Разработка мобильного четырехногого робота

Коломейцев А.А.

НИУ "МЭИ"

27 мая 2020 г.

АННОТАЦИЯ

В рамках работы рассматривается разработка шагающего четырехногого робота, с упором на проблему разработки конечностей, подбора электроприводов и их управления. Сравнены разные типы электроприводов, пригодных для создания конечностей для шагающих роботов. Рассчитаны соответствующие нагрузки, рассмотрены разные типы управления в зависимости от требований к качеству перемещения робота в режиме ходьбы.

ОГЛАВЛЕНИЕ

АННОТАЦИЯ 1

ВВЕДЕНИЕ 4

ГЛАВА 1 ШАГАЮЩИЕ РОБОТЫ **6**

1.1 Задачи работы 6

1.2 Актуальность работы 6

1.3 Анализ 7

1.3.1 Рассчёт худшего статического случая 7

1.3.2 Подбор электроприводов 8

ГЛАВА 2 МЕХАНИЧЕСКАЯ КОНСТРУКЦИЯ **10**

2.1 Проектирование ног 10

2.2 Проектирование корпуса 11

2.3 Подбор комплектующих 12

2.3.1 Силовая электроника 12

2.3.2 Логическая электроника 12

2.4 Аккумулятор 13

2.5 Остальные детали 14

ГЛАВА 3 КИНЕМАТИКА КОНЕЧНОСТЕЙ РОБОТА **16**

3.1 Общее положение 16

3.2 Общее решение четырёхзвенника 18

3.3 Прямая кинематика 20

3.4 Обратная кинематика 22

3.5 Оптимизация численного решения обратной задачи 25

ГЛАВА 4 ОРИЕНТАЦИЯ ТЕЛА РОБОТА **28**

4.1 Кватернионы 28

4.2 Произвольные повороты тела в пространстве 28

4.3 Связь ориентации тела с конфигурацией ног 28

ГЛАВА 5 МОДЕЛИРОВАНИЕ ХОДЬБЫ **29**

5.1 Траектории движения ног 29

5.2 Проблема выбора оптимальной траектории 29

ГЛАВА 6 ПРОГРАММНАЯ АРХИТЕКТУРА **30**

6.1 Структура управления 30

6.2 Протокол передачи данных 30

6.3 Кинематические расчеты 32

6.3.1 Вычисление графа прямой кинематики 33

6.3.2 Решение тригонометрической задачи о четырехзвеннике 33

6.3.3 Вычисление Якобиана 34

6.4 Тестирование кода 35

6.5 Абстракция над вычислениями 35

ГЛАВА 7 ЗАКЛЮЧЕНИЕ **36**

БИБЛИОГРАФИЧЕСКИЙ СПИСОК 37

ПРИЛОЖЕНИЕ А: КОМПЛЕКТУЮЩИЕ 38

ПРИЛОЖЕНИЕ Б: КОД КЛАССА 40

ВВЕДЕНИЕ

Шагающие роботы - это класс роботов, имитирующих движения людей и животных. Миллионы лет эволюциии показывают, что передвижение при помощи ног это наиболее эффективный способ быстро приспосабливаться к плохим, неровным поверхностям. Люди пытались описывать ходьбу шагающими механизмами, математическими формулами. На сегодняшний день понятно, что ни один из этих двух способов к результатам, применимым на практике, не приведет. От шагающей системы требуется приспособиться к тем условиям, в которых она раньше не была.

Рынок шагающих роботов

Классифицировать шагающие машины можно не только по количеству ног. Некоторые роботы комплектуются также и колесами, для увеличения скорости передвижения по ровным поверхностям.

На текущий момент можно найти огромное количество шагающих роботов. Гексаподы, робо-пауки и т.д. Но среди сотен моделей можно выделить всего несколько роботов, ходьба которых максимально приближена к животной.

Из четырёхногих роботов лучшие результаты показывают:

- Роботы Spot и Spot-Mini от компании Boston Dynamics
- Робот Mini-Cheetah от студентов MIT
- Робот ANYmal от студентов Цюрихского университета
- Робот HyQReal от IIT (Итальянского технологического института)
- Роботы LaikaGo и AlienGo от компании Unitree Robotics

Среди двуногих можно выделить такие проекты, как:

- Робот Digit от компаний Agility Robotics и Ford
- Робот Cassie от компании Agility Robotics
- Робот Atlas от компании Boston Dynamics

Всех выше перечисленных роботов объединяет одно важное свойство - они обучены ходить. Алгоритм их ходьбы не описан статическими константами в коде, он создан при помощи методов машинного обучения. Благодаря

этому все они показывают высокую степень мобильности и адаптивности к окружающей среде. Их перемещения похожи на то, как могли бы перемещаться животные со схожим механическим строением тела.

Будущее шагающих роботов именно за машинным обучением, а если быть точнее, за глубоким обучением с подкреплением (reinforcement learning). Модели, обученные в симуляции показывают более высокий КПД при перемещении и меньшее потребление тока, чем модели ходьбы, описанные человеком вручную [1].

ГЛАВА 1

ШАГАЮЩИЕ РОБОТЫ

1.1 Задачи работы

В рамках работы рассматривается разработка шагающего четырехного-го робота, с упором на проблему разработки конечностей, подбора электроприводов и их управления. Сравниваться будут два типа электроприводов: заводские микро-сервоприводы с редуктором и бесколлекторные синхронные двигатели (BLDC).

Требуется:

- Рассчитать худший (по нагрузке) статический случай для конечностей 4-х ногого робота.
- Подобрать два электропривода удовлетворяющих по крутящему моменту. Для двух типов электроприводов разработать модель конечностей.
- Собрать физическую модель конечностей двух типов.
- Сравнить управляемость и механические характеристики конечностей двух типов.
- Применить самую удачную конструкцию конечностей при сборке робота.
- Запрограммировать управление робота.

1.2 Актуальность работы

Задача разработки конечностей для шагающих роботов настолько же важная, как и задача навигации роботов. Сегодня она актуальна, как никогда ранее. Всё в большей степени людей стараются заменять шагающими роботами для работ вроде общего тех. осмотра помещений, исследования местности вдали от дорог и цивилизации, помощи в устранении последствий катастроф. Открытость методик, исходных кодов и готовых рассчитанных моделей приведет к массовой разработке шагающих роботов не только крупными предприятиями, но и мелкими разработчиками.

1.3 Анализ

Для поставленных ранее задач можно сформировать последовательность действий, которые приведут к их решению.

1.3.1 Рассчёт худшего статического случая

«Худшим» случаем называется такой, при котором одному или нескольким приводам нужно приложить максимальный момент для поворота звена конечности в нужную сторону.

Для того чтобы определить худший случай, нужно вообще определиться с кинематикой конечности. Оптимальной кинематикой можно считать конечность с 3 степенями свободы:

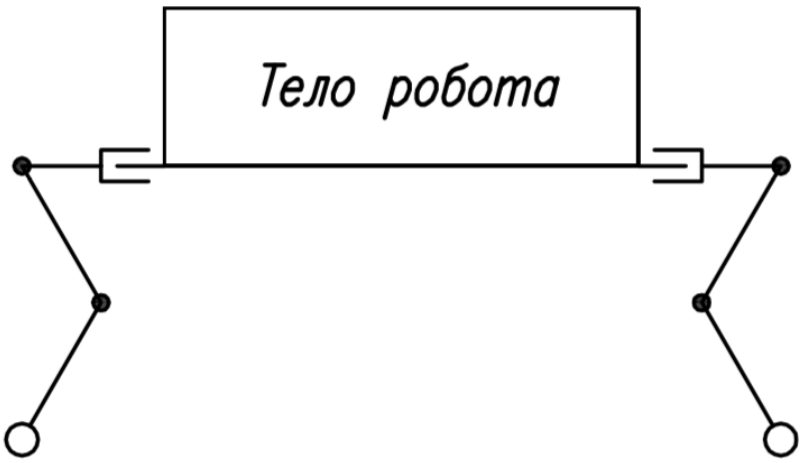


Рисунок 1.1 — Кинематика 4-х ногого робота, вид сбоку

Худшим случаем считается случай, в котором робот лежит на «животе» с выпрямленной конечностью. Чтобы подогнуть под себя конечность, нужно будет преодолеть момент $M_{\text{худш}}$ с учетом массы тела робота m.

