

Разработка мобильного четырехногого робота

Коломейцев А.А.

НИУ "МЭИ"

28 мая 2020 г.

АННОТАЦИЯ

В рамках работы рассматривается разработка шагающего четырехногого робота, с упором на проблему разработки конечностей и программирования. Спроектированы трёхмерные твердотельные чертежи системы, собрана физическая модель. Обоснован выбор конструктивных решений, выбор программной архитектуры, выбор алгоритмов. Применены лучшие практики работы с кодом, разработаны программные библиотеки для управления микроконтроллерами в составе робота.

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	9
ГЛАВА 2 МЕХАНИЧЕСКАЯ КОНСТРУКЦИЯ	10
2.1 Проектирование ног	10
2.2 Проектирование корпуса	12
2.3 Подбор комплектующих	13
2.3.1 Силовая электроника.....	13
2.3.2 Логическая электроника	14
2.4 Аккумулятор	14
2.5 Остальные детали	15
ГЛАВА 3 КИНЕМАТИКА КОНЕЧНОСТЕЙ РОБОТА	18
3.1 Общее положение	18
3.2 Общее решение четырёхзвенника	20
3.3 Прямая кинематика	22
3.4 Обратная кинематика	24
3.5 Оптимизация численного решения обратной задачи	27
ГЛАВА 4 ОРИЕНТАЦИЯ ТЕЛА РОБОТА	30
4.1 Кватернионы	30
4.2 Произвольные повороты тела в пространстве	30
4.3 Связь ориентации тела с конфигурацией ног	30

ГЛАВА 5	МОДЕЛИРОВАНИЕ ХОДЬБЫ	31
5.1	Траектории движения ног	31
5.2	Проблема выбора оптимальной траектории	31
ГЛАВА 6	ПРОГРАММНАЯ АРХИТЕКТУРА	32
6.1	Структура управления	32
6.2	Протокол передачи данных	32
6.3	Кинематические расчеты.....	35
6.3.1	Вычисление графа прямой кинематики	36
6.3.2	Решение задачи о четырехзвеннике	36
6.3.3	Вычисление Якобиана	37
6.4	Тестирование кода	38
6.5	Абстракция над вычислениями	38
ГЛАВА 7	ЗАКЛЮЧЕНИЕ	39
	БИБЛИОГРАФИЧЕСКИЙ СПИСОК.....	40

ВВЕДЕНИЕ

Шагающие роботы - это класс роботов, имитирующих движения людей и животных. Миллионы лет эволюции 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 ИИТ (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 [1] и меньшее потребление тока, чем модели ходьбы, описанные человеком вручную.

ГЛАВА 1

ШАГАЮЩИЕ РОБОТЫ

1.1 Задачи работы

В рамках работы рассматривается разработка шагающего четырехногого робота, с упором на проблему разработки конечностей и программирования.

Для выполнения работы требуется:

- Рассчитать худший (по нагрузке) статический случай для конечностей 4-х ногого робота.
- Подобрать электроприводы удовлетворяющие по крутящему моменту.
- Подобрать остальные комплектующие, входящие в состав робота и обеспечивающие его автономную работу.
- Спроектировать и собрать физическую модель робота.
- Запрограммировать управление робота.
- Изучить управляемость и механические характеристики такого 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, в которых требуется мобильность и подвижность человеческих ног, и при этом невозможно присутствие самого человека. Речь о работах вроде общего тех. осмотра помещений, исследования местности вдали от дорог и цивилизации, помощи в устранении последствий катастроф, пребывание в опасной для человека среде (под воздействием вредных газов, излучения). Открытость методик, исходных кодов и готовых моделей в открытом доступе 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 для дальнейшей работы. Хотя изначально данных было недостаточно, исходя из опыта были сделаны некоторые предположения по требуемым габаритам конечностей робота и крутящим моментам двигателей, которые в последствии оказались верны.

Далее исходя из дополненных данных о длинах и моментах нужно лишь рассчитать ограничение на массу тела робота, после чего можно продолжать работу.

