Рисунок 1.1 — Манипулятор, обслуживающий Cafe X

Второй робот находится в Токио в кафе под названием *Henna*.

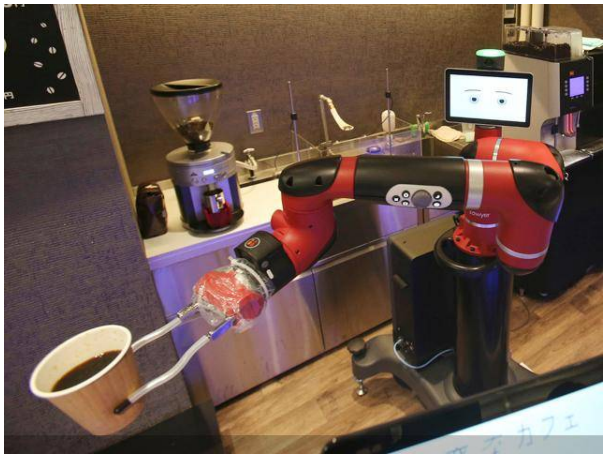

Рисунок 1.2 — Манипулятор, обслуживающий Henna Cafe

Как бы то ни было, но оба эти робота выполняют роль баристы. На самом деле их экономическое преимущество по сравнению с обычными торговыми автоматами неочевидно, в представленных примерах, манипуляторы выполняют, скорее, роль аттракциона, нежели действительно способствуют автоматизации рутинных процессов.

Если бы была возможность встраивать роботизированную систему, способную не только наливать кофе, но и формировать заказы из различных блюд, а

также способную выполнять работу продавцов значительно быстрее, это могло бы сэкономить бизнесу серьезные суммы денег на содержании штата, при этом увеличив пропускную способность заведений.

Глава 2. Задача прямой кинематики

2.1 Введение

При разработке манипуляторов(и не только) возникает задача определения положения рабочего элемента по заданным углам. Данная проблема носит название "задача прямой кинематики". Решается с помощью простейших афинных преобразований, которые представляются в виде однородных матриц преобразования координат. Рассмотрим данный метод более подробно.

2.2 Однородные координаты и матрицы преобразований

Имея N-степеней свободы, мы можем представить операцию масштабирования и поворота с помощью матрицы размером NxN, в то же время мы не можем включить информацию о смещении. На помощь приходят однородные координаты, здесь к обычной матрице поворота добавляется столбец, включающий в себя смещение системы координат.

Например матрица:

$$\begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & cos(\alpha) & -sin(\alpha) & dy \\ 0 & sin(\alpha) & cos(\alpha) & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

не только поворачивает вектор вокруг оси x на α градусов, но и смещает его на dx, dy, dz вдоль осей x, y и z соответственно.

В качестве примера возьмем манипулятор UR10 фирмы Universal Robotics(см. рис. 2.1).

На рисунке изображен манипулятор, система координат и направление осей вращения джойнтов. Длины звеньев нам не даны, но имеются абсолютные

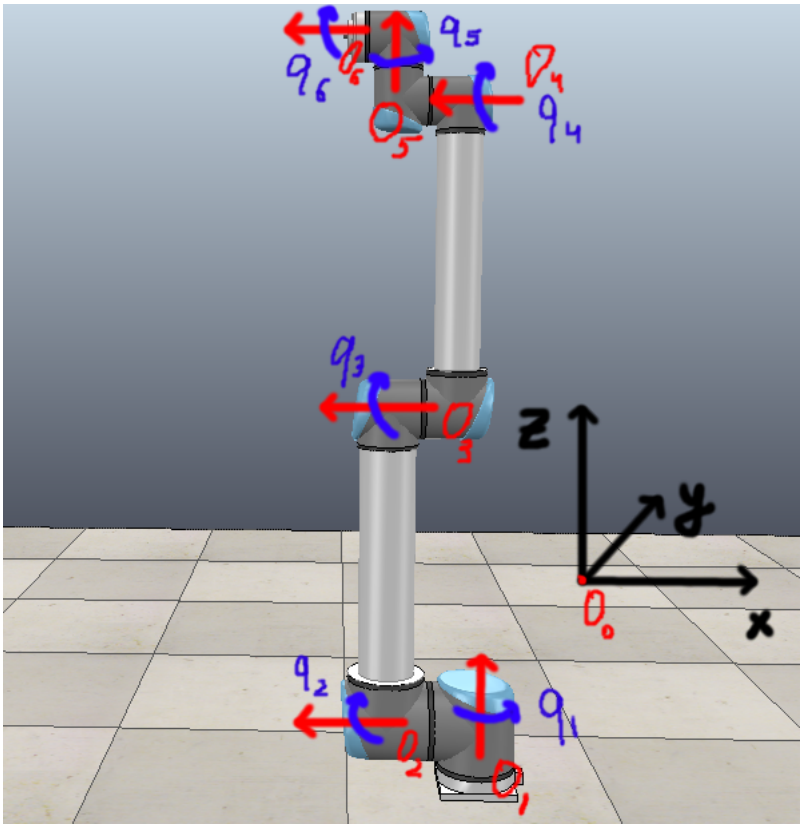

Рисунок 2.1 — Схема UR10. Этот рисунок не претендует на звание кинематической схемы. Кинематическую схему см. на рис. 2.3

координаты джойнтов:

$$
\begin{aligned}
O_0 &= \{0.0000, 0.0000, 0.0000\} \\
O_1 &= \{-0.2500, 0.2500, 0.0447\} \\
O_2 &= \{-0.3427, 0.2500, 0.1280\} \\
O_3 &= \{-0.3621, 0.2506, 0.7401\} \\
O_4 &= \{-0.3555, 0.2500, 1.3123\} \\
O_5 &= \{-0.4139, 0.2500, 1.3696\} \\
O_6 &= \{-0.4712, 0.2500, 1.4282\} \\
O_7 &= \{-0.4986, 0.2500, 1.4280\}
\end{aligned}
$$

Исходя из данных координат, мы можем найти относительные смещения систем координат:

$$O_{i,i+1} = O_{i+1} - O_i$$

Осталось составить сами матрицы переходов между системами координат. Небольшое пояснение к нотации: $\boldsymbol{R_{x/y/z}(q_i)}$ - матрица поворота вокруг

оси x, y или z на угол q_i. $\boldsymbol{\vec{dx}}$ - вектор смещения(первые три элемента четвертого столбца матрицы однородного преобразования). $\vec{\mathbf{0}}$ - вектор вида $\{0, 0, 0\}$. С использованием таких определений матрица преобразования принимает вид:

$$\begin{bmatrix} \boldsymbol{R_{axis}(\varphi)} & \boldsymbol{\vec{dx}} \\ \boldsymbol{\vec{0}^T} & 1 \end{bmatrix}$$

Матрицы преобразований строятся достаточно шаблонно: нам нужно повернуть вектор вокруг оси вращения джойнта, учитывая направление поворота, и переместить систему координат на вектор относительного смещения повернутый на тот же самый угол вокруг той же самой оси.

Запишем матрицы преобразований:

$$T_{01} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{\vec{O}_{01}} \\ \boldsymbol{\vec{0}^T} & 1 \end{bmatrix}$$

$$T_{12} = \begin{bmatrix} \boldsymbol{R_z(q_1)} & \boldsymbol{R_z(q_1) \cdot \vec{O}_{12}} \\ \boldsymbol{\vec{0}^T} & 1 \end{bmatrix}$$

$$T_{23} = \begin{bmatrix} \boldsymbol{R_x(-q_2)} & \boldsymbol{R_x(-q_2) \cdot \vec{O}_{23}} \\ \boldsymbol{\vec{0}^T} & 1 \end{bmatrix}$$

$$T_{34} = \begin{bmatrix} \boldsymbol{R_x(-q_3)} & \boldsymbol{R_x(-q_3) \cdot \vec{O}_{34}} \\ \boldsymbol{\vec{0}^T} & 1 \end{bmatrix}$$

$$T_{45} = \begin{bmatrix} \boldsymbol{R_x(-q_4)} & \boldsymbol{R_x(-q_4) \cdot \vec{O}_{45}} \\ \boldsymbol{\vec{0}^T} & 1 \end{bmatrix}$$

$$T_{56} = \begin{bmatrix} \boldsymbol{R_z(q_5)} & \boldsymbol{R_z(q_5) \cdot \vec{O}_{56}} \\ \boldsymbol{\vec{0}^T} & 1 \end{bmatrix}$$

$$T_{67} = \begin{bmatrix} \boldsymbol{R_x(-q_6)} & \boldsymbol{R_x(-q_6) \cdot \vec{O}_{67}} \\ \boldsymbol{\vec{0}^T} & 1 \end{bmatrix}$$

Полная матрица преобразования:

$$T = \prod_{i=0}^{6} T_{i,i+1}$$

2.3 Denavit–Hartenberg conventions и D-H параметры

В 1955 года Жак Денавит и Ричард Хартенберг разработали соглашение для стандартизации пространственных связей[1]. Соглашения очень простые

- Координатные системы привязаны к джойнтам
- Ось Z координатной системы направлена вдоль оси вращения джойнта.
- Ось X параллельна общей нормали осей вращения предыдущего и текущего джойнта $Z_{i-1} \times Z_i$.
- Ось Y выбирается исходя из X и Z так, чтобы координатная система соответствовала правилу правой руки.

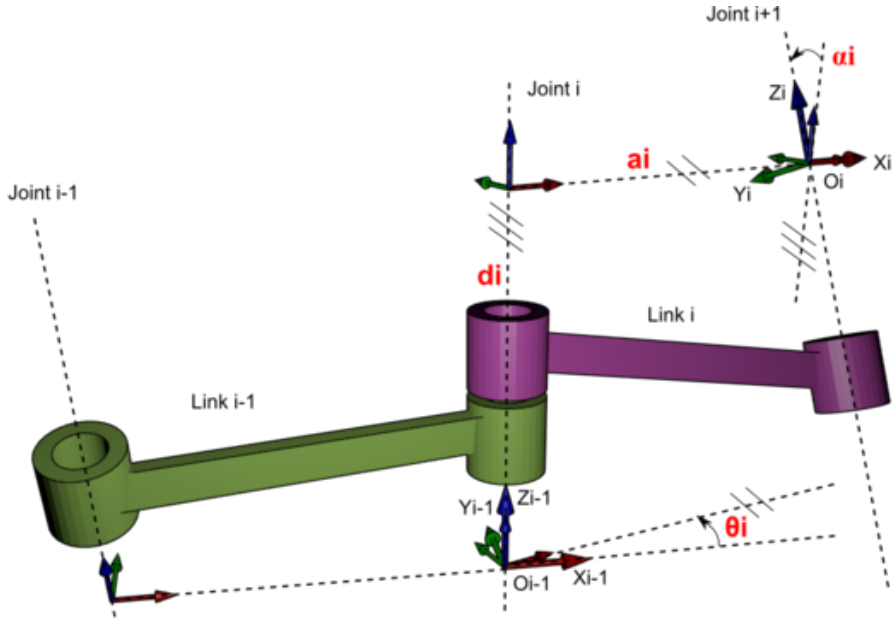

Рисунок 2.2 — Пример D-H параметров

Исходя из вышеуказанных соглашений можно выделить четыре параметра(см. рис. 2.2), необходимых для того, чтобы совершать преобразования:

- d - смещение по оси Z предыдущего джойнта до общей нормали. В случае параллельных осей этот параметр выбирается произвольно.
- θ - угол поворота вокруг предыдущего джойнта, необходимый для совмещения осей X. Это наш текущий угол поворота джойнта плюс начальное смещение.
- r на рисунке обозначен, как a - длина общей нормали.
- α - угол поворота вокруг общей нормали, необходимый для совмещения осей Z

Принимая во внимание указанные выше параметры, можно представить преобразование координат между звеньями как

$$T = Z_1 X_1 T_2 X_2 ... Z_n X_n$$

где

$$\begin{aligned} Z_i &= A_Z(d_i) R_z(\theta_i) \\ X_i &= A_X(r_i) R_x(\alpha_i) \end{aligned} \tag{2.1}$$

Здесь A - однородна матрица смещения координат, а R_a - однородная матрица поворота вокруг оси a.

Выполнение данного соглашения усложняет процесс вычисления преобразований, но дает эффективный метод стандартизации и автоматизации расчета прямой кинематики: реализовав алгоритм преобразований в виде 2.1 единожды, можно решать задачу прямой кинематики для любого робота, для которого выполняется данное соглашение, не изменяя алгоритма.