При расчёте в первом приближении можно пренебречь трением и весами звеньев. Также можно учесть, что нагрузка, создаваемая массой тела робота будет распределена равномерно по всем 4-м ногам. Это значит что на одну ногу будет приходится лишь $\frac{1}{4}m$ тела робота.

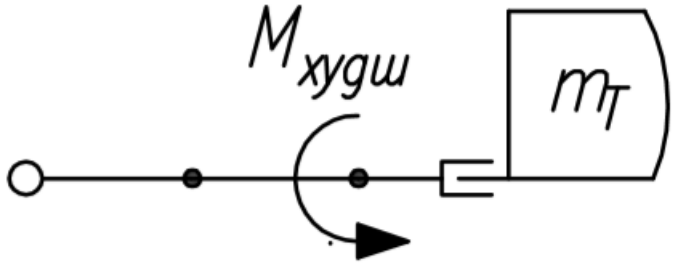


Рисунок 1.2 — Кинематика 4-х ногого робота, вид сбоку (худший случай)

Также, в первом приближении допустим, что тело робота может достигать 2-х килограмм. Большая часть веса придется на каркас конструкции, собранный из пластика. Меньшая часть веса придется на аккумулятор. Еще более маленькая часть придется на проводку и крепежи. Самыми легкими составляющими конструкции окажутся электронные компоненты.

При таком расчете можно считать что каждой ноге надо будет «поднять» около 0.25 кг веса. Тогда в худшем случае, приложенный момент вычисляется просто:

$$M_{\text{худш}} = (l_1 + l_2)mg,$$

здесь l_1 и l_2 - длины звеньев.

Мы можем подобрать длины звеньев таким образом, чтобы они обеспечивали достаточную для задач ходьбы рабочую область и одновременно наименьший требуемый момент. Однако сначала нам нужно будет измерить реальные моменты на подобранных электроприводах.

1.3.2 Подбор электроприводов

Исходя из ожидаемых моментов и скоростей были выбраны два вида электроприводов:

- Заводской сервопривод DSSERVO RDS3225
- BLDC-мотор DYS BGM5208-200-12

Использовать паспортные данные двигателей для расчётов нельзя, нужно измерить их реальный момент вращения. Для измерения крутящих мо-

ментов были разработаны стенды, принцип действия которых можно описать следующей схемой:

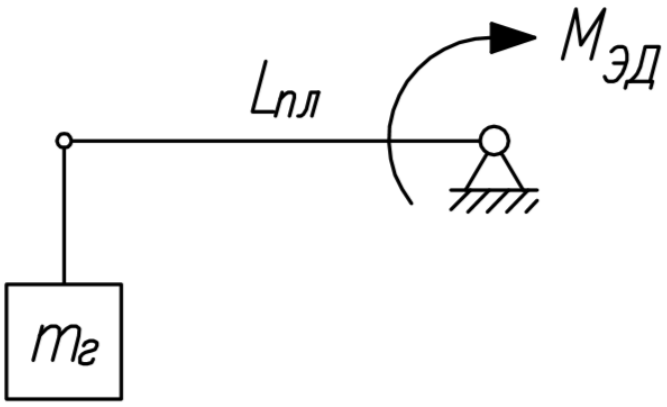
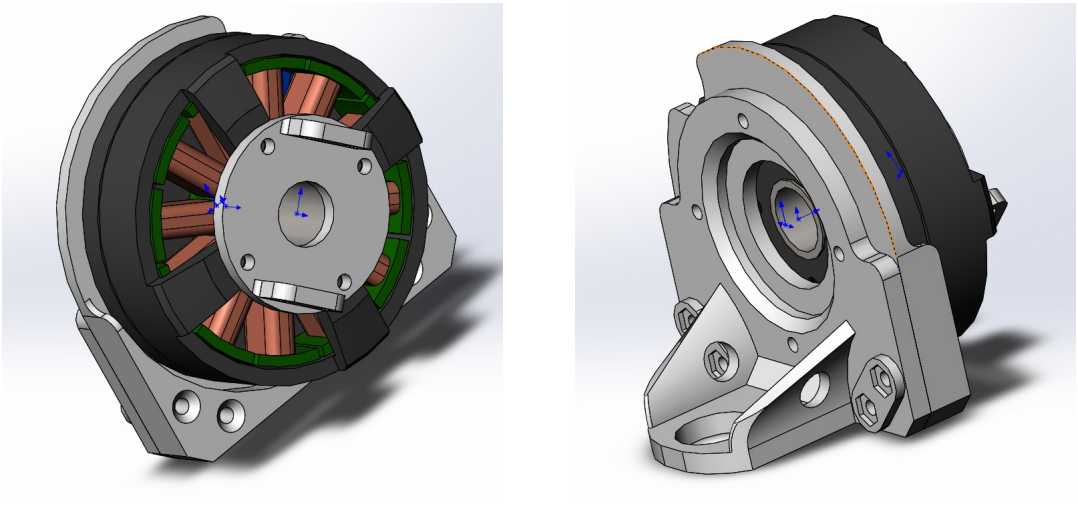


Рисунок 1.3 — Кинематика стенда для измерения крутящего момента

На изображении $m_{\text{г}}$ - масса груза, которая нам известна, и которую мы можем менять. $L_{\text{пл}}$ - длина плеча, также известная нам. Из приведенных величин можно легко найти экспериментально значение момента $M_{\text{ЭД}}$.

Чертеж стенда для BLDC двигателя:



(а) (б)

Рисунок 1.4 — (а) Стенд для измерения крутяжего момента BLDC мотора, вид спереди; (б) Тот же стенд, вид сзади.

ГЛАВА 2

МЕХАНИЧЕСКАЯ КОНСТРУКЦИЯ

2.1 Проектирование ног

Основная, и самая сложная с точки зрения механики часть шагающего робота, это конечности. Как и корпус, ноги проектировались в виде трёхмерных, твердотельных чертежей. Для уменьшения количества уникальных деталей конструкция всех четырех ног была унифицирована. Таким образом снижена сложность и затратность в производстве деталей.

Сложные по форме детали изготовлены из пластика на 3D-принтере, в конструкции также присутствуют металлические стержни, стандартные металлические кронштейны, крепежные элементы (винты, гайки).

Перед проектированием были выдвинуты функциональные требования к конечностям. Они должны быть как можно менее габаритными, для сохранности редукторов сервоприводов и для увеличения скорости движения нужно максимально уменьшить момент инерции конечности. Основной способ достижения этой цели – уменьшение веса всех деталей. Поэтому крепежные и элементы корпуса, были изготовлены из пластика, там где это было возможно. Были использованы полые цилиндрические аллюминиевые трубки в качестве стержней, обеспечивающих жесткость.

Есть еще один, не менее эффективный способ снизить момент инерции ног, не уменьшая общего веса конечностей – концентрация основной массы как можно выше, ближе к месту крепления ноги к корпусу. Поэтому сервоприводы, как одни из самых тяжелых элементов конструкции, были перенесены максимально близко к корпусу и максимально далеко от пола.

В связи с этим в конструкции возник механический четырехзвенник, позволяющий для вращения последнего звена ноги установить сервопривод не непосредственно в шарнир, а передать вращение двигателя издалека. Четырехзвенник усложнил кинематическую схему ноги, его расчет рассмотрен далее в пункте 3.2, однако такая конструкция не только хорошо повлияла на механические характеристики ноги, но и на эстетические тоже. Крутящий

момент передается при помощи кинематического четырехзвенника и двигатель, который имеет неоптимальные габариты, не пришлось размещать в последнем узле ноги, что позволило сделать конечность максимально компактной.

В конструкции предусмотрена установка датчиков касания на стопы, в ходе развития проекта. Стопы можно будет заменить на более совершенные, а конструкция ноги позволит без проблем проложить провода внутри стержней, либо сокрытыми за стенками корпуса.

Из-за специфики сервоприводов возникают проблемы во время сборки, которые замедляют процесс сборки и требуют наличия управляющей электроники. Дело в том что при установке в конструкцию ноги, сервопривод должен быть верно сконфигурирован, или проще говоря, быть заранее повернутым на известный далее управляющей программе угол поворота. Так называемый технический угол. Этим минусом обладают все сервоприводы такого типа. В других видах приводов конфигурировать углы поворота при сборке не нужно, но каждый раз при запуске нужно калибровать приводы, поворачивая их в нулевое, начальное положение. Такие приводы не были использованы в силу своей дороговизны и в силу того что наличие концевых выключателей в конструкции конечности сильно бы усложнило разработку. Для прототипа это излишне.