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:

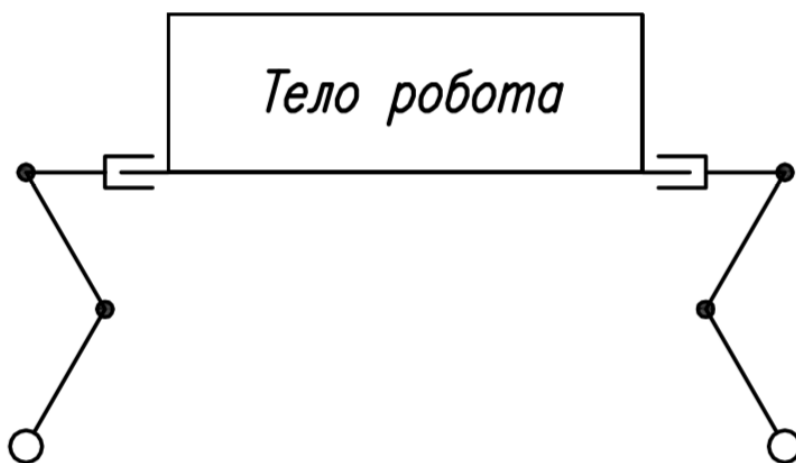


Рисунок 1.1 — Кинематика 4-х ногого робота, вид сбоку

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 .

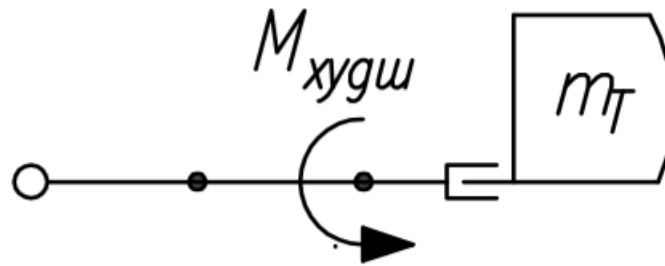


Рисунок 1.2 — Кинематика 4-х ногого робота, вид сбоку (худший случай)

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 приходиться лишь $\frac{1}{4}m$ тела робота.

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:

$$M_{\text{худш}} = \frac{1}{4}(l_1 + l_2)mg,$$

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.

Спроектированные для прототипа конечности имеют характеристики:

$$l_1 \approx (44 + 87) \times 10^{-3} = 131 \text{ мм}$$

$$l_2 \approx 134.5 \times 10^{-3} = 134.5 \text{ мм}$$

С использованием приводов с крутящим моментом 2.5Н максимально допустимый вес робота будет примерно равен 3.8 кг. Такое требование к весу, с учетом изготовления деталей из алюминия и пластика, вполне выполнимо.

1.3.2 Подбор электроприводов

Исходя из ожидаемых моментов и скоростей был выбран заводской сервопривод *DSSERVO RDS3225*.

Использовать паспортные данные двигателей для расчётов может быть рискованно, нужно измерить их реальный момент вращения. Для измерения крутящих моментов были разработаны стенды, принцип действия которых можно описать следующей схемой:

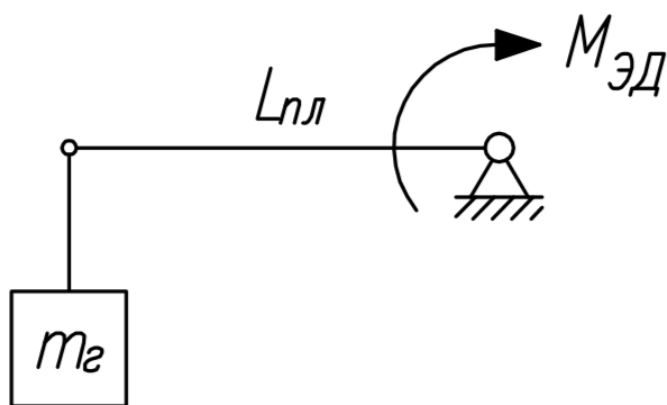


Рисунок 1.3 — Кинематика стенда для измерения крутящего момента

На изображении m_g - масса груза, которая нам известна, и которую мы можем менять. $L_{пл}$ - длина плеча, также известная нам. Из приведенных величин можно легко найти экспериментально значение момента $M_{эд}$. После проверки сервопривода на стенде, оказалось что действительная величина крутящего момента соответствует паспортным данным в пределах погрешности измерений.

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. Основным 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 алюминиевые трубки в качестве стержней, обеспечивающих жесткость.