Кинематическая схема манипулятора UR10 с учетом вышеописанных соглашений см. на рис. 2.3, а соответствующие схеме DH параметры на рис. 2.4.

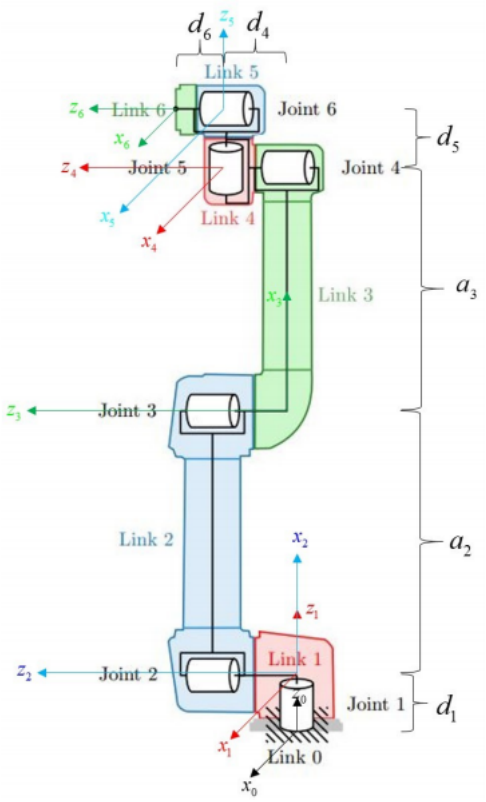

Рисунок 2.3 — Кинематическая схема манипулятора UR10.

Denavit-Hartenberg parameters

Joint	Type	a	α	d	θ	Offset
1	Revolute	0.00000	π/2	0.1273	q1	0.00
2	Revolute	-0.612	0.00	0.00000	q2	-π/2
3	Revolute	-0.5723	0.00	0.00000	q3	0.00
4	Revolute	0.00000	π/2	0.163941	q4	-π/2
5	Revolute	0.00000	-π/2	0.1157	q5	0.00
6	Revolute	0.00000	0.00	0.0922	q6	0.00

Рисунок 2.4 — DH параметры для манипулятора UR10.

Глава 3. Задача обратной кинематики

3.1 Введение

Очень часто при решении задачи управления манипулятором возникает задача определения углов поворотов звеньев, соответствующих заданному положению. Она носит название "задача обратной кинематики". Кроме того данная проблема также возникает в компьютерной графике. Существует большое количество способов ее решения:

- Аналитическое решение
- Использование псевдо-обратного якобиана(pseudoinverse Jacobian)
- Damped least squares
- Использование транспонированного якобиана(Jacobian transpose)

В работе рассмотрены все самые популярные методы решения задачи.

3.2 Аналитическое решение

Аналитическое решение очень удобно тем, что оно позволяет определить все возможные решения, время его вычисления детерминировано и сложность его расчета мала. Но существует ряд проблем, которые не позволяют его использовать для всех случаев жизни и которые породили новые численные методы решения задачи.

Во-первых аналитическое решение существует не всегда. К примеру, если манипулятор избыточен, то решений для каждой точки - бесконечное количество, соответственно так просто решение не определить.

Во-вторых его крайне сложно найти для манипуляторов с большим количеством степеней свободы. Вам понадобится много усидчивости и внимательности, чтобы правильно вывести уравнение для расчета обратной кинематики для механизма с 6 степенями свободы.

На самом деле существуют программные пакеты, такие как IKFast, которые по модели вашего манипулятора могут подобрать вам аналитическое

решение. Но здесь важно понимать, что просто уравнение, а его аппроксимация полиномами высокой степени, которая в свою очередь решается численными методами, но в силу некоторых эвристических соображений, решается она очень быстро, порядка 10 мкс(требует уточнения)[2]. К сожалению даже для таких пакетов существуют случаи, когда решение найти невозможно.

Рассматривать геометрические способы определения аналитического решения я не буду, поскольку дело это неблагодарное, а сразу перейду к численным методам.

3.3 Откуда у якобиана ноги растут

Как мы могли заметить якобиан достаточно часто встречается при решении задачи обратной кинематики. Это объясняется его крайне полезными свойствами, о которых мы поговорим ниже.

Формально якобиан определяется как

$$J = \frac{\partial x}{\partial q}$$

Согласно правилам дифференцирования сложных функций:

$$J = \frac{\partial x}{\partial t}\frac{\partial t}{\partial q} \rightarrow \frac{\partial x}{\partial t} = J\frac{\partial q}{\partial t}$$

Или:

$$\dot{x} = J\dot{q} \tag{3.1}$$

То есть якобиан связывает скорости в operational space со скоростями в joint space. Разрешив уравнение относительно $\dot{q}$, получим:

$$\dot{q} = J^{-1}\dot{x}$$

Пусть x_d и x_e - желаемое и текущее положение рабочего элемента соответственно. Тогда ошибка

$$e = x_d - x_e \tag{3.2}$$

Продифференцировав 3.2 получим,

$$\dot{e} = \dot{x_d} - \dot{x_e}$$

что с учетом 3.1 принимает вид

$$\dot{e} = \dot{x_d} - J(q)\dot{q}$$

Тогда выбрав скорости, как

$$\dot{q} = J^{-1}(q)(\dot{x_d} + Ke) \tag{3.3}$$

получим систему, эквивалентную

$$\dot{e} + Ke = 0 \tag{3.4}$$

Полная схема работы алгоритма представлена на рисунке 3.1

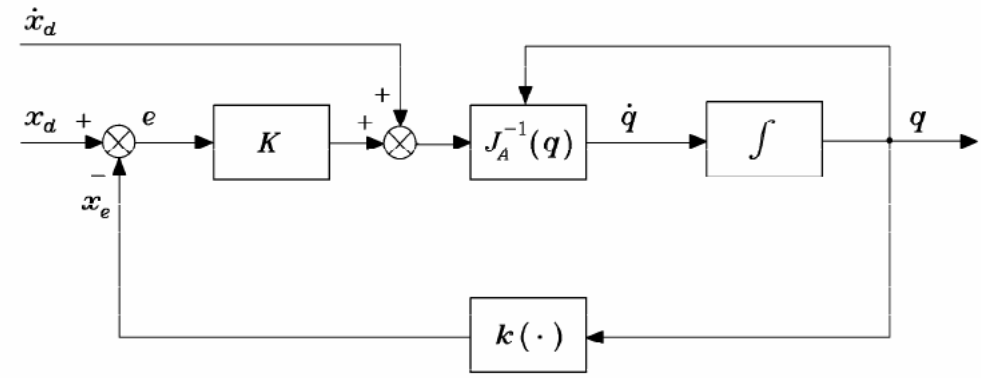

Рисунок 3.1 — Алгоритм обратной кинематики с использованием обратного Якобиана

Система 3.4 стабильна, если K - положительно определенная матрица. Сама же матрица K управляет скоростью стабилизации системы. Предполагая, что $\dot{x_d} = 0$ в 3.3, получим упрощенное выражение для скоростей:

$$\dot{q} = J^{-1}(q)Ke \tag{3.5}$$

Уравнение 3.5 приводит нас к итеративному способу решения проблемы: будем интегрировать необходимое смещение углов с помощью метода Ньютона, то есть выбрав небольшой интервал Δt будем на каждой итерации аппроксимировать смещение на текущей итерации как

$$\Delta q = \dot{q} \cdot \Delta t$$

что с учетом 3.5 принимает вид

$$\Delta q = J^{-1}(q) K e \Delta t \tag{3.6}$$

и далее интегрировать полное смещение

$$q(t_k) = q(t_{k-1}) + \Delta q$$

На каждом этапе работы алгоритма требуется пересчитывать ошибку положений e, что, как следствие, требует расчета прямой кинематики $FK(q(t_{k-1}))$. Алгоритм останавливается, когда e становится меньше заданного порога.

У данного метода есть один недостаток. Уравнение 3.6 имеет решение только в том случае, если J - квадратная матрица и имеет полный ранг, что на самом деле довольно редкая ситуация. Квадратным Якобиан не является, если манипулятор является избыточным, а теряет ранг в случае, когда одно или несколько звеньев двигаются сонаправленно, таким образом теряются степени свободы.

3.4 Псевдо-обратный якобиан

Когда манипулятор имеет избыточные звенья, то есть количество его степеней свободы больше количества переменных, которыми мы хотим управлять(к примеру манипулятор, имеющий 6 степеней свободы, в то время, как мы хотим управлять только тремя - положением без учета ориентации), то Якобиан не является квадратной матрицей, и задачу аппроксимации его обращения можно свести к проблеме линейной оптимизации с ограничениями. [3]

Пусть известны v_e - скорость рабочего элемента и J - Якобиан. Примем следующую квадратичную форму за функцию потерь,

$$g(\dot{q}) = \frac{1}{2} \dot{q} W \dot{q} \tag{3.7}$$

которая с свою очередь должна удовлетворять ограничению 3.1. Здесь W - любая симметричная, положительно определенная матрица размерности $[n\ x\ n]$. К примеру, взяв в качестве нее тензор инерции, мы минимизируем кинетическую энергию движения.

Соответствующий задаче минимизации Лагранжиан имеет вид:

$$L(\dot{q}, \lambda) = \frac{1}{2}\dot{q}W\dot{q} + \lambda(v_e - J\dot{q})$$

Необходимым условием для наличия критической точки является

$$\nabla L = 0$$

откуда

$$\begin{cases} (\frac{\partial g}{\partial \dot{q}})^T = \vec{0} \\ (\frac{\partial g}{\partial \lambda})^T = \vec{0} \end{cases}$$

Продифференцировав, получим

$$\begin{cases} \frac{\partial g}{\partial \dot{q}} = W\dot{q} - J^T\lambda = 0 \\ \frac{\partial g}{\partial \lambda} = v_e - J\dot{q} = 0 \end{cases}$$

Выразим $\dot{q}$ из первого уравнения

$$\dot{q} = W^{-1}J^T\lambda \tag{3.8}$$

Подставив во второе уравнение, получаем

$$v_e = JW^{-1}J^T\lambda$$

Если J имеет полный ранг, тогда матрица $JW^{-1}J^T$ обратима и мы можем вычислить множители Лагранжа:

$$\lambda = (JW^{-1}J^T)^{-1}v_e \tag{3.9}$$

Подставив в 3.8 получим выражение для вычисления $\dot{q}$

$$\dot{q} = W^{-1}J^T(JW^{-1}J^T)^{-1}v_e \tag{3.10}$$

Домножив 3.10 на J легко убедиться, что ограничение 3.1 выполняется. Кроме того достаточно просто проверить, что найденное решение - точка минимума. Вторая производная функции потерь равна

$$\frac{\partial^2 g}{\partial \dot{q}^2} = W$$

где W - положительно определенная матрица, следовательно функция g выпукла вниз и критическая тока является точкой минимума.

Существует особый случай, когда W - единичная матрица. Тогда выражение 3.10 принимает вид:

$$\dot{q} = J^{\dagger} v_e, \text{ где } J^{\dagger} = J^T (J J^T)^{-1} \text{ - правый псевдо-обратный якобиан} \qquad (3.11)$$

Это решение минимизирует норму скоростей звеньев.

Кроме того псевдо-обратный якобиан имеет одно очень полезное свойство, а именно матрица $I_n - J^{\dagger} J$ проецирует вектор на ядро Якобиана. Если мы в качестве минимизируемого функционала примем

$$g\prime(\dot{q}) = \frac{1}{2}(\dot{q} - \dot{q}_0)^T (\dot{q} - \dot{q}_0)$$

то, проведя аналогичные вычисления, мы получим следующее решение

$$\dot{q} = J^{\dagger} v_e + (I_n - J^{\dagger} J)\dot{q}_0 \qquad (3.12)$$

Это означает, что если мы будем выбирать определенным образом вектор q_0, мы можем решать второстепенные задачи. К примеру, задав $v_e = 0$, можно изменять конфигурацию манипулятора, не изменяя положения рабочего элемента, что может быть удобно для избежания препятствий.

Обычным способом выбора является $q_0 = k(\frac{\partial w(q)}{\partial q})^T$, где $k > 0$ - скаляр, а $w(q)$ - второстепенная целевая функция. Поскольку движение выполняется вдоль градиента $w(q)$, то движение манипулятора пытается локально максимизировать функцию, при этом не оказывая влияния на решение основной задачи.