2.2 Проектирование корпуса

Требования к корпусу можно разделить на три составляющие: массовые, габаритные и эстетические. Снижать массу нужно для того чтобы разгрузить конечности робота, защитить редукторы электроприводов. Снижению массы способствует максимальный отказ от металлических деталей, а там где это невозможно (в силу требований по жесткости), нужно использовать эффективные сечения профилей, желательно из аллюминия.

В качестве каркаса, к которому крепятся ноги робота был выбран конструкционный аллюминиевый профиль, из-за его легкости, жесткости и простоты крепления новых деталей к профилю.

Объем корпусу придают тонкие цилиндрические аллюминиевые стержни, в узлах пластиковые крепежи. Такая конструкция позволяет использовать максимально много места внутри корпуса, что упрощает размещение электронных компонентов внутри.

Самой сложной задачей была компоновка всех компонентов внутри корпуса, так как нужно сохранить их "доступность"в любой момент времени для замены или диагностики. Таким образом повышается общая ремонтопригодность прототипа, а это сильно ускоряет работу с ним.

2.3 Подбор комплектующих

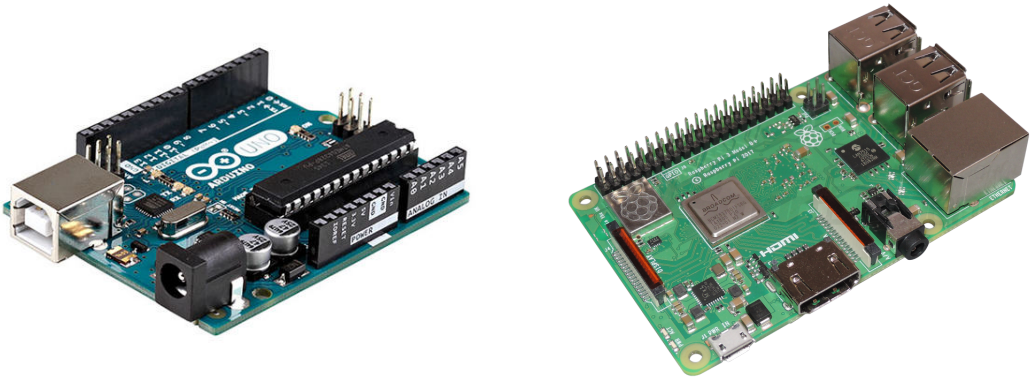
2.3.1 Силовая электроника

Из силовой электроники в роботе присутствуют:

- Электрические сервоприводы
- *DC-DC* преобразователи напряжения.

2.3.2 Логическая электроника

В качестве управляющего микрокомпьютера выступает *Raspberry Pi 3B+*. Этого достаточно для совершения множества матричных операций в секунду и для эффективной работы с высокоуровневыми абстракциями в коде.



(а) *Arduino Uno* (б) *Raspberry Pi 3B+*

Рисунок 2.1 — Логическая электроника в составе робота

Raspberry Pi выступает исключительно как своеобразный модуль для высокоуровневых вычислений и не работает с «железом» напрямую. Вместо этого происходит отправка команд на микроконтроллер в составе платформы *Arduino*. Команды формируются и передаются в цифровом виде на микроконтроллер. Сделано это по двум причинам:

- Отделение логики вычислений состояния робота от самого процесса управления.
- Электрическая защита дорогостоящего микрокомпьютера от возможных скачков тока в цепи силовой электроники.

О протоколе передачи данных и способе формирования команд подробнее написано в пункте 6.2.

2.4 Аккумулятор

В качестве аккумулятора был использован Литий-полимерный аккумулятор (*Li-Po* аккумулятор). Преимущества такого выбора приведены ниже:

- Высокая токоотдача
- Большая емкость
- Быстрая зарядка
- Наличие дешевой электроники для контроля разряда таких аккумуляторов
- Малый вес и габариты

С такими большими плюсами существуют и минусы, которые стоит принимать во внимание при эксплуатации такого типа аккумуляторов:

- Высокая токоотдача повышает риск получить травму при работе
- Высоки шансы на сгорание питающейся от аккумулятора электроники в случае короткого замыкания
- Требуются специальные условия хранения, если аккумуляторы долгое время не используются
- *Li-Po* аккумуляторы взрывоопасны при неправильной эксплуатации

Из-за большего опыта работы с *Li-Po* аккумуляторами (несколько по-

Рисунок 2.2 — *Li-Po* аккумулятор на 2600 мА·ч

следних лет) и было решено использовать их.

2.5 Остальные детали

Пластиковые детали изготовливались на *3D*-принтере методом *FDM* печати (послойной) из материала *PET-G*. Преимущества материала для данного робота описаны ниже:

- Прочность *ABS*
- Термостойкость *ABS*
- Долговечность *ABS*
- Не требователен к условиям печати, как *PLA*
- Низкая термоусадка (почти не меняет размеры при нагревании/остывании)
- Высокие ударопрочные свойства
- Возможность красить и стерилизовать
- Не токсичен

Может показаться что при изготовлени деталей на *3D*-принтере последним можно придавать любую форму, но при *FDM* технологии печати это не так. Технология достаточно дешева и может использоваться в дешевых домашних *3D*-принтерах, поэтому у нее есть свои ограничения на печать «нависающих» над печатной областью частей детали. Поэтому при проектировании деталей из пластика нужно исходить из двух принципов. Первый

заключается в том чтобы не делать новой детали из пластика, если есть альтернатива в линейке деталей, который изготавливаются по ГОСТ и продаются в специализированных магазинах. Второй принцип заключается в том чтобы использовать в детали как можно меньше материала и скомпоновать ее так, чтобы при печати максимально избежать создания поддержек для нависающих частей. Всё это накладывает свои конструктивные ограничения и требует от конструктора некоторого опыта.

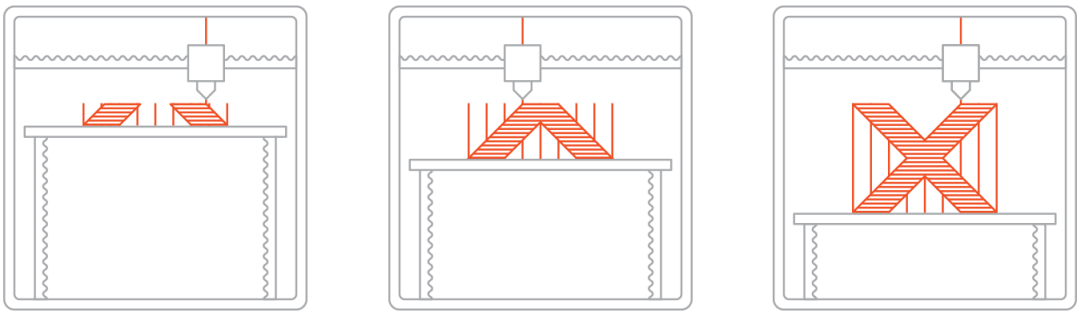


Рисунок 2.3 — Иллюстрация *3D*-печати по технологии *FDM*

Из металла выполнены покупные кронштейны, стержни ног. Стержни – стандартные полые цилиндрические аллюминиевые профили. Подгонялись под нужную длину при помощи ножовки. Крепление к пластиковым узлам происходило вкручиванием в торцы стержней винтов. Такой способ приводит к хорошему зацеплению деталей.

Для того чтобы избежать разрушения пластиковых деталей в месте вкручивания винта в торец, и из-за невозможности нарезать резьбу в пластиковой детали, были использованы вставки из металлических гаек внутри пластикового материала.

ГЛАВА 3

КИНЕМАТИКА КОНЕЧНОСТЕЙ РОБОТА

3.1 Общее положение

Изначально планировалось, что конечность робота в общем виде будет представлять манипулятор с тремя степенями свободы. Такая конструкция множество раз рассмотрена другими людьми, существует аналитическое решение прямой и обратной задачи.

Проблема такой кинематики в механической сложности ее реализации с точки зрения конструктора. Из-за особенностей и требований, описанных в пункте 2.1, конструкция ноги получилась, как на рисунке ниже.