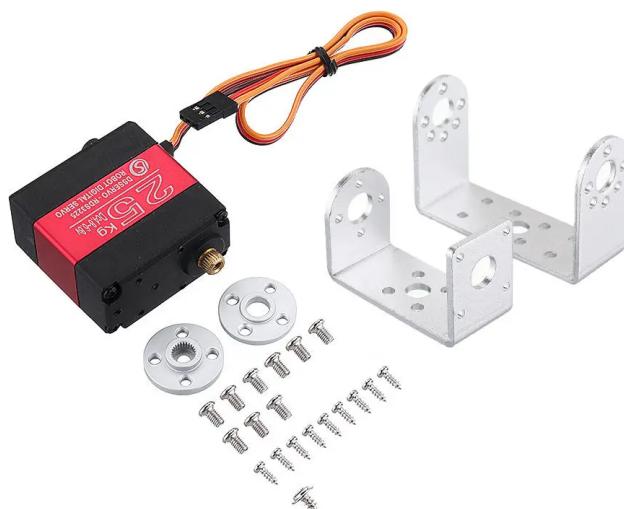


Рисунок 2.1 — Сервоприводы *DSSERVO RDS3225*, использованные в прототипе робота

Есть еще один, не менее эффективный способ снизить момент инерции ног, не уменьшая общего веса конечностей – концентрация основной массы как можно выше [2], 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 двигателя, который имеет неоптимальные габариты, не пришлось размещать в последнем узле ноги, что позволило сделать конечность максимально компактной.

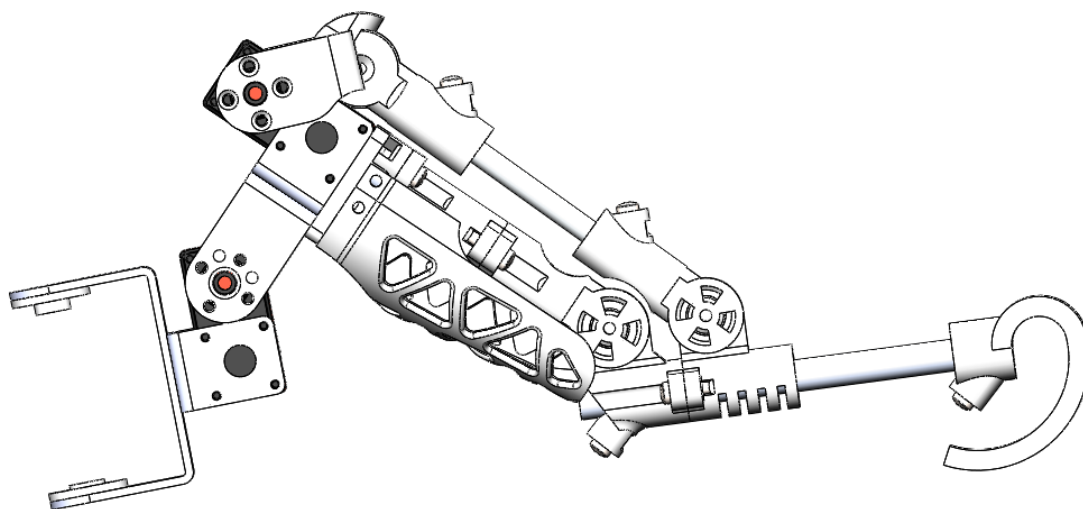
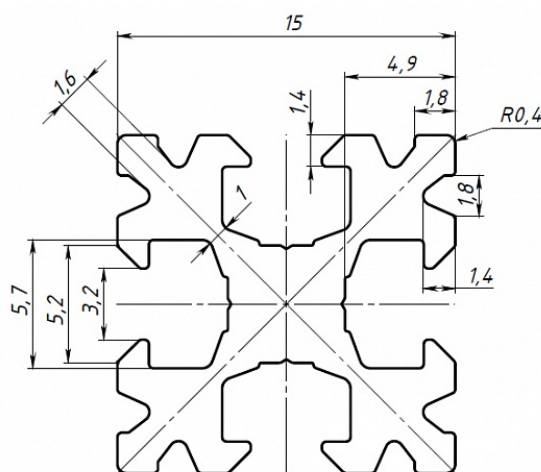


Рисунок 2.2 — Твёрдотельный чертеж конечности робота. Можно заметить что двигатели размещены у основания конечности.