Типичные второстепенные целевые функции [3]:

– Мера управляемости,

$$w(q) = \sqrt{\det J(q) J^T(q)}$$

максимизируя которую, манипулятор избегает вырожденных положений во время движения.

– Расстояние до пределов джойнтов

$$w(q) = -\frac{1}{2n} \sum_{i-1}^{n} (\frac{q_i - \overline{q_i}}{q_{iM} - q_{im}})^2$$

где q_{iM} и q_{im} - максимальные и минимальные значения углов i-го джойнта. Таким образом при движении манипулятор старается держать значения углов как можно ближе к медианам возможных принимаемых значений.

– Расстояние до препятствия

$$w(q) = \min ||\vec{p}(q) - \vec{o}||$$

Здесь $\vec{o}$ - положение препятствия, $\vec{p}(q)$ - точка манипулятора. Таким образом во время движения манипулятор будет пытаться держать расстояние до препятствия как можно большим.

Важно заметить, что метод с использованием псевдо-обратного Якобиана не решает проблему обратимости полностью. Уравнение 3.9 имеет решение только в случае, если Якобиан имеет полный ранг, что неверно в пространстве сингулярных конфигураций. Это может привести к колебаниям вокруг решения имеющего сингулярную конфигурацию, посколько $J^{\dagger}$ принимает очень большие значения в этой области.

3.4.1 Примеры и результаты работы алгоритма

Были проведены эксперименты с различными значениями точности работы(порога остановки) и скоростью схождения алгоритма.

Используемые обозначения:

– ε - порог остановки, рассчитывается как норма вектора ошибки.
– k - скорость схождения. Подразумевается диагональная матрица со значениями k.
– T - время работы алгоритма(определяется после нахождения решения).
– J_m - максимальное значение нормы Якобиана(определяется после нахождения решения).

Параметры одинаковые для всех экспериментов:

– Начальное положение углов $\vec{q} = \vec{0}$.
– Начальное положение рабочего элемента $\vec{s} = \{-0.4986, 0.25, 1.1428\}^T$.
– Требуемое положение рабочего элемента $\vec{t} = \{-1.0992, 0.7642, 0.6451\}^T$.

Начальное положение имеет высокую степень сингулярности(см. рис. 2.1).

Эксперимент 1:

$\varepsilon = 0.001$м. $= 0.1$см.

$k = 0.1$

$T = 502$мс.

$J_m = 3931$

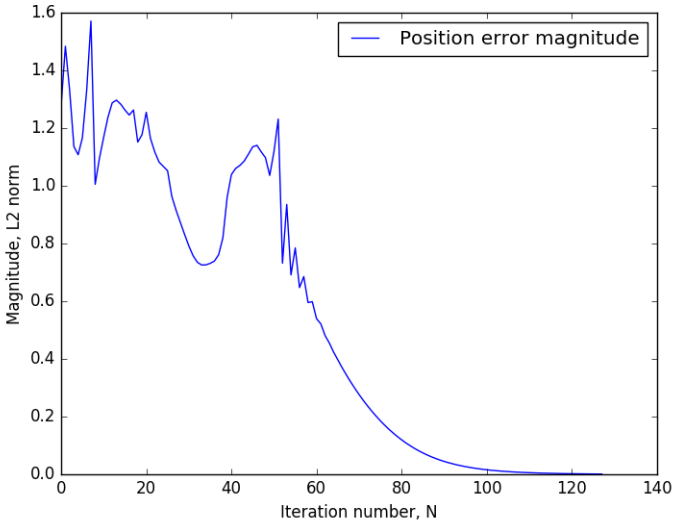

Рисунок 3.2 — Ошибка положения

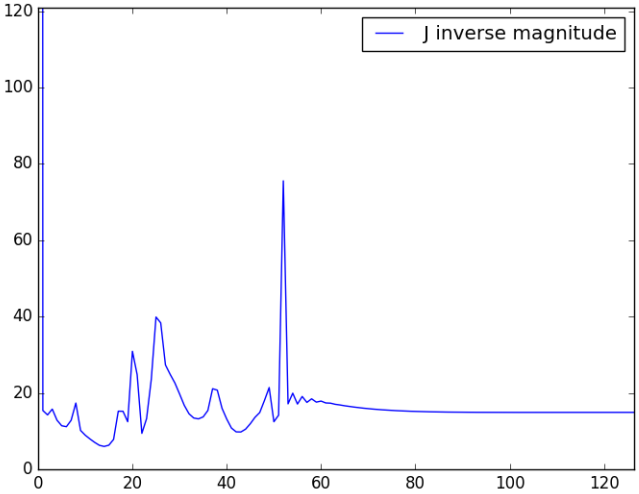

Рисунок 3.3 — Норма обратного якобиана

Как видно из рис. 3.3 в начале работы алгоритма Якобиан принимает очень большие значения, это связано с тем, что начальное положение манипулятора выбрано сингулярным. Это в свою очередь ведет к колебаниям в ошибке, что видно на рисунке 3.2. Это не представляет проблемы в данном случае, поскольку требуемая конфигурация не сингулярна.

Эксперимент 2:

$\varepsilon = 0.00001$м. $= 0.001$см.

$k = 0.1$

$T = 599$мс.

$J_m = 3931$

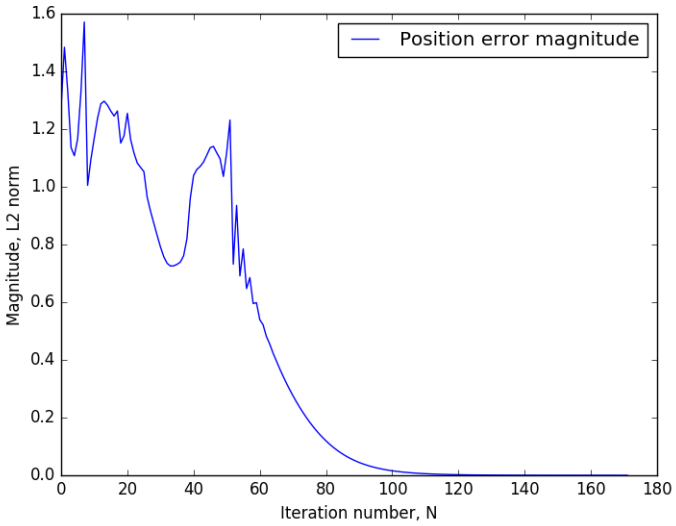

Рисунок 3.4 — Ошибка положения

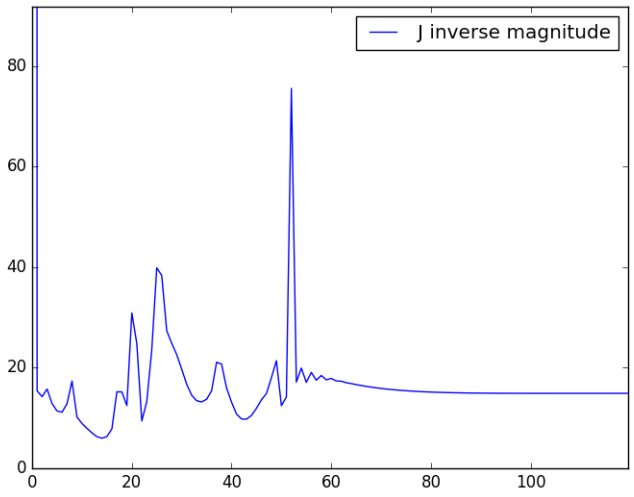

Рисунок 3.5 — Норма обратного якобиана

Увеличена точность работы. Графики не отличаются от полученных в первом эксперименте ничем з исключением длительности. Как видно из результатов увеличив точность в 100 раз, время работы выросло всего на 100 миллисекунд. Точности в 10мкм достаточно для решения большинства задач.

Эксперимент 3:

$\varepsilon = 0.00001$м. $= 0.001$см.

$k = 0.25$

$T = 205$мс.

$J_m = 3931$

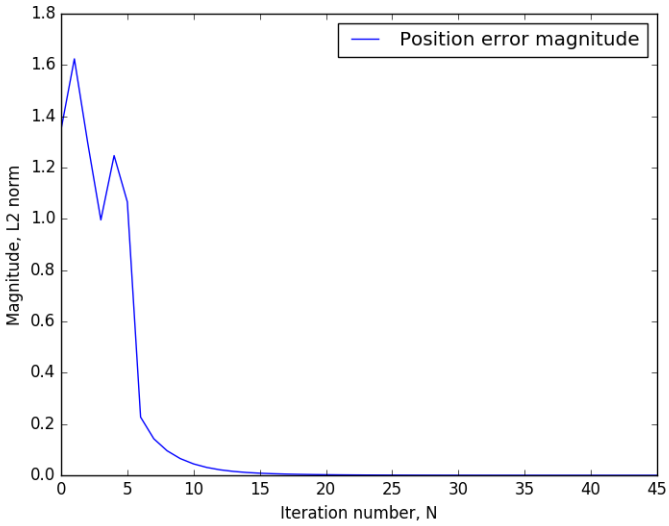

Рисунок 3.6 — Ошибка положения

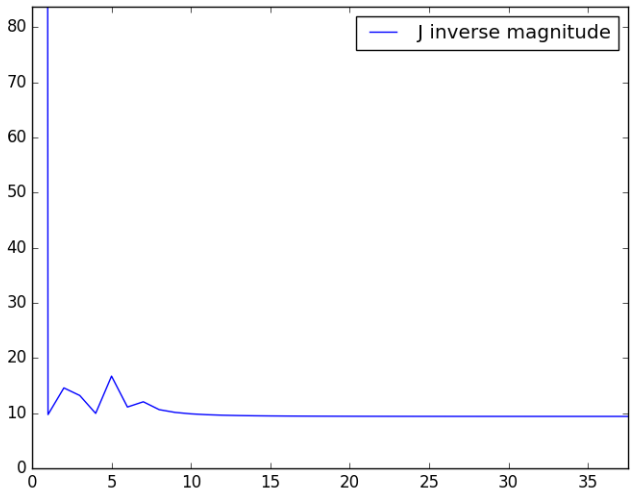

Рисунок 3.7 — Норма обратного якобиана

Был увеличен коэффициент схождения до значения $k = 0.25$, за счет чего получен большой прирост скорости работы: текущее время работы составило 205 мс., что примерно на 300 миллисекунд меньше, чем при $k = 0.1$. Естественно взамен скорости мы получаем некоторые проблемы. Чем больше шаг,

тем больше шанс перешагнуть через решение, таким образом возможно получить незатухающие колебания вокруг точки решения. Особенно актуальна эта проблема в сингулярных конфигурациях, когда и без того большой Якобиан умножается на высокий коэффициент. К примеру взяв $k = 1$ мы никогда не получим решение для этого же случая.

3.5 Damped least squares

Как уже было сказано, уравнение 3.9 имеет решение, только если Якобиан имеет полный ранг, чего не происходит в пространстве сингулярных конфигураций. Существует другой способ обращения Якобиана, названный Damped Least-Squares pseudo-inverse. Решение слабо отличается от найденного в предыдущей главе:

$$J^* = J^T(JJ^T + k^2I) \tag{3.13}$$

$$\dot{q} = (J^*)^{-1}v_e \tag{3.14}$$

где k - сглаживающий(damping) коэффициент, обеспечивающий численную стабильность вычислений при сингулярных конфигурациях. Это решение можно найти минимизировав функционал потерь[3]:

$$g''(\dot{q}) = \frac{1}{2}(v_e - J\dot{q})^T(v_e - J\dot{q}) + \frac{1}{2}k^2\dot{q}^T\dot{q} \tag{3.15}$$

3.5.1 Примеры и результаты работы алгоритма

Здесь также были проведены эксперименты с различными параметрами алгоритма.

Были проведены эксперименты с различными значениями точности работы(порога остановки) и скоростью схождения алгоритма.

Используемые обозначения:

- ε - порог остановки, рассчитывается как норма вектора ошибки.
- α - скорость схождения. Подразумевается диагональная матрица со значениями α.

– k - сглаживающий коэффициент.
– T - время работы алгоритма(определяется после нахождения решения).
– J_m - максимальное значение нормы Якобиана(определяется после нахождения решения).