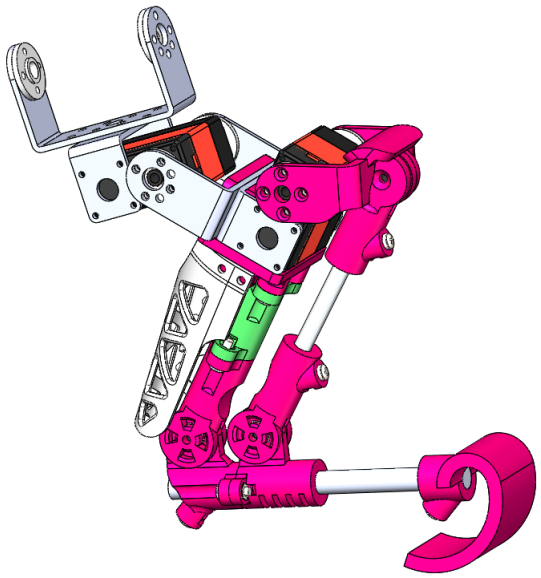


Рисунок 3.1 — Чертеж конструкции ноги

Для упрощения понимания ниже приведена трехмерная кинематическая схема конечности.

В составе конечности присутствует непрямой кинематический четырехзвенник (три из четырех сторон имеют разные длины). Это усложнило кинематическую задачу, сделало ее менее тривиальной. Как оказалось далее, наличие четырехзвенной передачи сделало невозможным нахождение аналитического решения обратной задачи кинематики.

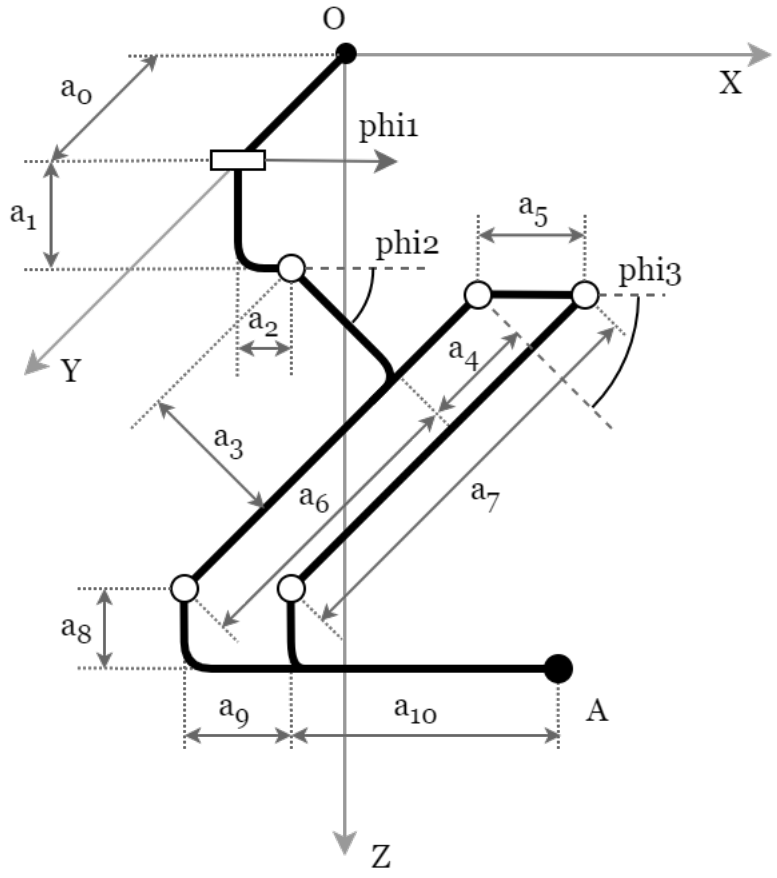


Рисунок 3.2 — Трехмерная кинематическая схема ноги робота

На кинематической схеме отмечены величины, численные значения которых приведены ниже:

$$a_0 = 0.01\text{м}$$
$$a_1 = 46.22 \times 10^{-3}\text{м}$$
$$a_2 = 20 \times 10^{-3}\text{м}$$
$$a_3 = 44 \times 10^{-3}\text{м}$$
$$a_4 = 20 \times 10^{-3}\text{м}$$
$$a_5 = 27 \times 10^{-3}\text{м}$$
$$a_6 = 87 \times 10^{-3}\text{м}$$
$$a_7 = 107 \times 10^{-3}\text{м}$$
$$a_8 = 12.62 \times 10^{-3}\text{м}$$
$$a_9 = 24.5 \times 10^{-3}\text{м}$$
$$a_10 = 110 \times 10^{-3}\text{м}$$

Заметим, что расстояния a_5 и a_9 различаются! Диапазоны рабочих углов следующие:

$$\varphi_1 = 0 \dots \frac{\pi}{8}$$
$$\varphi_2 = -\frac{\pi}{2} \dots 0$$
$$\varphi_3 = -\frac{\pi}{4} \dots \frac{\pi}{4}$$

Если зафиксировать первую степень свободы, $\varphi_1 = 0$, тогда рабочая область ноги будет лежать в плоскости XZ. Выглядеть рабочая область с зафиксированным φ_1 будет следующим образом:

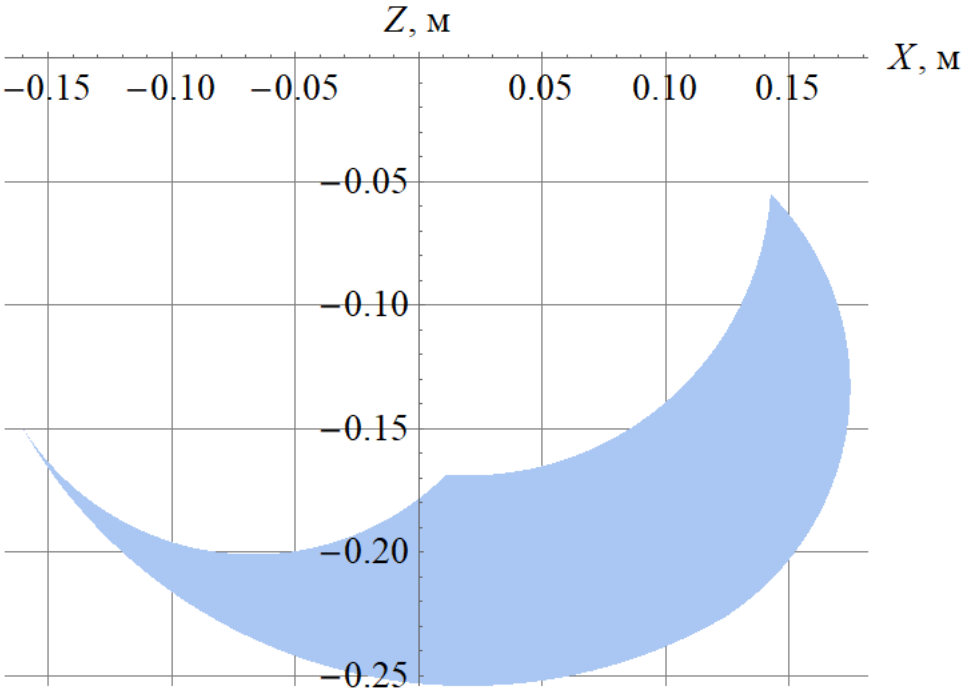


Рисунок 3.3 — Сечение рабочей области конечности робота

Для построения рабочей области использованы уравнения прямой кинематики, которые будут выведены в пункте 3.3.

3.2 Общее решение четырёхзвенника

При построении аналитического решения четырехзвенной передачи было важно подобрать функции так, чтобы для рабочих диапазонов углов $\varphi_1, \varphi_2, \varphi_3$ не возникало переходов через ноль, или через бесконечно боль-

шие числа. Важно избежать использования кусочных функций и условных операторов для упрощения программирования. Малые диапазоны углов в рассматриваемой конечности упрощают эту задачу.

На рисунке 3.4 приведена схема четырехзвенника. Найдем зависимость φ_4 от φ_3. Здесь $l = a_4 + a_6 = a_7$ введен для упрощения расчётных формул, т.к. две стороны четырехзвенника имеют одну длину.