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 элек-



(a) Сечение профиля



(б) Внешний вид профиля

Рисунок 2.3 — Конструкционный алюминиевый профиль

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

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 ремонтно-пригодность прототипа, а это сильно ускоряет работу с ним.

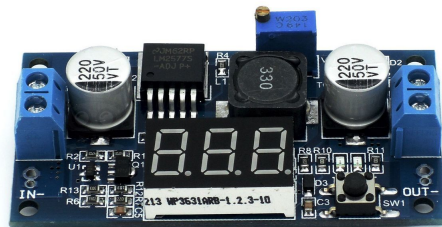
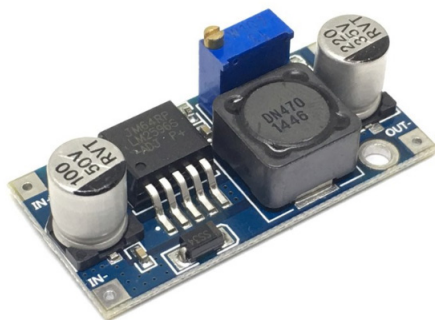
2.3 Подбор комплектующих

2.3.1 Силовая электроника

Из силовой электроники в работе присутствуют:

- Сервоприводы
- *DC-DC* преобразователи напряжения.

Про подбор сервоприводов рассказано в пункте 1.3.2. Преобразователи напряжения используются максимально компактные и простые в настройке. В основном используются импульсные преобразователи на базе микросхем *LM-XXXX* и *XL-XXXX*. Они хорошо взаимозаменяемы друг с другом. Для двух пар ног выбраны два преобразователя с максимальным током нагрузки 5 А. Для питания логической части используется преобразователь с током нагрузки 3 А.



(а) Преобразователь на базе *LM2596S*

(б) Преобразователь на базе *LM2577*

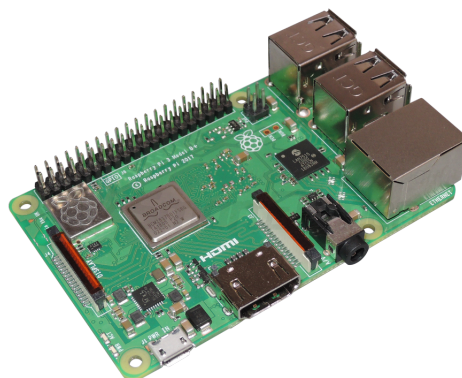
Рисунок 2.4 — Некоторая силовая электроника в составе робота

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 2.5 — 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



Рисунок 2.6 — *Li-Po* аккумулятор на 2600 мА·ч

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
- Требуется 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 последних лет) и было решено использовать их.

2.5 Остальные детали

Пластиковые детали изготавливались 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 изготовлении 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.

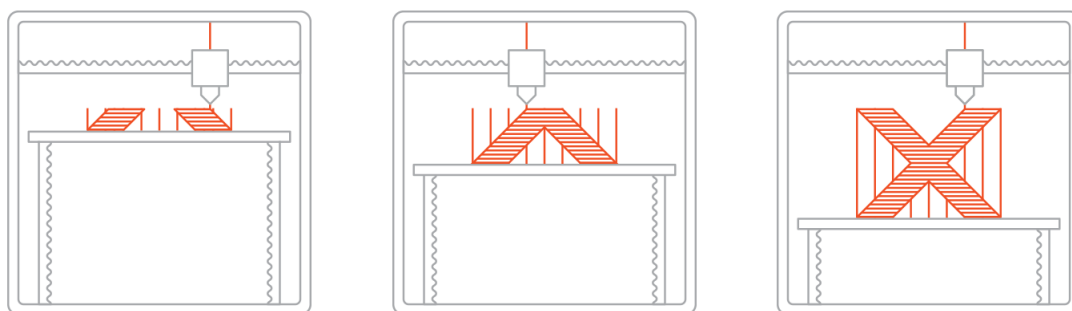


Рисунок 2.7 — Иллюстрация 3D-печати по технологии *FDM*

Из металла выполнены покупные кронштейны, стержни ног. Стержни – стандартные полые цилиндрические алюминиевые 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.

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.