Параметры одинаковые для всех экспериментов:
– Начальное положение углов $\vec{q} = \vec{0}$.
– Начальное положение рабочего элемента $\vec{s} = \{-0.4986, 0.25, 1.1428\}^T$.
– Требуемое положение рабочего элемента $\vec{t} = \{-1.0992, 0.7642, 0.6451\}^T$.

Начальное положение имеет высокую степень сингулярности(см. рис. 2.1).

Эксперимент 1:

$\varepsilon = 0.001$м. $= 0.1$см.
$\alpha = 0.1$ $k = 0.3$
$T = 861$мс.
$J_m = 15.78$

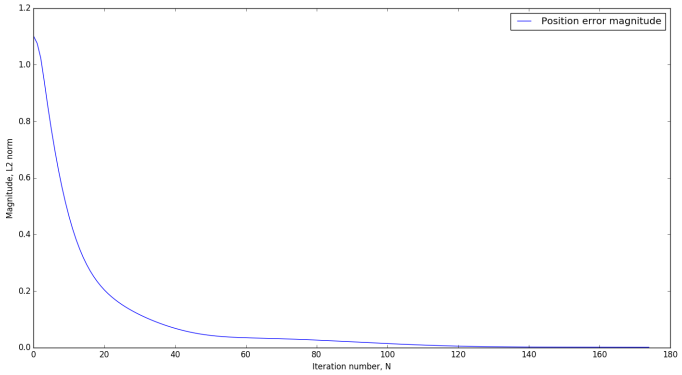

Рисунок 3.8 — Ошибка положения

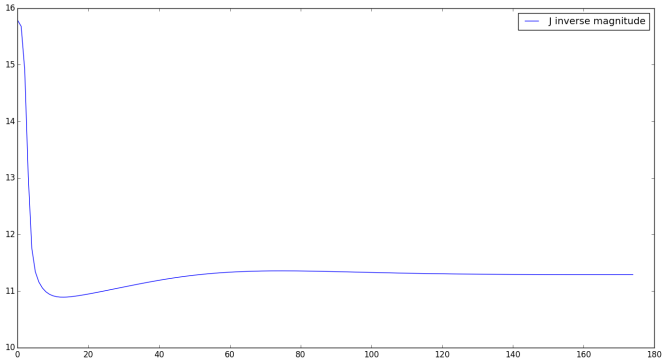

Рисунок 3.9 — Норма обратного якобиана

Как видно из рис. 3.9 максимальное значение нормы Якобиана в сингулярной области сократилось больше чем в 200 раз по сравнению с аналогичным экспериментом, используя псевдо-обратный Якобиан. Кроме того графики нормы Якобиана и магнитуды ошибки положений на рис. 3.8 стали более гладкими, но ценой скорости работы - 800 мс. против 500 мс. с использованием псевдо-обратного Якобиана.

Эксперимент 2:

$\varepsilon = 0.00001$м. $= 0.001$см.

$\alpha = 0.1$ $k = 0.1$

$T = 507$мс.

$J_m = 141$

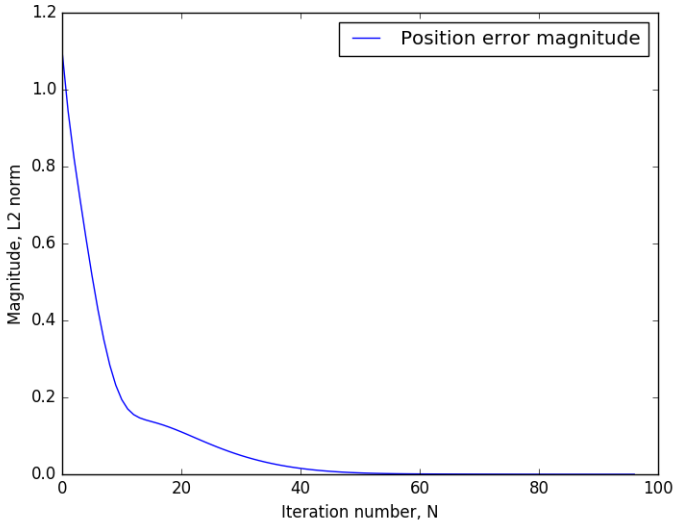

Рисунок 3.10 — Ошибка положения

Сглаживающий коэффициент уменьшен в три раза с 0.3 до 0.1 и повышена точность до 10 мкм. Как видно уменьшение сглаживающего коэффициента приводит к резкому увеличению в скорости работы: хотя и была повышена точность, но время поиска решения сократилось до 507 мс., что уже почти на 100 мс. быстрее аналогичного эксперимента с использованием псевдо-обратного Якобиана.

Эксперимент 3:

$\varepsilon = 0.00001$м. $= 0.001$см.

$\alpha = 0.25$ $k = 0.1$

$T = 172$мс.

$J_m = 141$

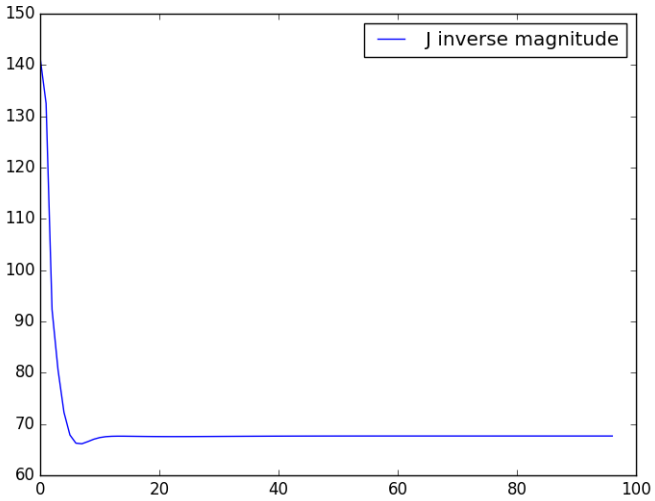

Рисунок 3.11 — Норма обратного якобиана

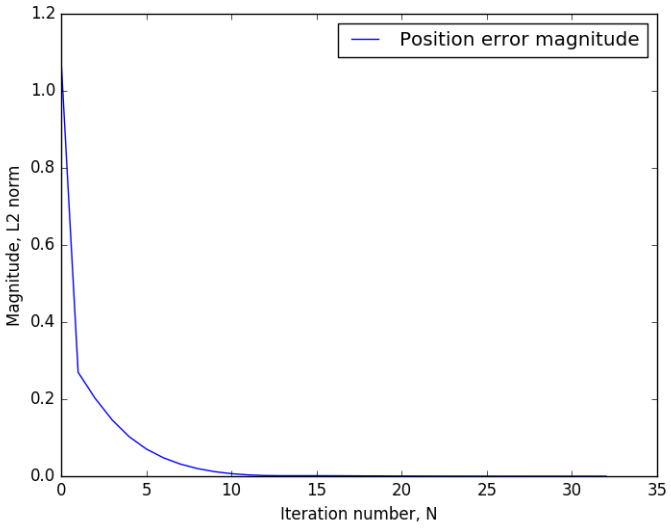

Рисунок 3.12 — Ошибка положения

Увеличив коэффициент схождения до 0.25, мы получаем огромный выигрыш в скорости работы 172 мс. против 507 мс. в прошлом эксперименте. К сожалению даже со сглаживающим фактором невозможно увеличивать коэффициент до особо больших значений.

Эксперимент 4:

$\varepsilon = 0.00001$м. $= 0.001$см.

$\alpha = 1$ $k = 0.1$

$T = 172$мс.

$J_m = 141$

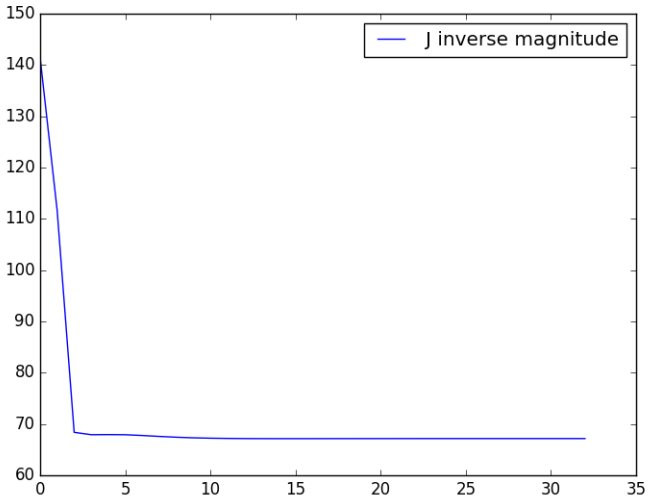

Рисунок 3.13 — Норма обратного якобиана

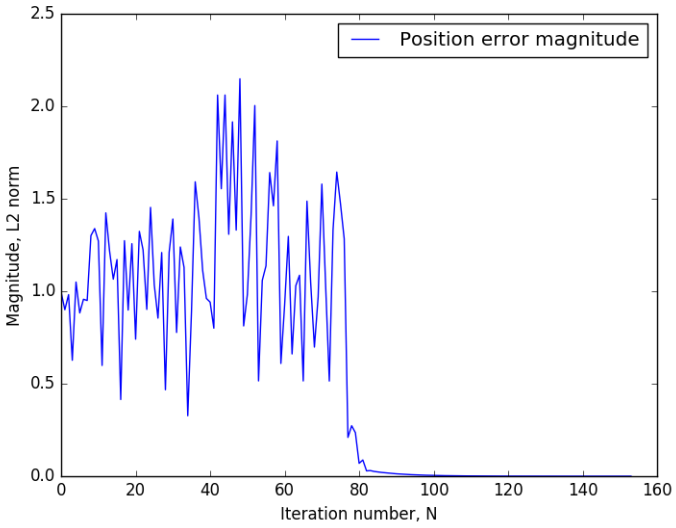

Рисунок 3.14 — Ошибка положения

То, что можно получить в сингулярной области, имея высокий коэффициент схождения показано на рисунках 3.14 и 3.15. К счастью алгоритм все еще сходится, потому что решение находится вне пространства сингулярных конфигураций. В противном случае он мог бы никогда не найти решение, колеблясь в некоторой области вокруг него.

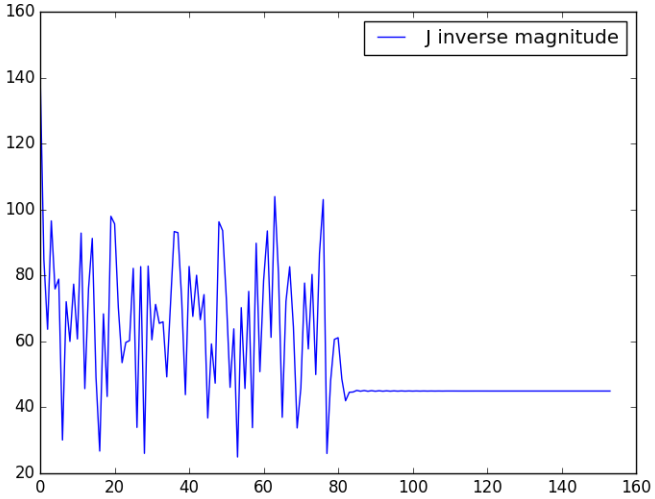

Рисунок 3.15 — Норма обратного якобиана

3.6 Транспонирование Якобиана

Более простой с точки зрения вычислений алгоритм может быть получен, используя метод функции Ляпунова.[3] Возьмем в качестве кандидата на роль функции Ляпунова следующую квадратичную форму

$$V(e) = \frac{1}{2} e^T K e \tag{3.16}$$

где $e = x_d - x_e$, а K - положительно определенная матрица. Функция $V(c) > 0 \forall c \neq 0$ и $V(0) = 0$. Дифференцируя 3.16 и принимая во внимание, что $\dot{e} = \dot{x}_d - \dot{x}_e$, получим

$$\dot{V} = e^T K \dot{x}_d - e^T K \dot{x}_e = e^T K \dot{x}_d - e^T K J \dot{q} \tag{3.17}$$

Задавая скорости как

$$\dot{q} = J^T K e$$

и подразумевая $\dot{x}_d = 0$, получим, что 3.17 вырождается в

$$\dot{V} = e^T K J J^T K e \tag{3.18}$$

Функция 3.18 отрицательно определена, если Якобиан имеет полный ранг, в этом случае условия $\dot{V} < 0$ и $V > 0$ выполнены, а следовательно V является функцией Ляпунова и система асимптотически стабильна. Алгоритм также имеет проблемы в случае, если Якобиан имеет не полный ранг.