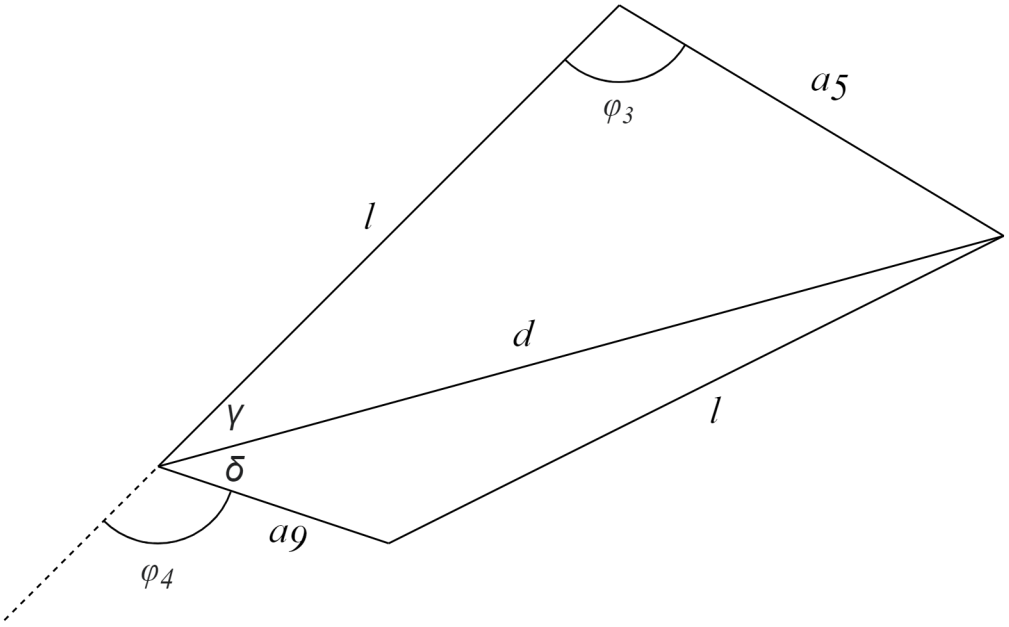


Рисунок 3.4 — Четырехзвенник

Сначала найдем диагональ d по теореме косинусов, для того чтобы выразить противолежащие углы:

$$d = \sqrt{a_5^2 + l^2 - 2a_5 l \cos(\varphi_3)} \tag{3.1}$$

С помощью уравнения 3.1 выразим углы:

$$\gamma = \arcsin\left(\frac{a_5}{d}\cos\varphi_3\right) \tag{3.2}$$

$$\delta = \arccos\left(\frac{d^2 + a_9^2 - l^2}{2a_9 l}\right) \tag{3.3}$$

И искомый угол φ_4 будет находиться следующим образом:

$$\varphi_4 = \pi - \gamma - \delta \tag{3.4}$$

Подставим результаты 3.1, 3.2, 3.3 в уравнение 3.4 таким образом получим зависимость φ_4 от φ_3:

$$\varphi_4 = \pi - \arcsin\left(\frac{a_5}{\sqrt{a_5^2 + l^2 - 2a_5 l\cos(\varphi_3)}}\cos\varphi_3\right) - \\ - \arccos\left(\frac{a_5^2 + l^2 - 2a_5 l\cos(\varphi_3) + a_9^2 - l^2}{2a_9 l}\right) \tag{3.5}$$

3.3 Прямая кинематика

Используя найденную в пункте 3.2 связь построим решение прямой задачи кинематики по координатам точки A. Трехмерная кинематическая схема приведена на рисунке 3.2. Зафиксируем сначала первую степень свободы ($\varphi_1 = 0$) и найдем координаты точки A в плоскости, параллельной плоскости XZ:

$$X_A = a_2 + a_3\cos(\varphi_2) + a_6\sin(\varphi_2) - \\ - a_8\cos(\varphi_2 + \varphi_4) + (a_9 + a_{10})\sin(\varphi_2 + \varphi_4) \tag{3.6}$$

$$Z_A = a_1 - a_3\sin(\varphi_2) + a_6\cos(\varphi_2) + \\ + a_8\sin(\varphi_2 + \varphi_4) + (a_9 + a_{10})\cos(\varphi_2 + \varphi_4) \tag{3.7}$$

Если мы предположим что $\varphi_1 = \frac{\pi}{2}$, тогда конечность будет «поднята над землей» в таком положении. Координата Z точки A станет равна нулю, координата Y увеличится. Так как от φ_1 зависят только координаты Y и Z, получим следующие уравнения для координат точки A:

$$X_A = a_2 + a_3\cos(\varphi_2) + a_6\sin(\varphi_2) - \\ - a_8\cos(\varphi_2 + \varphi_4) + (a_9 + a_{10})\sin(\varphi_2 + \varphi_4) \tag{3.8}$$

$$Y_A = a_0 + \sin(\varphi_1)[a_1 - a_3\sin(\varphi_2) + a_6\cos(\varphi_2) + \\ + a_8\sin(\varphi_2 + \varphi_4) + (a_9 + a_{10})\cos(\varphi_2 + \varphi_4)] \quad (3.9)$$

$$Z_A = \cos(\varphi_1)[a_1 - a_3\sin(\varphi_2) + a_6\cos(\varphi_2) + \\ + a_8\sin(\varphi_2 + \varphi_4) + (a_9 + a_{10})\cos(\varphi_2 + \varphi_4)] \quad (3.10)$$

Таким образом прямая задача кинематики решена.

Следует отметить, что из-за того что $a_9 < a_5$, диапазон углов φ_4 получился шире, чем у φ_3. Это наглядно можно продемонстрировать на проекции математической модели ноги на плоскость XZ. Здесь синей точкой отмечено положение реальное положение точки A. Красной точкой отмечено положение точки A в том случае, если бы в четырехзвеннике a_5 был равен a_9. В крайних положениях третье звено конечности поворачивается на угол немного больший, чем поворачивается вал сервопривода.

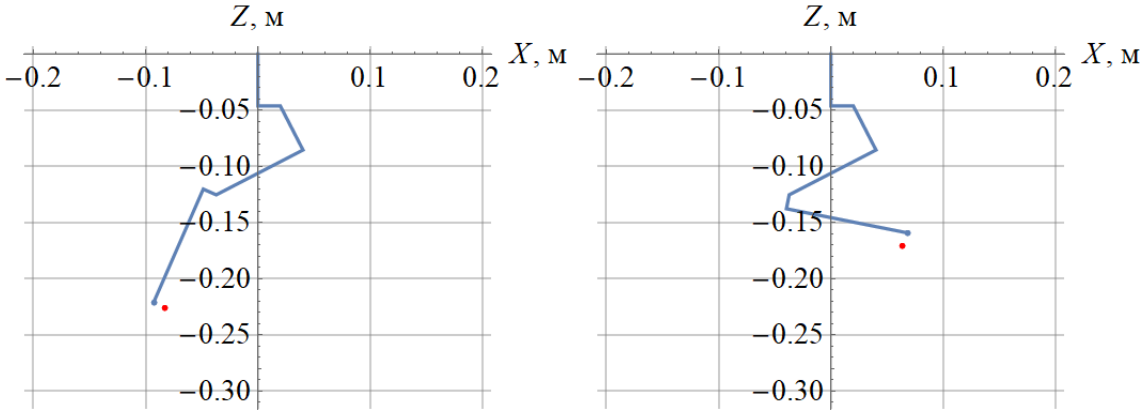


Рисунок 3.5 — (а) Третье звено полностью «разогнуто»; (б) Третье звено полностью «согнуто».

Таким образом за счет использования четырехзвенной передачи, увеличена рабочая область конечности. Ниже на графике зеленая область – рабочая область при $a_5 = a_9$, синяя область – расширение рабочей области при $a_5 > a_9$.

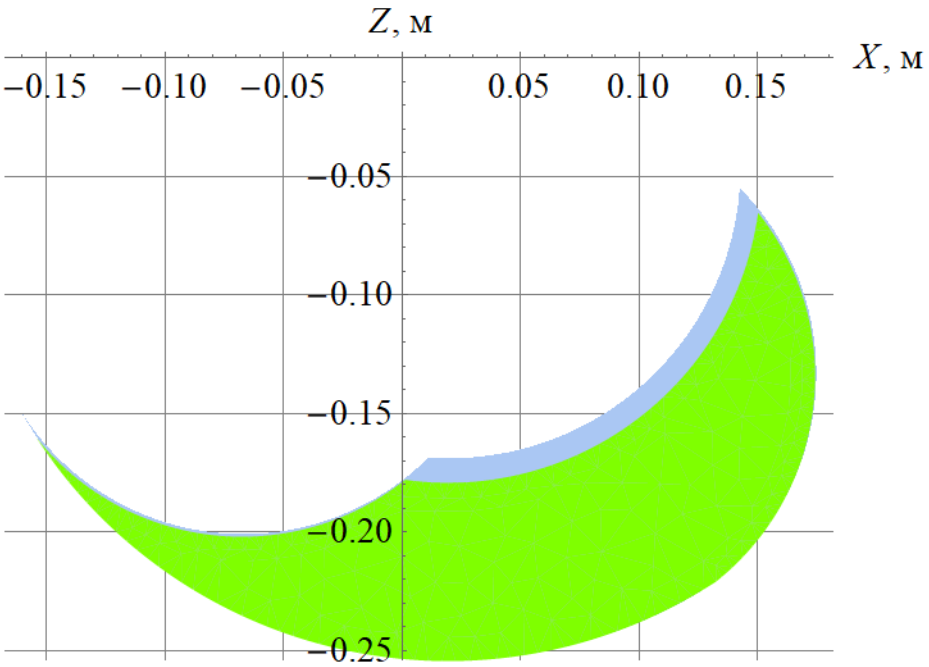


Рисунок 3.6 — Сравнение рабочих областей

3.4 Обратная кинематика

Как уже упоминалось ранее в пункте 3.1, аналитического решения для задачи обратной кинематики нет. Это значит что нужно использовать численные методы решения.