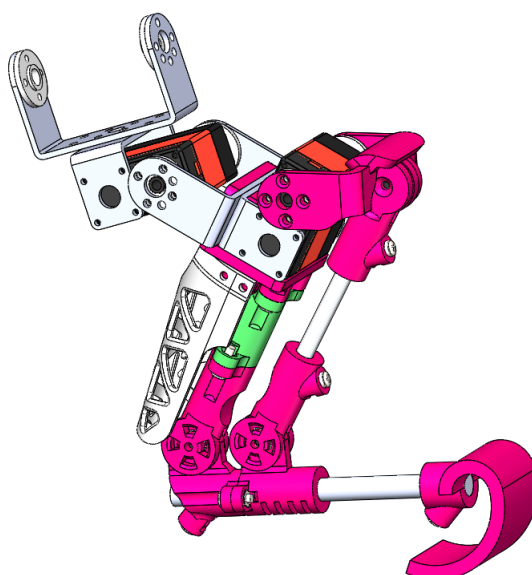


Рисунок 3.1 — Чертеж конструкции ноги

Для упрощения понимания ниже приведена трехмерная кинематическая схема конечности.

В составе конечности присутствует не прямой 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.

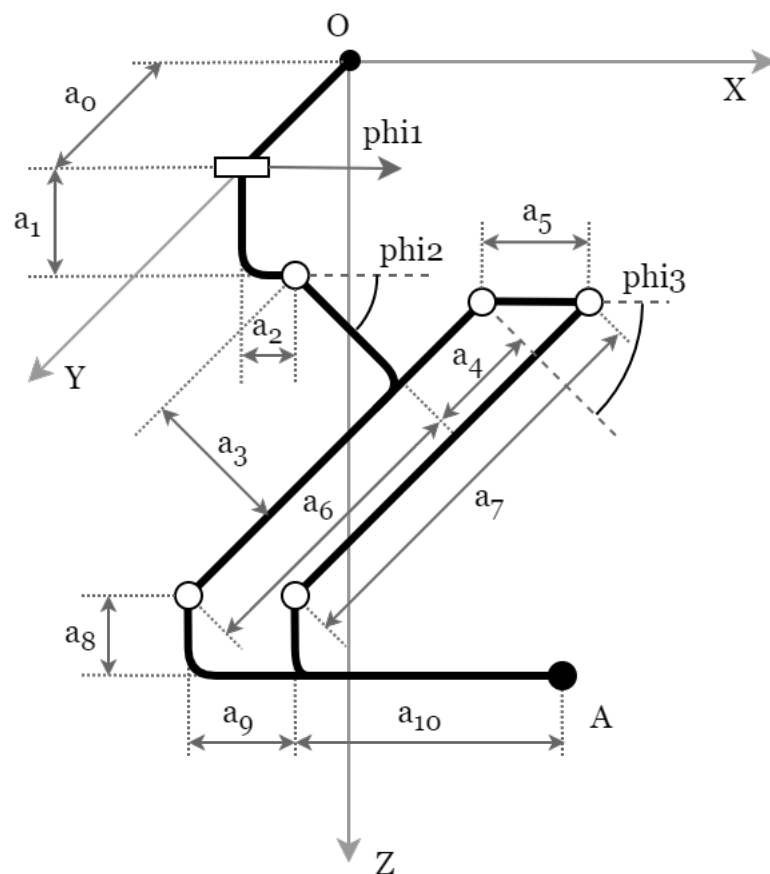


Рисунок 3.2 — Трехмерная кинематическая схема ноги робота

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

$$a_0 = 0.01_m$$

$$a_1 = 46.22 \times 10^{-3}_m$$

$$a_2 = 20 \times 10^{-3}_m$$

$$a_3 = 44 \times 10^{-3}_m$$

$$a_4 = 20 \times 10^{-3}_m$$

$$a_5 = 27 \times 10^{-3}_m$$

$$a_6 = 87 \times 10^{-3}_m$$

$$a_7 = 107 \times 10^{-3}_m$$

$$a_8 = 12.62 \times 10^{-3}_m$$

$$a_9 = 24.5 \times 10^{-3}_m$$

$$a_{10} = 110 \times 10^{-3}_m$$

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

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

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:

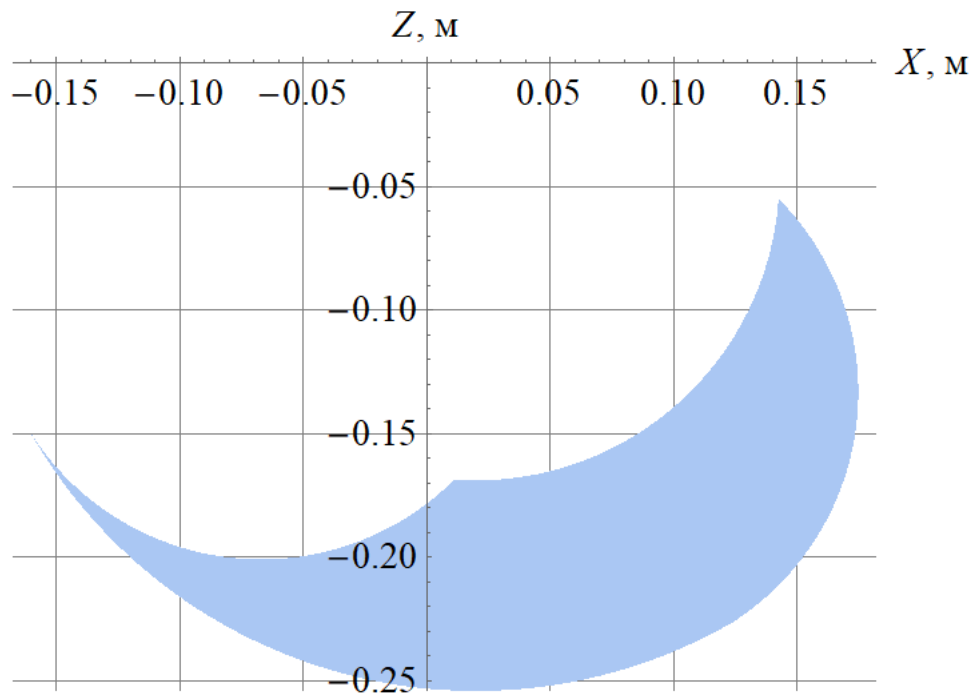


Рисунок 3.3 — Сечение рабочей области конечности робота

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.

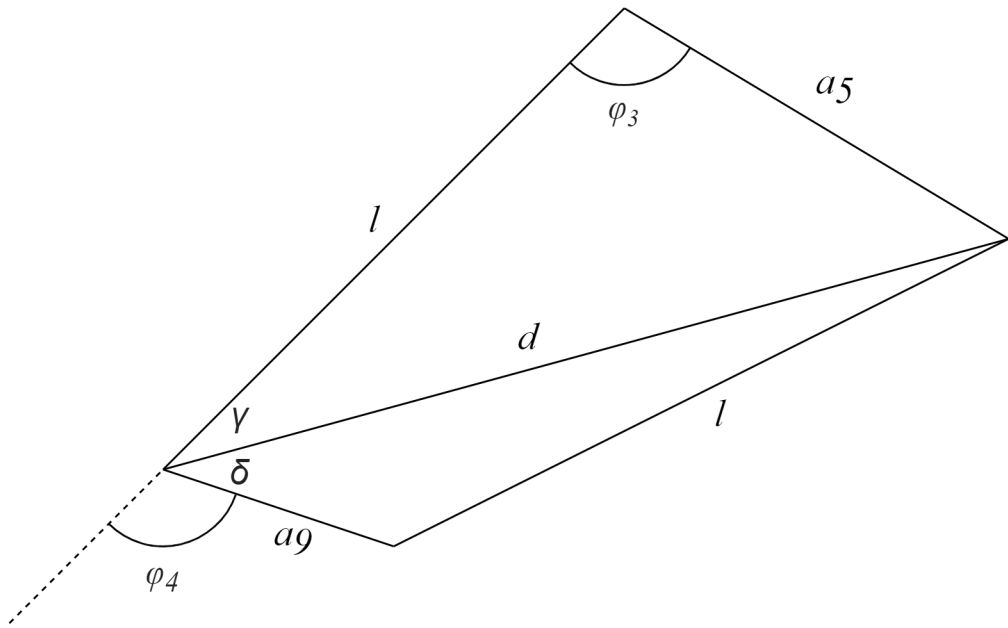


Рисунок 3.4 — Четырехзвенник

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

$$d = \sqrt{a_5^2 + l^2 - 2a_5l \cos(\varphi_3)} \quad (3.1)$$

С помощью уравнения 3.1 выразим углы:

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

$$\delta = \arccos \left(\frac{d^2 + a_9^2 - l^2}{2a_9l} \right) \quad (3.3)$$

И искомый угол φ_4 будет находиться следующим образом:

$$\varphi_4 = \pi - \gamma - \delta \quad (3.4)$$

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

$$\begin{aligned} \varphi_4 = \pi - \arcsin \left(\frac{a_5}{\sqrt{a_5^2 + l^2 - 2a_5l \cos(\varphi_3)}} \cos \varphi_3 \right) - \\ - \arccos \left(\frac{a_5^2 + l^2 - 2a_5l \cos(\varphi_3) + a_9^2 - l^2}{2a_9l} \right) \end{aligned} \quad (3.5)$$

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 :

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

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

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 :

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

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

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

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.

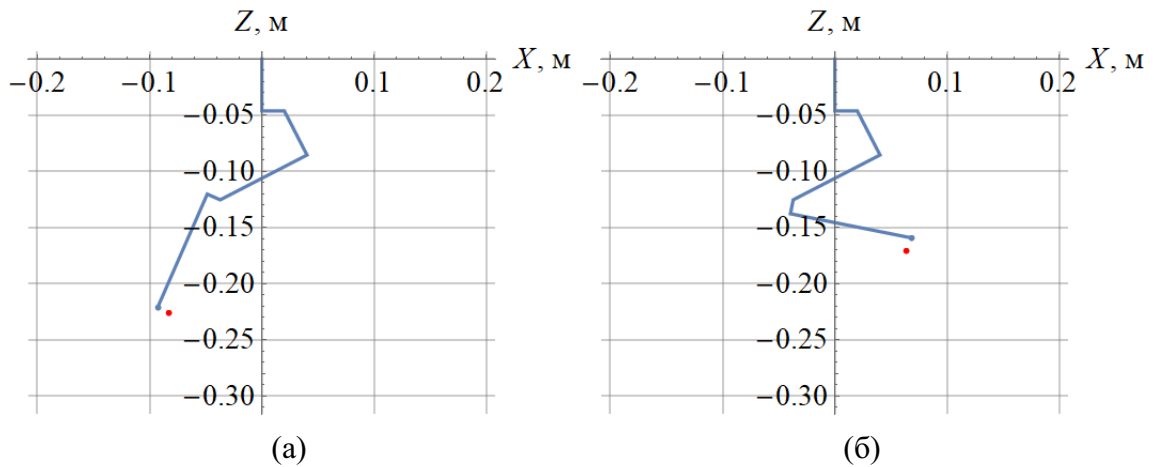


Рисунок 3.5 — (а) Третье звено полностью «разогнуто»; (б) Третье звено полностью «согнуто».

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 $a_5 > a_9$.

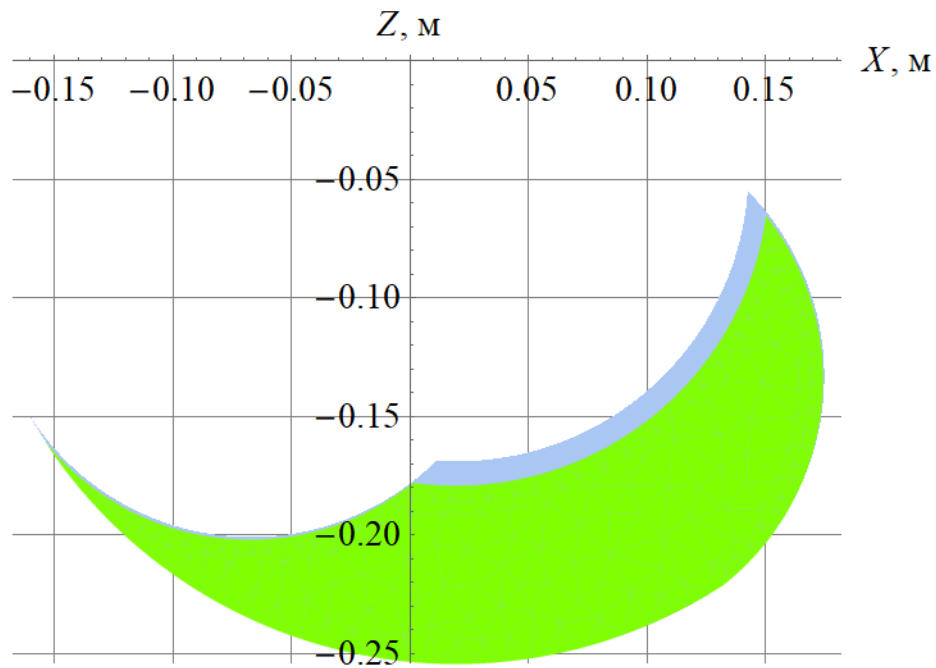


Рисунок 3.6 — Сравнение рабочих областей

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.