3.7 Геометрическое вычисление Якобиана

Аналитический расчет Якобиана требует дифференцирования матрицы преобразований, что для манипуляторов с большим количеством степеней свободы может оказаться слишком трудоемкой задачей. Вместо этого можно использовать простейшие геометрические соотношения и уравнения кинематики.

Представим Якобиан в виде

$$J = \begin{bmatrix} J_{P1} & \dots & J_{Pn} \\ & \dots & \\ J_{O1} & \dots & J_{On} \end{bmatrix}$$

где

$$J_{Pi} = \frac{\partial P}{q_i} \in \Re^3$$
$$J_{Oi} = \frac{\partial O}{q_i} \in \Re^3$$

$P(q)$ и $O(q)$ - функции положения и ориентации рабочего элемента.

Тогда, исходя из выражения для связи скоростей в предыдущем фрейме и в текущем

$$\omega_i = \omega_{i-1} + \dot{\theta} z_{i-1}$$
$$\dot{p}_i = \dot{p}_{i-1} + \omega_i \times r_{i-1,i}$$

можно получить[3]

$$\begin{bmatrix} J_{Pi} \\ J_{Oi} \end{bmatrix} = \begin{cases} \begin{bmatrix} z_{i-1} \\ \vec{0} \end{bmatrix} & \text{для призматического джойнта} \\ \begin{bmatrix} z_{i-1} \times (p_e - p_{i-1}) \\ z_{i-1} \end{bmatrix} & \text{для вращающего джойнта} \end{cases}$$

Здесь

$$\begin{cases} z_{i-1} = R_1^0(q_1) \dots R_{i-1}^{i-2}(q_{i-1}) z_0 \\ z_0 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \end{cases} \tag{3.19}$$

$$\begin{cases} p_{i-1} = A_1^0(q_1)...A_{i-1}^{i-2}(q_{i-1})p_0 \\ p_0 = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T \end{cases}$$

Стоит заметить, что $z_0 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$ в 3.19 всегда выбирает третий столбец матрицы поворота, потому что сделано предположение о выполнении Denavit–Hartenberg conventions 2.3. В противном случае он выбирает ось вращения согласно модели вашего робота.

Глава 4. Планирование путей

4.1 Введение

Прежде чем робот сможет выполнить движение нужно знать, как и куда робот должен двигаться. Данную задачу разделяют на две составляющие: планирование пути и планирование траектории. Первая решает кинематическую задачу, определяя несколько дискретных точек, через которые должен пройти робот, чтобы избежать столкновения с препятствиями. Вторая - задачу динамики, на этом этапе составляются циклограммы движения с заданными начальными условиями и ограничениями на скорости и ускорения. Кроме того обе задачи можно также разделить на два подтипа: генерирование пути или траектории заранее или в режиме реального времени. Первый случай отлично подходит, если среда, в которой работает робот, статична, т.е. препятствия не меняют своего расположения. Второй подход решает проблему работы в изменяющейся во времени среде, но не обеспечивает какой либо оптимальности в планировании траекторий.

В ROS интегрирован фреймворк поиска путей и планирования траекторий **MoveIt**, который в свою очередь использует open source библиотеку **OMPL**, в которой реализованы все самые популярные sample based планировщики траекторий. Рассмотрим реализованные алгоритмы более подробно.

4.2 Алгоритмы планирования пути

4.2.1 Grid based search

Одним из видов представления карты окружающей среды является разбиение карты на дискретную сетку. Для данного способа представления карты можно использовать обычный двумерный массив или же граф. Каждой ячейке или ребру ставится в соответствие вес, который характеризует

"цену"перемещения согласно некоторой функции потерь. При этом ячейкам с препятствиями присваивается очень большой(маленький) вес, чтобы исключить столкновения в пути.

Имея построенную карту можно планировать путь, используя классические алгоритмы поиска кратчайшего пути на графе, такие как алгоритм Дейкстры, A^*, D^* и т.п.

4.2.2 PRM

Probabilistic roadmaps(PRM) - эффективный вероятностный алгоритм поиска путей в известной местности. Принцип работы достаточно прост: он заполняет карту N случайными сэмплами из пространства состояний. Каждый раз, сгенерировав новое случайное состояние, алгоритм пытается соединить новую вершину с k ближайшими соседями в графе или со всеми вершинами в некой окрестности δ, проверяя при этом выполнение ограничений, наложенных динамикой системы, а также на столкновения. После того, как карта сгенерирована, можно планировать траектории. По уже описанному алгоритму к графу добавляются две новые вершины: исходная и целевая. После чего применяются стандартные алгоритмы поиска кратчайшего пути на графе, такие как алгоритм Дейкстры[4], A^*[5] и т.п.

Из преимуществ алгоритма можно указать возможность повторного использования карт для разных начальных и конечных состояний, а также скорость работы после построения карты.

К недостаткам относятся экспоненциальный рост размера карты в зависимости от размерности пространства состояний, а также отсутствие оптимальности найденного пути во всех смыслах. Кроме того вероятность успешного поиска пути сильно зависит количества сэмплов N и их распределения. Посмотрите на следующий рисунок:

Проход между препятствиями слишком узок для того, чтобы в него попало достаточное количество сэмплов для построения пути. Эту проблему можно решить, выбирая сэмплы так, чтобы в окрестностях препятствий плотность их распределения была выше. Но это решение одной частной проблемы. На данный момент не существует универсального метода распределения случайных

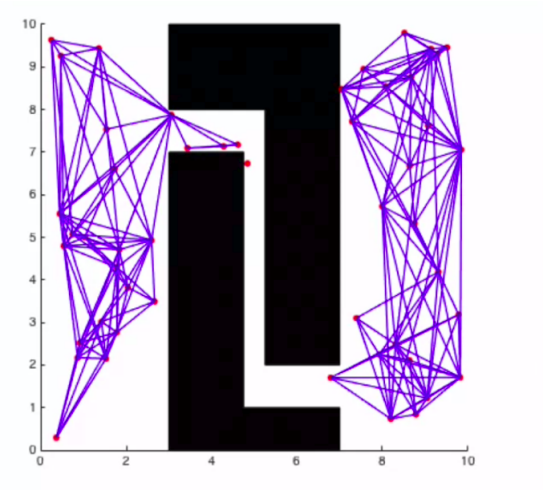

Рисунок 4.1 — Probabilistic road map не сможет найти путь

состояний на карте так, чтобы путь был найден при произвольной конфигурации окружающей среды.[6]

4.2.3 RRTConnect

Одним из самых эффективных алгоритмов поиска пути является **RRTConnect**, который основан на использовании двух **RRT**(Rapidly-exploring random trees)[7]. Является probabilistically complete, в том смысле, что вероятность верно найденного решения стремится к единице, если время поиска $t \to \infty$.

Что же такое RRT? Это обычное дерево, к которому на каждой итерации построения добавляется случайная вершина. Чтобы построить данное дерево нужно определить два параметра: δ, который определяет расстояние между вершинами дерева в пространстве поиска, и N - максимальное количество итераций. Допустим, на текущем шаге построения мы имеем дерево изображенное на рис. 4.2.

Сначала случайным образом выбирается новая точка X в пространстве поиска(рис. 4.3). После в дереве определяется ближайшая к ней вершина Y и, если она находится на расстоянии меньшем чем δ от X, удовлетворяет всем ограничениям наложенными динамикой системы, а также на пути нет препятствий, то она добавляется к дереву. Если же вершина удалена больше, чем на

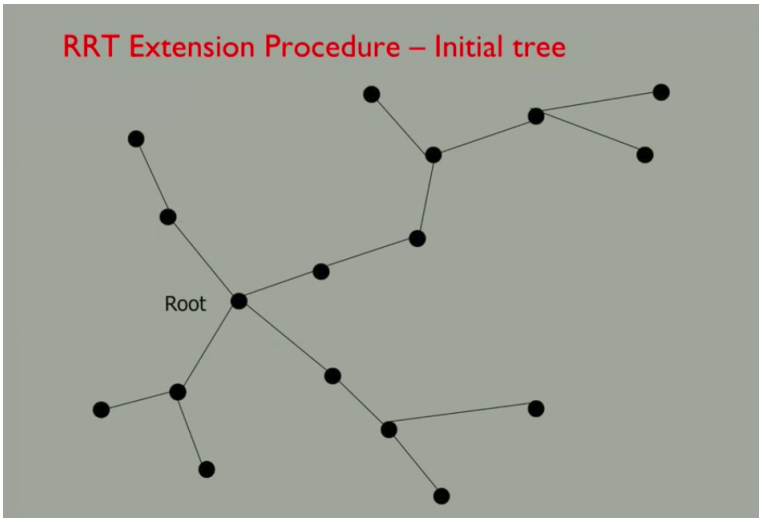

Рисунок 4.2 — Исходное дерево

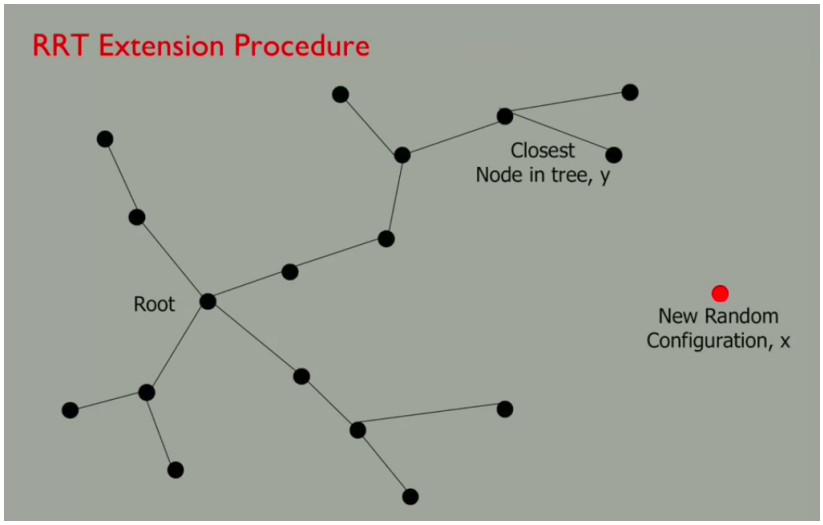

Рисунок 4.3 — Новая случайная точка

δ, то ее заменяют вершиной на расстоянии δ от вершины Y на прямой, соединяющей ее с вершиной X(см. рис. 4.4). В случае, если путь не удовлетворяет ограничениям динамики, переходим к следующей итерации.

RRTConnect строит два RRT дерева: одно - из текущей вершины, второе - из требуемого положения - и на каждой итерации пытается соединить оба дерева. В алгоритме построения есть одно небольшое изменение - операция extend для вершины X выполняется до тех пор, пока на пути не возникнет препятствия или эта вершина не окажется добавленной к дереву. Путь считается найденным, если оба дерева оказались соединены. Если же количество итераций превысило N, поиск считается проваленным.[8]

Узкое место в работе алгоритма - поиск ближайшего соседа, что при хранении дерева в виде обычного связного списка имеет сложность $O(n)$.