Будем искать решение с помощью метода Ньютона для систем нелинейных уравнений. В общем виде который будет выглядеть следующим образом:

Пусть дана система из n нелинейных уравнений с n неизвестными.

$$\begin{cases} f_1(x_1, \ldots, x_n) = 0 \\ f_2(x_1, \ldots, x_n) = 0 \\ \vdots \\ f_1(x_1, \ldots, x_n) = 0 \end{cases}$$

Где $f_i(x_1, \ldots, x_n) : \mathbb{R}^n \to \mathbb{R}, i = 1, \ldots, n$ – нелинейные функции, определенные и непрерывно дифференцируемые в некоторой области $G \subset \mathbb{R}^n$. Для

записи в векторном виде введем величины:

$$\overline{x} = [x_1, x_2, \dots x_n]^T$$
$$F(x) = [f_1(x), f_2(x), \dots, f_n(x)]^T = 0$$

Нужно найти такой вектор $\overline{x}^* = [x_1^*, x_2^*, \dots x_n^*]^T$, чтобы было верно равенство $F(\overline{x}^*) = 0$. Формула для нахождения решения итеративным методом выглядит следующим образом:

$$x^{(k+1)} = x^{(k)} - J^{-1}(x^{(k)}) \times F(x^{(k)})$$

Где $k = 1{,}2\dots$, а J – матрица Якоби:

$$J = \begin{bmatrix} \frac{\partial f_1(x_1)}{\partial x_1} & \cdots & \frac{\partial f_1(x_n)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n(x_1)}{\partial x_1} & \cdots & \frac{\partial f_n(x_n)}{\partial x_n} \end{bmatrix}$$

Применительно к задаче, рассматриваемой в данной работе, вектор $\overline{x}$ является вектором углов $\overline{\varphi} = [\varphi_1, \varphi_2, \varphi_3]^T$. Вектор $F(x)$ это правая часть уравнений прямой кинематики:

$$F(\overline{\varphi}) = [X_A(\overline{\varphi}), Y_A(\overline{\varphi}), Z_A(\overline{\varphi})]^T$$

В качестве критерия остановки итераций используется $\left\| F(\varphi^{(k)}) - F^* \right\| < \varepsilon$, где F^* – известное положение точки A нашего манипулятора, для которого ищутся углы звеньев.

Алгоритм поиска решения обратной задачи ниже. В нём $\varphi^{(0)}$ – начальное приближение, ε – точность вычислений, i – число итераций (перед выполнением алгоритма устанавливается равным нулю).

1. Расчёт Якобиана:

$$J = \begin{bmatrix} \frac{\partial X_A(\varphi_1)}{\partial \varphi_1} & \frac{\partial X_A(\varphi_2)}{\partial \varphi_2} & \frac{\partial X_A(\varphi_3)}{\partial \varphi_3} \\ \frac{\partial Y_A(\varphi_1)}{\partial \varphi_1} & \frac{\partial Y_A(\varphi_2)}{\partial \varphi_2} & \frac{\partial Y_A(\varphi_3)}{\partial \varphi_3} \\ \frac{\partial Z_A(\varphi_1)}{\partial \varphi_1} & \frac{\partial Z_A(\varphi_2)}{\partial \varphi_2} & \frac{\partial Z_A(\varphi_3)}{\partial \varphi_3} \end{bmatrix}$$

2. Вычисление погрешности:

$$err = F(\varphi^{(k)}) - F^*$$

3. Вычисление вектора полного шага:

$$p = J^+ \times err$$

где J^+ – псевдо-обратная матрица Якобиана.

4. Приращение вектора полного шага к ответу:

$$\varphi^{(i+1)} = \varphi^{(i)} - p$$

5. Приращение количества итераций:

$$i = i + 1$$

6. Если заданная точность не достигнута и не превышен лимит на количество итераций, возвращаемся к пункту 1.

На практике оказалось, что скорость сходимости, и сама сходимость сильно зависит от начального приближения. После реализации алгоритма было выявлено две проблемы в его работе:

1. Невозможно подобрать такие начальные приближения, при которых бы метод сходился во всем трехмерном пространстве рабочей области.
2. В областях, в которых метод сходился в большом удалении от начального приближения, требовалось более 15 итераций для нахождения решения с заданной точностью.

Вторая проблема связана с производительностью вычислителей, которые нужно производить на микрокомпьютере в режиме реального времени.

Быстрое решение обратной задачи для любой точки трехмерной рабочей области это основное требование которое ставилось в начале разработки системы, поэтому итеративное приближение из предыдущих точек не подходит в данной ситуации. В следующем пункте приведено решение возникших проблем.

3.5 Оптимизация численного решения обратной задачи

Путем проб и ошибок было решено кэшировать результаты вычислений прямой кинематики во время запуска программы. Проще говоря, подготовить набор предрассчитанных начальных приближений, которые будут использоваться от момента запуска робота до момента завершения работы. Во время решения обратной задачи нужно искать в кэш-таблице ближайшее готовое решение, принимать его за начальное условие текущей задачи, и от него досчитывать более точное решение.

В качестве первой реализации две сотни предрассчитанных решений были помещены в кэш-таблицу. Каждая запись в таблице имеет две колонки. В первой колонке помещается вектор координат $[X_A,Y_A,Z_A]$, во второй колонке вектор соответствующих им углов звеньев $[\varphi_1, \varphi_2, \varphi_3]$. В момент поиска наилучшего начального приближения для некоторого положения конечности $[X_A^*,Y_A^*,Z_A^*]$ находится такая запись кэш-таблицы, для которой евклидово расстояние между $[X_A^*,Y_A^*,Z_A^*]$ и $[X_A,Y_A,Z_A]$ будет минимально.

Простейшая реализация кэширования решений обратной кинематки:

```
def cache_inversed_kinematics():
    global cache
    cache = []
    phi1_start, phi1_end = PHI1_RANGE
    phi2_start, phi2_end = PHI2_RANGE
    phi3_start, phi3_end = PHI3_RANGE

    for phi1 in np.linspace(phi1_start, phi1_end, num=2):
        for phi2 in np.linspace(phi2_start, phi2_end, num=10):
            for phi3 in np.linspace(phi3_start, phi3_end, num=10):
                angles = [phi1, phi2, phi3]
                cache.append((direct(angles), np.array(angles)))
```

Пример реализации поиска решения в кэше:

```
def find_nearest_solution(coordinates):
    global cache
    return min(cache, key=lambda di: np.linalg.norm(
        (coordinates - di[0]),
        ord=2
    ))[1]
```

Таким образом получилось решить сразу две проблемы, описанные в пункте 3.4. Теперь на всей рабочей области обеспечена сходимость алгоритма. Поиск по кэш-таблице гораздо менее тяжеловесен чем итерации метода Ньютона, а после того как найдено «ближайшее» решение, метод Ньютона быстро досчитывает ответ в среднем за 2-3 итерации.