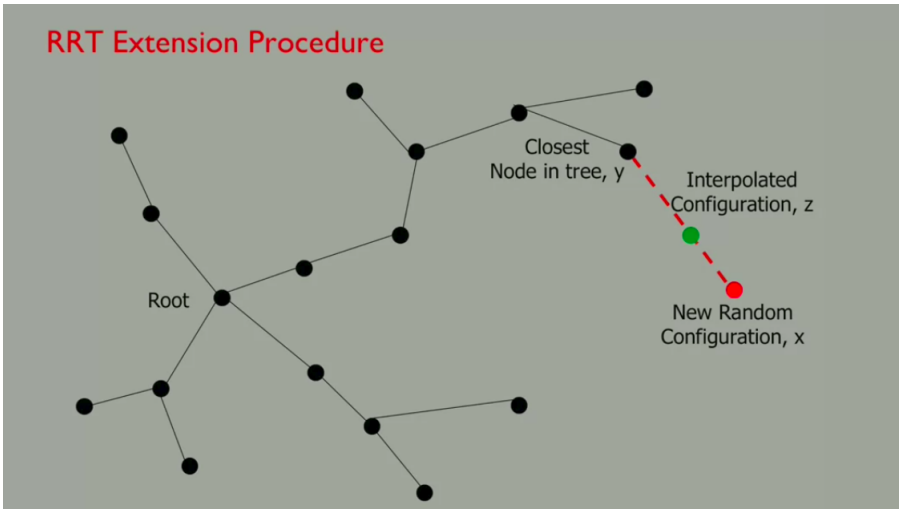

Рисунок 4.4 — Добавляем новую вершину к дереву

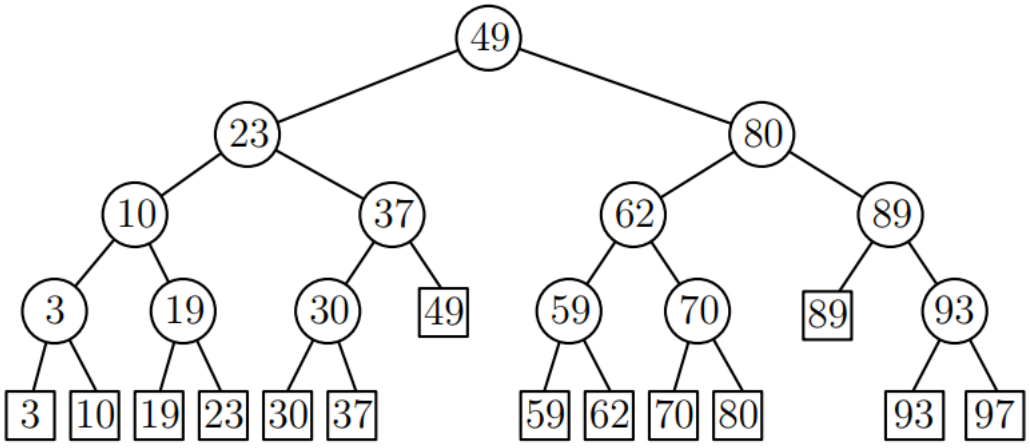

Рисунок 4.5 — Пример построенного одномерного K-D дерева

Дабы ускорить процесс используются так называемые K-D(k-dimensional) деревья. K-D дерево - бинарное дерево поиска, в котором каждая вершина делит пространство на две части по какой-либо из осей координат, а листья - непосредственно сами точки в пространстве. Для многомерных пространств ось координат по которой происходит разделение меняется с каждым новым продвижением вниз по дереву. На рис. 4.5 изображено одномерное K-D дерево. Каждая вершина разделяет пространство на два отрезка. Изучить, как устроен поиск ближайшего соседа вы можете на странице Википедии, посвященной данному алгоритму[9].

Основным преимуществом данного алгоритма является крайне высокая практическая эффективность.[8] Поэтому на сегодняшний день это один из самых популярных алгоритмов поиска пути для манипуляторов. К недостатками относится отсутствие оптимальности найденного пути.

Глава 5. Планирование траекторий

5.1 Введение

Существует несколько различных ситуаций, в которых требуется планировать траекторию:

– Траектория между несколькими точками без избегания препятствий
– Траектория между двумя точками без избегания препятствий
– Траектория между двумя точками избегающая препятствия

Первый случай - результат работы геометрического планировщика, построившего путь с учетом препятствий. Второй случай возникает в ситуациях, когда геометрический планировщик построил путь свободный от препятствий или же известно, что они отсутствуют. Третий случай является результатом отсутствия геометрического планировщика или быстро изменяющейся динамической среды, в которой находится много перемещающихся препятствий(к примеры многолюдное помещение и тп.п). Этот случай довольно редок на практике, поскольку удешевление микропроцессорной техники и вычислительных ресурсов привело к тому, что появилась возможность в режиме реального времени строить пути свободные от препятствий в условиях динамической среды. Поэтому этот случай рассмотрен не будет. Кроме того не рассмотрен случай генерирования траекторий по нескольким точкам с избеганием препятствий, потому что в этом случае путь разбивается на пары точек и используются те же самые методы, что и в траекториях между двумя точками.

5.2 Планирование траекторий между несколькими точками без обхода препятствий

После построения планировщиком пути, необходимо построить траекторию, то есть определить динамические параметры, такие как скорость и ускорение на всем пути. Во-первых нужно получить непрерывный путь, для решения этой задачи можно использовать сплайны. Степень сплайна выбирается

исходя из требований к плавности траекторий. В данной работе используются кубические сплайны, которые обеспечивают гладкость функций положения и скорости, а также непрерывность функции ускорения. Функция рывка разрывна, но ограничена сверху и снизу.

5.2.1 Интерполяция пути кубическим сплайном

Каждый полином кубического сплайна имеет четыре коэффициента и представляется в виде: $a_0x^3 + a_1x^2 + a_2x + a_3$. Из этого следует, что мы можем удовлетворить четыре условия на начальные и конечные значения переменных. Будем считать, что нам известны положения q_k, q_{k+1}, а также скорости v_k, v_{k+1} в моменты времени t_k и t_{k+1}. В таком случае перед нами стоит задача восстановить функцию [10]:

$$s(t) = \{q_k(t),\, t \in [t_k, t_{k+1}], k = 0, \ldots, n-1\},$$
$$q(k) = a_{k0} + a_{k1}(t - t_k) + a_{k2}(t - t_k)^2 + a_{k3}(t - t_k)^3$$

При это нужно удовлетворить наложенные ограничения:

$$\begin{aligned} q_k(t_k) = q_k,\, q_k(t_{k+1}) = q_{k+1}, &\qquad k = 0, \ldots, n-1 \\ \dot{q}_k(t_{k+1}) = \dot{q}_{k+1}(t_{k+1}) = v_{k+1}, &\qquad k = 0, \ldots, n-2 \\ \ddot{q}_k(t_{k+1}) = \ddot{q}_{k+1}(t_{k+1}), &\qquad k = 0, \ldots, n-2 \\ \dot{q}_0(t_0) = v_0, & \\ \dot{q}_{n-1}(t_n) = v_n & \end{aligned}$$

Запишем уравнения для k-го полинома, где $T_k = t_{k+1} - t_k$:

$$\begin{cases} q_k(t_k) = a_{k0} = q_k \\ \dot{q}_k(t_k) = a_{k1} = v_k \\ q_k(t_{k+1}) = a_{k0} + a_{k1}T_k + a_{k2}T_k^2 + a_{k3}T_k^3 = q_{k+1} \\ \dot{q}_k(t_{k+1}) = a_{k1} + 2a_{k2}T_k + 3a_{k3}T_k^2 = v_{k+1} \end{cases}$$

Разрешая систему относительно неизвестных коэффициентов, получим:

$$\begin{cases} a_{k0} = q_k \\ a_{k1} = v_k \\ a_{k2} = \frac{1}{T_k}\left[\frac{3(q_{k+1}-q_k)}{T_k} - 2v_k - v_{k+1}\right] \\ a_{k3} = \frac{1}{T_k^2}\left[\frac{2(q_k-q_{k+1})}{T_k} + v_k + v_{k+1}\right] \end{cases}$$

Для решения полученной системы необходимо знать скорости в промежуточных точках, которые мы пока что не знаем. Воспользуемся условиями непрерывности ускорений:

$$\ddot{q}_k(t_{k+1}) = \ddot{q}_{k+1}(t_{k+1}),\ k = 0,\ldots,n-2$$
$$\ddot{q}_k(t_{k+1}) = 2a_{k,2} + 6a_{k,3}T_k,\ k = 0,\ldots,n-2$$
$$\ddot{q}_{k+1}(t_{k+1}) = 2a_{k+1,2},\ k = 0,\ldots,n-2$$

Исходя из этих условий, а также принимая во внимание уравнения параметров $a_{k,2}$, $a_{k,3}$ и $a_{k+1,2}$, после нехитрых математических операций и умножения на $\frac{T_kT_{k+1}}{2}$ получим следующее выражение:

$$T_{k+1}v_k + 2(T_{k+1} + Tk)v_{k+1} + T_kv_{k+2} = \frac{3}{T_kT_{k+1}}[T_k^2(q_{k+2} - q_{k+1}) + T_{k+1}^2(q_{k+1} - q_k)],$$
$$k = 0,\ldots,n-2$$

Эти же выражения можно записать в матричной форме $Av = c$, где:

$$A = \begin{bmatrix} 2(T_0+T_1) & T_0 & 0 & \cdots & & & 0 \\ T_2 & 2(T_2+T1) & T_1 & & & & 0 \\ \vdots & & & \ddots & & & 0 \\ & & & T_{n-2} & 2(T_{n-3}+T_{n-2}) & T_{n-3} & 0 \\ 0 & \cdots & & 0 & T_{n-1} & 2(T_{n-3}+T_{n-2}) & T_{n-1} \end{bmatrix}$$

$$c = \begin{pmatrix} \frac{3}{T_0T_1}[T_0^2(q_2 - q_1) + T_1^2(q_1 - q_0)] - T_1v_0, \\ \frac{3}{T_1T_2}[T_1^2(q_3 - q_2) + T_2^2(q_2 - q_1)] \\ \vdots \\ \frac{3}{T_{n-3}T_{n-2}}[T_{n-2}^2(q_{n-1} - q_{n-2}) + T_{n-2}^2(q_{n-2} - q_{n-3})] \\ \frac{3}{T_{n-2}T_{n-1}}[T_{n-1}^2(q_n - q_{n-1}) + T_{n-1}^2(q_{n-1} - q_{n-2}) - T_{n-2}v_n] \end{pmatrix}$$

$$v = \begin{pmatrix} v_1 & v_2 & \cdots & v_{n-2} & v_{n-1} \end{pmatrix}^{\mathsf{T}}$$

Переменные v_0 и v_n исключены из уравнений, поскольку уже известны. Матрица A обладает свойством строгого диагонального преобладания[11], а следовательно всегда обратима. Это означает, что мы всегда можем вычислить значения скоростей в промежуточных точках: $v = A^{-1}c$. После этого вычислить коэффициенты полиномов не составляет труда.

5.2.2 Параметризация траектории

После интерполяции пути тем или иным способом все еще требуется удовлетворить ограничениям на максимальные и минимальные значения производных. Кроме того, хотелось бы, чтобы траектория была оптимальной во времени.

Предположим, что наш полином зависит не от времени, а от параметра u. Пусть u - функция времени вида $u(t) = \lambda t$. Тогда:

$$\dot{p}(u) = \frac{dp}{du}\frac{du}{dt} = \frac{dp}{du}\lambda$$
$$\ddot{p}(u) = \frac{d^2u}{dt^2}(\frac{du}{dt})^2 = \frac{d^2u}{dt^2}\lambda^2$$
$$\dots$$

В этом случае, чтобы удовлетворить наложенным ограничениям и обеспечить оптимальность траектории выберем λ как:

$$\lambda = min(\frac{v_{max}}{p^1_{max}(u)}, \frac{a_{max}}{\sqrt{p^2_{max}(u)}}, \frac{j_{max}}{\sqrt[3]{p^3_{max}(u)}}, \dots)$$

После параметризации сплайна он уже зависит от времени, участки соответствующие его полиномам можно легко пересчитать по формуле:

$$t_k = u_k/\lambda$$

5.2.3 Результаты

Для примера возьмем следующую траекторию с двумя степенями свободы. Поскольку время прохождения каждой точки совпадает для траекторий для каждой степени свободы, то траектории автоматически синхронизированы во времени.

$$t = \{0, 5, 7, 8, 10, 15, 18\},$$
$$q = \{\{3, -2, -5, 0, 6, 12, 8\}, \{-5, -2, -3, 0, 5, 8, 15\}\},$$
$$v0 = \{2, 0\},\ v1 = \{-3, 0\}$$
$$v_{max} = 10,\ a_{max} = 15,\ j_{max} = 5$$

Результат интерполяции и параметризации представлен на рисунке ниже:

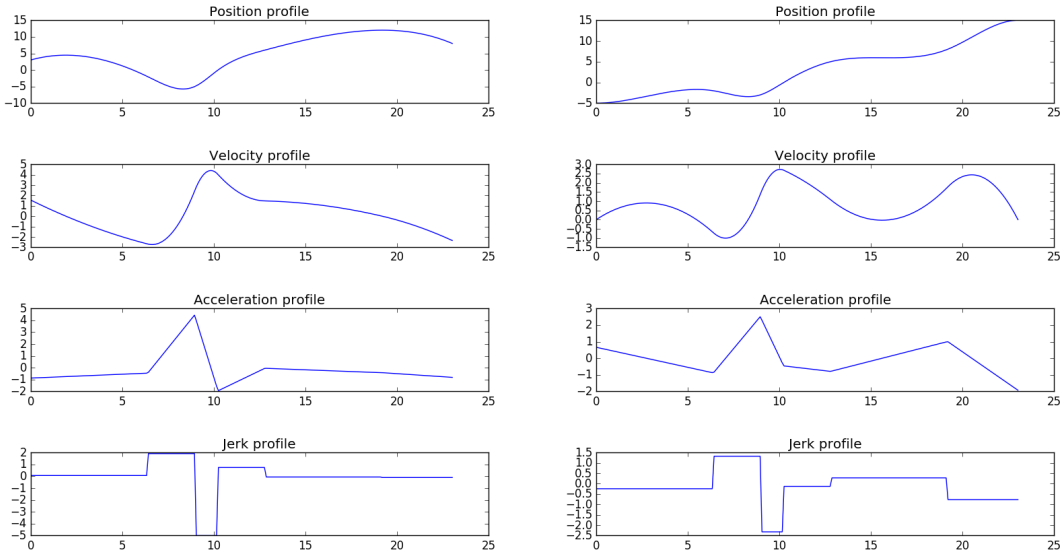

Рисунок 5.1 — Интерполяция и параметризация траектории от времени

Как видно из рисунка, данный метод не обеспечивает непрерывности ускорений в начале и в конце движения, что может быть неприемлемо в некторых задачах. В этом случае можно интерполировать всю траекторию или же только первый и последний участок полиномами пятой степени.

5.3 Планирование траекторий между двумя точками без обхода препятствий

5.3.1 Планирование без обхода препятствий

Существует огромное количество способов спланировать траекторию между двумя точками без обхода препятствий, но мы остановимся на одном из самых популярных выборов для этой задачи - double S-curve траектории, которые обеспечивают гладкость функции положения и скорости, а также непрерывность функции ускорения. Рывок в свою очередь разрывен, но ограничен максимальным значением.

Ограничения накладываемые на траекторию: v_{max}, a_{max}, j_{max}.

Кроме того выбор начальной и конечной скоростей v_0 и v_n остается за нами, а вот начальное и конечное ускорения a_0 и a_n равны нулю.

Траектория имеет три фазы:

– Фаза ускорения $t \in [0,T_a]$

– Фаза максимальной скорости $t \in [T_a, T_a + T_v]$

– Фаза замедления $t \in [T_a + T_v, T]$, где $T = 2T_a + T_v$

Не всегда можно построить траекторию, удовлетворяющую всем наложенным ограничениям. Например, если требуемый путь мал по сравнению с разницей начальной и конечной скоростей, то может оказаться невозможным удовлетворить начальным и конечным значениям скоростей из-за ограничений на максимальные значения ускорения и рывка. В данном случае в траектории присутствует только одна фаза ускорения. Поэтому перед расчетом траектории необходимо убедиться в том, что траекторию можно построить последовательными импульсами рывка: одним - положительным и вторым - отрицательным. Для проверки положим:

$$T_j^* = min \left\{ \sqrt{\frac{|v_1 - v_0|}{j_{max}}}, \frac{a_{max}}{j_{max}} \right\}$$

Если $T_j^* = \frac{a_{max}}{j_{max}}$, ускорение достигает своего максимального значения и участок с нулевым рывком может присутствовать в траектории. Кроме того, траектория может быть построена только при выполнении следующих условий:

$$q_1 - q_0 > \begin{cases} T_j^*(v_0 + v_1), & \text{если } T_j^* < \frac{a_{max}}{j_{max}} \\ \frac{1}{2}(v_0 + v_1)\left[T_j^* + \frac{|v_1 - v_0|}{a_{max}}\right], & \text{если } T_j^* = \frac{a_{max}}{j_{max}} \end{cases} \tag{5.1}$$

Если неравенство 5.1 выполняется, то могут произойти два случая. Пусть $v_{lim} = max(\dot{p(t)})$, тогда:

– Случай 1: $v_{lim} = v_{max}$

– Случай 2: $v_{lim} < v_{max}$

Теперь определим продолжительности каждой из фаз траектории, рассмотрев оба случая. Для начала определим искомые параметры:

T_{j1} – Время интервала в фазе ускорения, на котором рывок постоянен.

T_{j1} – Время интервала в фазе замедления, на котором рывок постоянен.

T_a – Время ускорения.

T_v – Промежуток, на котором скорость постоянна.

T_d – Время замедления.

T – Полное время траектории

Случай 1: $v_{lim} = v_{max}$

В этом случае есть возможность определить, достигается ли максимальное ускорение:

$$\text{Если } (v_{max} - v_0)j_{max} < a_{max}^2 \rightarrow \text{ускорение не достигает } a_{max} \tag{5.2}$$

$$\text{Если } (v_{max} - v_1)j_{max} < a_{max}^2 \rightarrow \text{ускорение не достигает } a_{max} \tag{5.3}$$

Если неравенство 5.2 выполняется, то временные интервалы в фазе ускорения можно вычислить следующим образом:

$$T_{j1} = \sqrt{\frac{v_{max} - v_0}{j_{max}}},\ T_a = 2T_{j1} \tag{5.4}$$

в противном случае:

$$T_{j1} = \frac{a_{max}}{j_{max}},\ T_a = T_{j1} + \frac{v_{max} - v_0}{a_{max}} \tag{5.5}$$

Временные интервалы фазы замедления, в случае выполнения неравенства 5.3, можно вычислить по следующим формулам:

$$T_{j2} = \sqrt{\frac{v_{max} - v_1}{j_{max}}}, \; T_d = 2T_{j2} \tag{5.6}$$

в противном случае:

$$T_{j1} = \frac{a_{max}}{j_{max}}, \; T_d = T_{j2} + \frac{v_{max} - v_1}{a_{max}} \tag{5.7}$$

Наконец, можно определить длину промежутка с константной скоростью:

$$T_v = \frac{q_1 - q_0}{v_{max}} - \frac{T_a}{2}(1 + \frac{v_0}{v_{max}}) - \frac{T_d}{2}(1 + \frac{v_1}{v_{max}}) \tag{5.8}$$

Если $T_v < 0$, это означает, что v_{lim} на самом деле меньше v_{max} и необходимо рассмотреть второй случай.

Случай 2: $v_{lim} < v_{max}$

В этом случае фаза с постоянной скоростью отсутствует($T_v = 0$). Если максимальное(минимальное) значения достигаются, то временные интервалы ускорений и замедлений могут быть вычислены:

$$T_{j1} = T_{j2} = T_j = \frac{a_{max}}{j_{max}} \tag{5.9}$$

а полное время участков:

$$T_a = \frac{\frac{a_{max}^2}{j_{max}} - 2v_0 + \sqrt{\Delta}}{2a_{max}} \tag{5.10}$$

$$T_d = \frac{\frac{a_{max}^2}{j_{max}} - 2v_1 + \sqrt{\Delta}}{2a_{max}} \tag{5.11}$$

где

$$\Delta = \frac{a_{max}^4}{j_{max}^2} + 2(v_0^2 + v_1^2) + a_{max}\left(4(q_1 - q_0) - 2\frac{a_{max}}{j_{max}}(v_0 + v_1)\right) \tag{5.12}$$

Если $T_a < 2T_j$ или $T_d < 2T_j$, то максимальное(минимальное) ускорения не достигаются. В этом случае вычислить траекторию, используя уравнения 5.9, 5.10, 5.11, 5.12, не представляется возможным. Но решение все же есть: можно уменьшать максимальное ускорение $a'_{max} = \gamma a_{max}$, где $0 < \gamma < 1$, до тех пор, пока траектория не построится.

5.3.2 Синхронизация траекторий во времени

Когда траектория строится для нескольких степеней свободы одновременно, то в случае, если конечная скорость для одной из них ненулевая, требуется синхронизация траекторий во времени. При этом заранее невозможно определить, какая из траекторий будет отработана быстрее, для этого требуется полный расчет каждой из траекторий, что может потребовать больших вычислительных ресурсов. Если предположить, что начальные скорости малы относительно требуемых перемещений, то с большой вероятностью траектория с наибольшим требуемым перемещением будет отрабатываться дольше всего. Посколько все траектории оптимальны во времени, построить ее так, чтобы она требовала меньше времени невозможно, но возможно "замедлить"траектории для оставшихся степеней свободы, используя тот же самый метод с уменьшением максимального ускорения на каждой итерации $a'_{max} = \gamma a_{max}$.

5.3.3 Результаты

Построим траекторию в трех степенях свободы:

$$q_0 = \{-2, 0, 10\},\ q_1 = \{20, 85, -10\}$$
$$v_0 = \{0, 5, 0\},\ v_1 = \{2, 4, 0\}$$
$$v_{max} = 30,\ a_{max} = 30,\ j_{max} = 100$$

Результат представлен на рисунке 5.2. Как видно из графиков, первые две траектории синхронизированы во времени, поскольку имеют ненулевые конечные скорости. В то же время для третьей - построена оптимальная во времени траектория, поскольку она имеет нулевую конечную скорость. Кроме того из первых двух наиболее продолжительной является вторая траектория, поэтому для нее расчитана оптимальная во времени траектория, в то время, как первая - "замедлена"до требуемого времени исполнения.

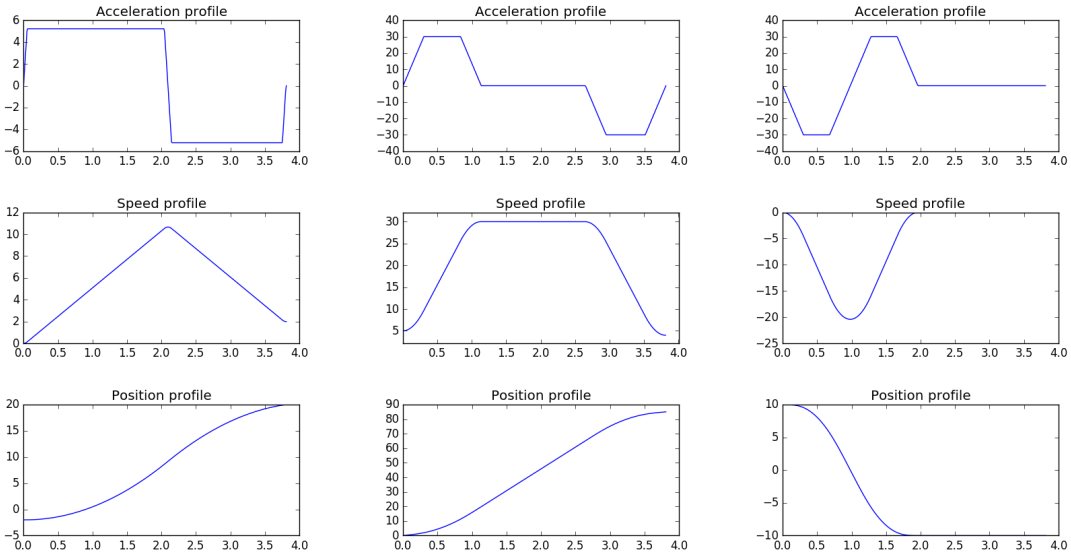

Рисунок 5.2 — Double s-curve траектория для точки имеющей три степени свободы

Глава 6. ROS и система управления

ROS имеет пакет MoveIT, который имеет в себе все необходимое для управления манипуляторами. В частности он включает в себя большое количество различных планировщиков путей, а также интерполятор траекторий. Кроме того, он умеет поддерживать карту окружающей среды, как полученную благодаря обработке данных с датчиков, так и фиксированную. Этот пакет и будет использован для управления *UR10*.

6.1 Выбор алгоритмов

– Планирование путей будет осуществляться с помощью RRT, поскольку это один из самых эффективных способов поиска пути.
– Планирование траекторий будет осуществляться с помощью интерполяции сплайнами 3-й степени.
– Управление реализуется я в Joint Space, а значит требуется решать задачу обратной кинематики. Используется алгоритм Damped Least-Squares, поскольку обеспечивает численную стабильность решения в области сингулярных конфигураций.