Таким образом реализация функции поиска решения обратной задачи выглядит так:

```
def inversed(coordinates):
    """
    SOLVING THE INVERSE KINEMATICS PROBLEM \n
    Param coordinates is [X, Y, Z] numpy vector, in meters.
    This function uses Newton's numerical method.
    """
    phis = find_nearest_solution(coordinates)
    error = np.ones((3,), dtype=np.float64)

    i = 0
    while np.linalg.norm(error, 2) > 1e-5 and i < 100:
        J = jacobian(phis)
        X = direct(phis)
        error = X - coordinates
        p = np.matmul(np.linalg.pinv(J), error)
        phis = phis - p
        i += 1

    return phis
```

Пути для дополнительной оптимизации:

1. Проследить за равномерным распределением кэшированных решений в пространстве рабочей области. В случае равномерного распределения, размер кэша можно сильно уменьшить. Это ускорит поиск по таблице.
2. Уменьшать количество кэшированных решений до тех пор пока метод Ньютона продолжает быстро сходиться. Это так же позволит сокра-

тить размер кэша и увеличить скорость поиска по нему.

ГЛАВА 4

ОРИЕНТАЦИЯ ТЕЛА РОБОТА

4.1 Кватернионы

4.2 Произвольные повороты тела в пространстве

4.3 Связь ориентации тела с конфигурацией ног

ГЛАВА 5

МОДЕЛИРОВАНИЕ ХОДЬБЫ

5.1 Траектории движения ног

5.2 Проблема выбора оптимальной траектории

ГЛАВА 6

ПРОГРАММНАЯ АРХИТЕКТУРА

6.1 Структура управления

6.2 Протокол передачи данных

Отправка команд от *Raspberry Pi* к *Arduino* и последующий прием обратной связи, производится через интерфейс *Serial*. Это самый надежный вариант при условии, что для установки соединения используется обыкновенный *USB* кабель, который уже защищен от помех и штекер которого надежно закреплен в разъеме.

Для достижения наилучшей скорости реакции микроконтроллера на команды был разработан специальный протокол передачи данных и написаны легковесные программные библиотеки. Для платформы *Arduino* код написан на языке *C++*, для микрокомпьютера *Raspberry Pi* была разработана библиотека на языке *Python*.

Библиотека для передачи данных написана для трёх платформ. Она сильно упрощает наладку управления *Arduino* с помощью *Raspberry Pi*. Основная идея программного кода в том, чтобы микрокомпьютер отправлял на микроконтроллер максимально простые команды, состоящие из массива байт, в который зашифрованы лишь целочисленный номер и вещественные аргументы. Обратная связь от микроконтроллера реализована в формате *JSON*.

Для справки: *JSON* это сокращение от *JavaScript Object Notation* – формата передачи данных. Как можно понять из названия, JSON произошел из *JavaScript*, но он доступен для использования на многих других языках, включая *Python, Ruby, PHP* и *Java*, в англоязычным странах его в основном произносят как *Jason*, то есть как имя ДжЭйсон.Легкочитаемый и компактный, *JSON* представляет собой хорошую альтернативу *XML* и требует куда меньше форматирования контента.

Объект *JSON* это набор данных в формате ключ-значение, который находится в фигурных скобках. Вот так выглядит *JSON* объект:

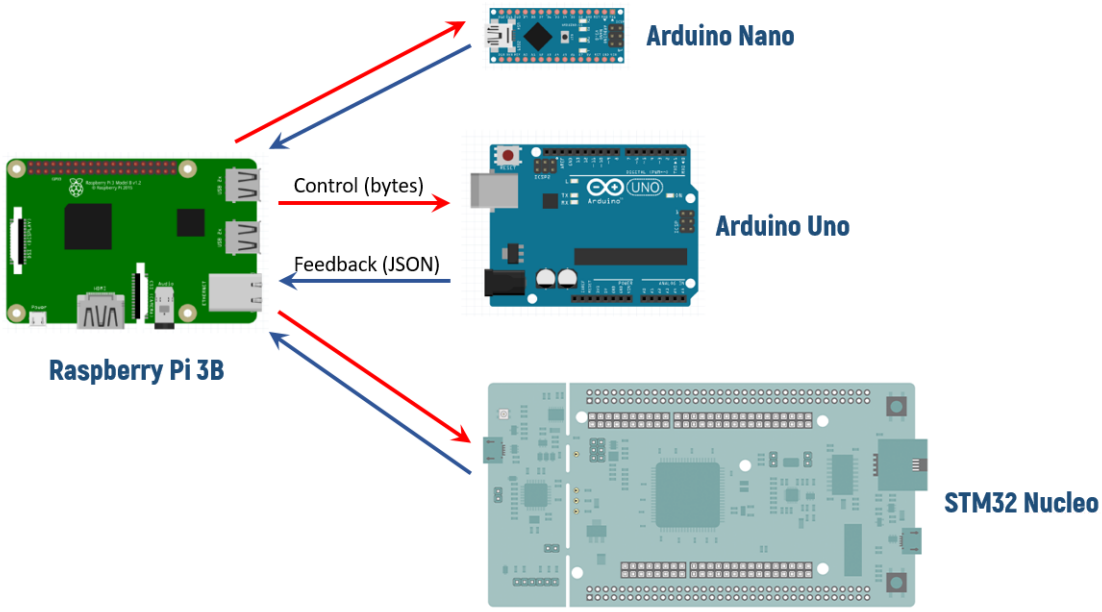


Рисунок 6.1 — Управление микроконтроллерами с помощью *Raspberry Pi*

```
{
  "first_name" : "Anton",
  "last_name" : "Kolomeytsev",
  "location" : "Moscow",
  "online" : true,
  "languages" : ["Russian", "English"]
}
```

Преимущества выбранного подхода:

- Микроконтроллер способен очень быстро расшифровать массив байт. Поэтому задержка между отправкой команды и её исполнением для человека не заметна.
- Невозможно перенести высокоуровневый функционал и принятие решений на микроконтроллер. И это хорошо.
- При таком формате общения между устройствами разработка укладывается в основы теории автоматического управления.
- Возможность подключить до четырёх микроконтроллеров к одному *Raspberry Pi*.

Протокол

Правила, по которым устанавливается соединение, следующие:

1. *Raspberry Pi* устанавливает соединение с *Arduino* по *USB*, открывая

Serial порт между устройствами.

2. *Arduino* отправляет свой уникальный идентификатор.
3. После получения идентификатора *Raspberry* может использовать его для обмена данными.

При написании кода на *Raspberry*, есть всего три простых функции:

Функция *push*: отправляет команду под номером *CODE* с аргументами *ARG*

```
clapi.device_id.push(CODE[, ARG1, ARG2, ..., ARGN])
```

Функция *pull*: блокирует выполнение программы, дожидается входящего сообщения от устройства. Возвращает *JSON*, который нам прислало устройство.

```
clapi.device_id.pull()
```

Функция *request*: отправляет команду как *push*, после чего дожидается ответа как *pull*.

```
clapi.device_id.request(CODE[, ARG1, ARG2, ..., ARGN])
```

Первая версия кода библиотеки для управления была написана в 2019 году, использовалась «в боевых условиях» и достойно показала себя на робототехнических соревнованиях *Eurobot*. С тех пор код был доработан, дополнительно протестирован и оптимизирован под новые нужды.

6.3 Кинематические расчеты

Перенос расчётов из систем математических расчетов вроде *Matlab, SciLab* или *Mathematica* на один из языков программирования это не всегда тривиальная задача. Самая сложная часть переноса – матричные вычисления, элементы математического анализа (пределы, производные, интегрирование), решение систем СЛАУ, решение систем нелинейных уравнений. При переносе кинематических расчётов в рамках данного проекта понадобилось реализовать численно следующие функции:

- Вычисление графа прямой кинематики
- Решение тригонометрической задачи о четырехзвеннике
- Вычисление Якобиана

- Решение обратной задачи кинематики

Использование библиотеки численной математики *Nupmy* на языке программирования *Python* сильно упростило задачу. Стоит отметить что *Numpy* сегодня активно используется для математических вычислений исследователями всех стран.

6.3.1 Вычисление графа прямой кинематики

Самая простая в реализации задача – нужно просто найти три координаты и вернуть вектор:

```
1  def direct(angles):
2      phi1, phi2, phi3 = angles[0], angles[1], angles[2]
3      phi4 = four_link_angle(phi3) - (pi/2)
4      X = A2 + A3 * cos(phi2) + A6 * sin(phi2) + A8 * sin(phi2 + phi4) + (A9 +
           A10) * cos(phi2 + phi4)
5      V = A1 - A3 * sin(phi2) + A6 * cos(phi2) + A8 * cos(phi2 + phi4) - (A9 +
           A10) * sin(phi2 + phi4)
6      Y = V * sin(phi1)
7      Z = V * cos(phi1)
8      return np.array([X, Y, Z])
```

Декомпозируем входные углы, подставляем их в формулы, получая X, Y, Z, и возвращаем вектор $F^* = [X, Y, Z]$.