Полная схема системы управления представлена на рис. 6.1. Система была про-

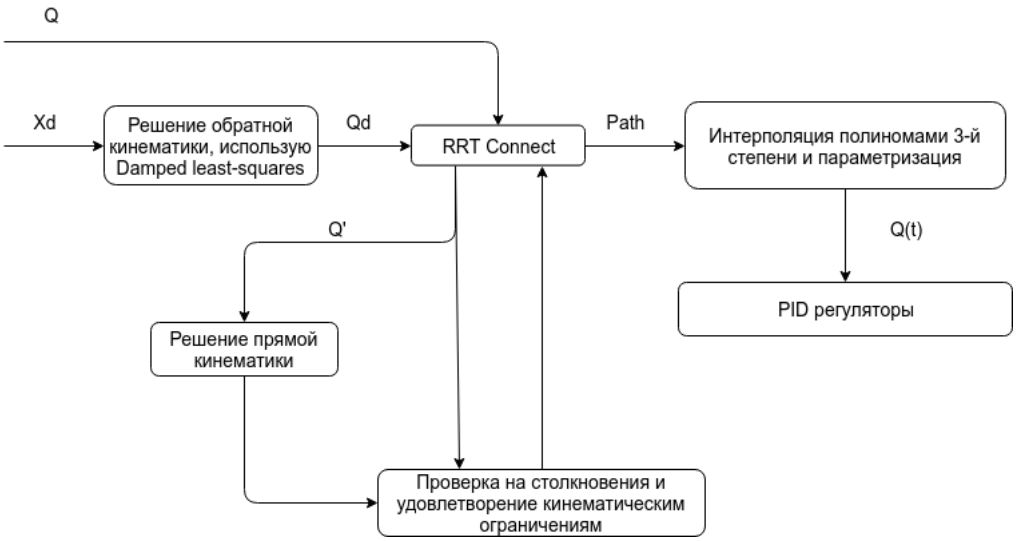

Рисунок 6.1 — Система управления роботом UR10

моделирована и протестирована в среде моделирования *Gazebo*.

Глава 7. Автоматизированное создание карт помещений

7.1 Введение

Поскольку помещения могут быть разных размеров и форм, требуется иметь механизм генерирования карты необходимых элементов, таких как столы и ячейки с продуктами, для любого помещения. Подразумеваются стандартизированные размеры мебели. Таким образом роботу не понадобится дополнительных датчиков, для построения карты помещения в режиме реального времени, что уменьшает стоимость создания и внедрения технологии.

7.2 Язык макросов XACRO

ROS предоставляет удобный механизм создания моделей роботов и карт местности с помощью XML файлов. Но такой подход тяжело параметризируется, поэтому для того, чтобы предоставить пользователям большую гибкость был разработан XML подобный язык макросов XACRO. Он позволяет добавлять в модели роботов параметры, устанавливаемые извне, на выходе обработчика макросов получается обычный файл модели URDF. Кроме того, используя макросы, возможно рекурсивно создавать неограниченное количество объектов. Эта возможность и была использована для создания стандартизированных карт помещений.

На данный момент создано лишь два высокоуровневых макроса:

– add_table - макрос добавляющий в помещение стол. Принимает в качестве параметров положение, масштабирующий коэффициент и координатную систему родителя.

– cells_group_macro - макрос добавляющий набор ячеек в помещение. Принимает в качестве параметров положение, масштабирующий коэффициент, систему координат родителя, а также длину, ширину и высоты, выраженные к количестве ячеек.

Если добавление стола реализуется довольно просто, то создание ячеек намного более трудоемко. Поскольку каждая группа ячеек состоит из нижнего слоя подставок, поднимающих контейнеры на минимальный доступный манипулятору уровень, а также самих ячеек связанных между собой.

К примеру приведенный в A.1 листинг генерирует карту, изображенную на рисунках 7.1 и 7.2. При этом отформатированный XACRO код длиной в

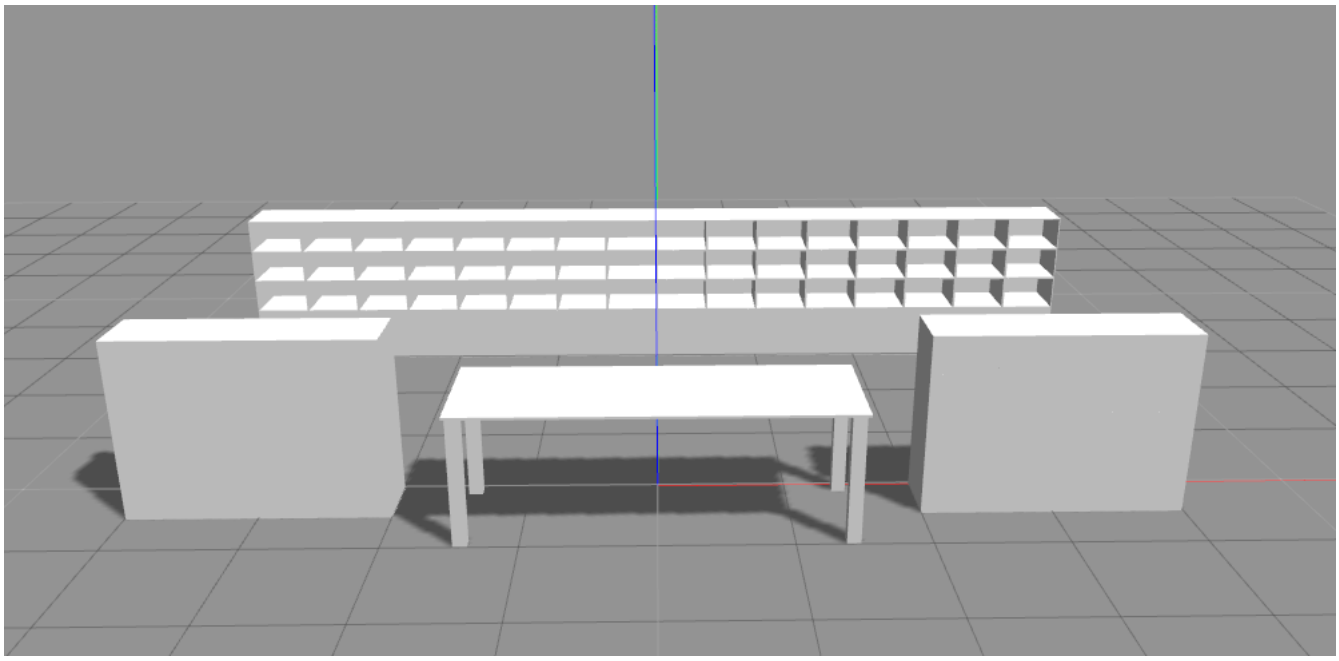

Рисунок 7.1 — Сгенерированная карта. Вид спереди.

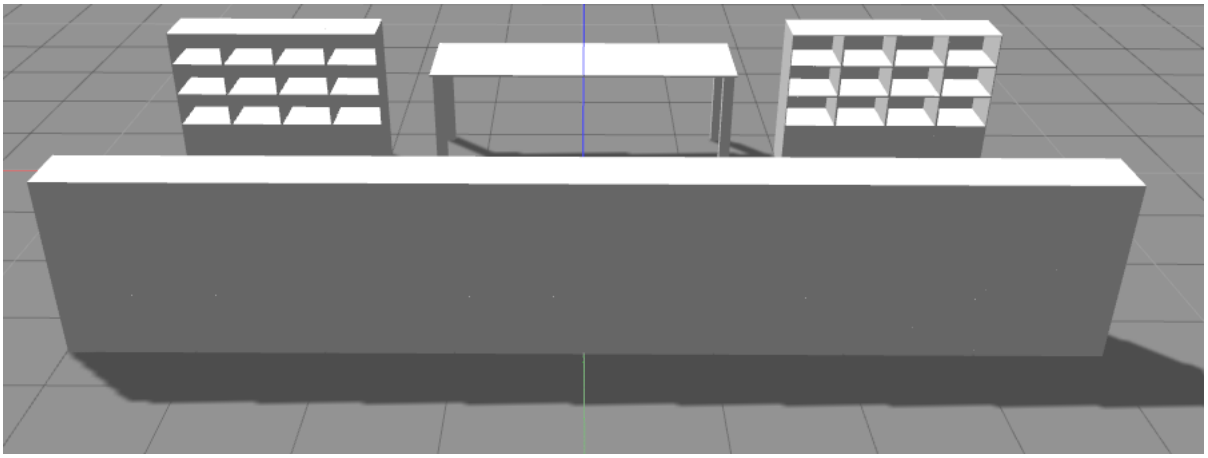

Рисунок 7.2 — Сгенерированная карта. Вид сзади.

44 строки разворачивается в неформатированный URDF код, имеющий 2465 строк.

Заключение

Основные результаты работы заключаются в следующем.

1. Исследованы существующие алгоритмы решения задачи прямой кинематики.
2. Исследованы существующие алгоритмы решения задачи обратной кинематики.
3. Исследованы существующие алгоритмы задачи поиска геометрических путей.
4. Исследованы существующие алгоритмы планирования траектории между двумя точками, а также планирования по полученному геометрическому пути.
5. На основе разобранных алгоритмов была построена система управления.
6. Полученная система была реализована с помощью фреймворка ROS и среды моделирования Gazebo.
7. Был реализован алгоритм автоматической генерации карты помещения, в котором оперирует робот.

В заключение автор выражает благодарность и большую признательность научному руководителю Куприянову Д.В. за поддержку, помощь, обсуждение результатов и научное руководство.

Список литературы

1. Denavit–Hartenberg parameters. — URL: https://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters.
2. OpenRave IKFast manual page. — URL: http://openrave.org/docs/0.8.0/openravepy/ikfast/.
3. *Bruno Siciliano, L. S.* Robotics. Modelling, planning and control / L. S. Bruno Siciliano. — Springer, 2009.
4. *Томас Кормен, Ч. Э. Л.* Алгоритмы: построение и анализ / Ч. Э. Л. Томас Кормен. — 2-е изд. — MIT Press, 2007.
5. A* search algorithm. — URL: https://en.wikipedia.org/wiki/A*_search_algorithm.
6. Coursera "Robotics: Computational Motion Planning".
7. Rapidly-exploring random tree. — URL: https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree.
8. *James Cuffner, S. L. V.* RRT-Connect: An Efficient Approach to Single-Query Path Planning. / S. L. V. James Cuffner. — 2000.
9. K-D trees. — URL: https://en.wikipedia.org/wiki/K-d_tree#Nearest_neighbour_search.
10. *Luigi Biagiotti, C. M.* Trajectory Planning for Automatic Machines and Robots / C. M. Luigi Biagiotti. — Springer, 2008.
11. Диагональное преобладание. — URL: https://en.wikipedia.org/wiki/Diagonally_dominant_matrix.

Приложение А

Листинги программного кода

Листинг А.1 Пример генерации карты с помощью XACRO

```
<?xml version="1.0"?>
<robot name="cells_env" xmlns:xacro="http://www.ros.org/wiki/xacro">
        <xacro:include filename="building_parts/cell_group.urdf.xacro" />
        <xacro:include filename="building_parts/table.urdf.xacro" />
        <xacro:property name="mesh_scale" value="0.001 0.001 0.001" />
        <link name="world" />
        <link name="base_link" />
        <link name="foodcort" />
        <joint name="world_joint" type="fixed">
                <parent link="world"/>
                <child link="base_link"/>
                <origin rpy="0.0 0.0 0.0" xyz="0.0 0.0 0.0" />
        </joint>
        <joint name="foodcort_joint" type="fixed">
                <parent link="base_link" />
                <child link="foodcort" />
                <origin rpy="0.0 0.0 0.0" xyz="1.5 0.0 0.0"/>
        </joint>
        <xacro:add_table id="0" parent="foodcort" scale="${mesh_scale}">
                <origin rpy="1.57 0.0 0.0" xyz="-3.0 0.0 0.0" />
        </xacro:add_table>
        <xacro:cells_group_macro id="0" parent="foodcort" x_size="4" z_size="3"
                scale="${mesh_scale}">
                <origin rpy="1.57 0.0 0.0" xyz="0.5 0.0 0.0" />
        </xacro:cells_group_macro>
        <xacro:cells_group_macro id="1" parent="foodcort" x_size="4" z_size="3"
                scale="${mesh_scale}" dir="${cell_right}">
                <origin rpy="1.57 0.0 0.0" xyz="-4.0 0.0 0.0" />
        </xacro:cells_group_macro>
        <xacro:cells_group_macro id="2" parent="foodcort" x_size="16"
                z_size="3" scale="${mesh_scale}">
                <origin rpy="1.57 0.0 3.14" xyz="2.5 3.0 0.0"/>
        </xacro:cells_group_macro>
</robot>
```