6.3.2 Решение тригонометрической задачи о четырехзвеннике

Тоже не сложная задача, если есть готовое решение в математическом пакете *Mathematica*.

```
1  def four_link_angle(phi3):
2      d = sqrt(A5**2 + A7**2 - 2 * A5 * A7 * cos(phi3 + pi/2))
3      gamma = asin((A5 / d) * cos(phi3))
4      delta = acos((d**2 + A9**2 - A7**2) / (2 * d * A9))
5      return (pi - gamma - delta)
```

Достаточно громоздкие вычисления, хотя угол φ_3 не сильно отличается от φ_4.

6.3.3 Вычисление Якобиана

Для численного решения обратной задачи кинематики нужно в каждой итерации вычислять Якобиан. Аналитический вид производной кинематического графа представленной в работе конечности неоправданно больших размеров, что делает невозможным его перенос из математического пакета в код на *Python*. Однако используя функцию вычисляющую прямую задачу кинематики, и устанавливая шаг для приращения каждого угла мы можем дифференцировать численно следующим образом [2]:

$$\frac{\partial f}{\partial x}(a,b) \approx \frac{f(a+h_1,b)-f(a,b)}{h_1}$$
$$\frac{\partial f}{\partial y}(a,b) \approx \frac{f(a,b+h_2)-f(a,b)}{h_2}$$

Можно оценить погрешность вычисления:

$$\frac{\partial f}{\partial x}(a,b) - \frac{f(a+h_1,b)-f(a,b)}{h_1} = \frac{h_1}{2}\frac{\partial^2 f}{\partial x^2}(c_1,b)$$
$$\frac{\partial f}{\partial y}(a,b) - \frac{f(a,b+h_2)-f(a,b)}{h_2} = \frac{h_2}{2}\frac{\partial^2 f}{\partial y^2}(a,c_2)$$

Где $c_1 \subset (a, a+h_1)$, а $c_2 \subset (b, b+c_2)$. Такой результат нас вполне устраивает, поэтому реализация в коде следующая:

```
1  def jacobian(angles):
2      h = pi * 1e-4 # diff step
3      F = direct(angles)
4      eye = np.eye(3) * h # matrix with all h in diagonal elements
5      dFdPhi1 = (direct(angles + eye[0]) - F) / h
6      dFdPhi2 = (direct(angles + eye[1]) - F) / h
7      dFdPhi3 = (direct(angles + eye[2]) - F) / h
8      return np.array([dFdPhi1, dFdPhi2, dFdPhi3]).reshape((3, 3)).T
```

Для оптимизации вычисления идут сразу по строкам матрицы. Решение обратной задачи кинематики уже было рассмотрено в пункте 3.4.

6.4 Тестирование кода

При разработке программной части любого устройства существует непропадающая проблема. Разработчики не могут писать код без ошибок. Более того, в сколько-нибудь сложных программных системах при внесении каких-либо изменений слишком сложно понять, как это отразится на других частях.

Для того, что бы минимизировать поиски ошибок в коде и ускорить их обнаружение, пишутся так называемые *Unit*-тесты. Такие тесты сами по себе являются программами, но сильно более простыми, чем сама система. Эти тесты автоматически запускаются при внесении в код системы изменений и выводят предупреждения в случае неправильного поведения или логики работы системы. Автоматические тесты – это хороший тон в программировании.

В нашем случае, например, тесты запускают функции решающие задачи кинематики с разными входными данными и сверяют ответы с теми числами, которые заранее рассчитаны в *Mathematica*. Если ответы не сходятся (с нужной точностью), нам покажут предупреждение. Таким образом ошибки, которые могли быть внесены в алгоритмы, связанные с кинематическими рассчетами, будут выявлены до того как код будет запущен на физической модели робота. Что повышает безопасность работы с такой системой.

Для тестирования кода на *Raspberry Pi* используется библиотека *PyTest*. В качестве примера, нам нужно протестировать функцию вычисляющую положение конечной точки ноги (прямая задача кинематики). Функция выглядит следующим образом:

6.5 Абстракция над вычислениями

ГЛАВА 7

ЗАКЛЮЧЕНИЕ

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

[1] Learning agile and dynamic motor skills for legged robots / Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy et al. // Science Robotics. — 2019. — jan. — Vol. 4, no. 26. — P. eaau5872.

[2] Morken Knut. Numerical Algorithms and Digital Representation. — The University of Oslo (UiO), 2010.

ПРИЛОЖЕНИЕ А
КОМПЛЕКТУЮЩИЕ

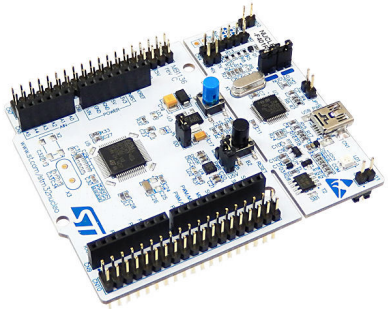
Бесколлекторный двигатель BGM5208-200-12

Количество полюсов:	12
Внутренний диаметр:	12мм
Вес:	85г
Размер:	63x24мм
Внутр. сопротивление:	15 Ом

Микрокомпьютер Raspberry Pi 3B

Процессор:	Broadcom BCM2837
Количество ядер:	4
Частота процессора:	1.2ГГц
Разрядность:	64 бита
Оперативная память:	1 ГБ
Интерфейсы:	4xUSB, Ethernet
Напряжение питания:	5В

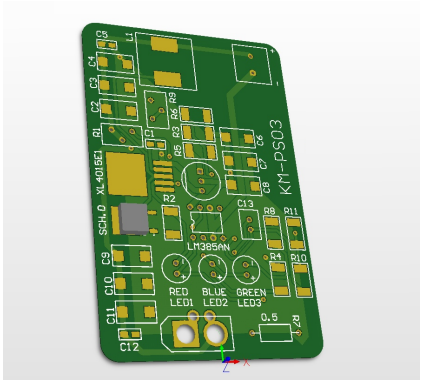
Микроконтроллер STM32 Nucleo F429ZI



Ядро:	Cortex M4
Рабочая частота:	84 МГц
Разрядность:	32 бита
Цифровых пинов:	81 шт.
Кол-во шин:	3xi2C, 4xSPI
Напряжение питания:	5В

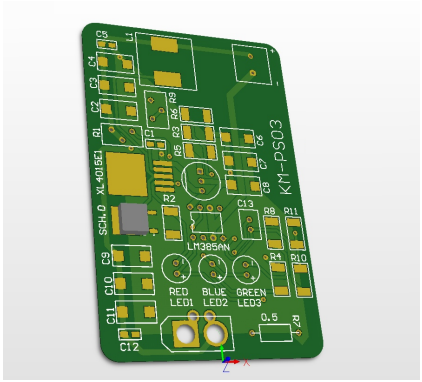
Продолжение приложения А на следующей странице...

Блок питания KMPS05



Ядро:	Cortex M4
Рабочая частота:	84 МГц
Разрядность:	32 бита
Цифровых пинов:	81 шт.
Кол-во шин:	3xi2C, 4xSPI, 3xUART
Напряжение питания:	5В

Драйвер бесколлекторных двигателей KMBD01



Ядро:	Cortex M4
Рабочая частота:	84 МГц
Разрядность:	32 бита
Цифровых пинов:	81 шт.
Кол-во шин:	3xi2C, 4xSPI, 3xUART
Напряжение питания:	5В

ПРИЛОЖЕНИЕ Б
КОД КЛАССА

```
14  class TaskPool():
15
16      def __init__(self, serial_wrapper):
17          self.serial_wrapper = serial_wrapper
18          self.running = True
19          self.main_thread = None # thread that processes incoming messages and
                tasks
20          self.tasks = list() # tasks queue
21          self.subscribers = dict() # (CODE -> LISTENER) where CODE is code of
                the Task for which the response is expected
22          self.task_lock = Lock()
23          self.inbox = dict() # for messages which nobody waited
24
25      def push_task(self, task):
26          """ Add task to perform and run main thread """
27          self.task_lock.acquire()
28          self.tasks.append(task)
29          # run thread if it is not alive
30          if not self.main_thread or not self.main_thread.isAlive():
31              self.main_thread = Thread(target=self.main_loop, daemon=False) #
                    daemon=True for force terminating
32              self.main_thread.start()
33          self.task_lock.release()
```