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

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

записи в векторном виде введем величины:

$$\bar{x} = [x_1, x_2, \dots, x_n]^T$$

$$F(x) = [f_1(x), f_2(x), \dots, f_n(x)]^T = 0$$

Нужно найти такой вектор $\bar{x}^* = [x_1^*, x_2^*, \dots, x_n^*]^T$, чтобы было верно равенство $F(\bar{x}^*) = 0$. 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:

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

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

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

В качестве критерия остановки итераций используется $\|F(\varphi^{(k)}) - F^*\| < \varepsilon$, 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 сходил в во всем трехмерном пространстве рабочей области.
2. В областях, в которых метод сходил в большом удалении от начального приближения, требовалось более 15 итераций для нахождения решения с заданной точностью.

Вторая проблема связана с производительностью вычислений, 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 предрасчитанных 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 предрасчитанных 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 кинематики:

```
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-

титель 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

ПРОГРАММНАЯ АРХИТЕКТУРА

6.1 Структура управления

Высокоуровневая логика запрограммирована на микрокомпьютере *Raspberry Pi*. Тот в свою очередь отправляет последовательность команд на платформу *Arduino*. Последняя управляет сервоприводами через специальный драйвер и отправляет информацию о своем состоянии в виде обратной связи обратно на микрокомпьютер.

FIXME НУЖНА ИЛЛЮСТРАЦИЯ

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

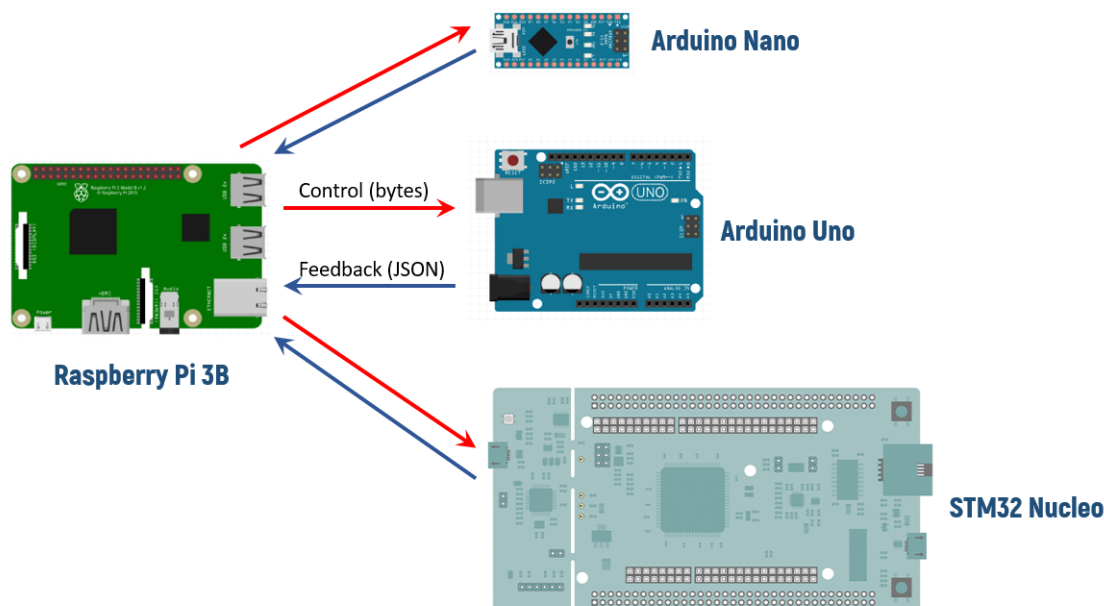


Рисунок 6.1 — Управление микроконтроллерами с помощью *Raspberry Pi*

JavaScript, но он доступен для использования на многих других языках, включая *Python*, *Ruby*, *PHP* и *Java*, в англоязычных 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:

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

Массив байт, отправляемый на микроконтроллер, всегда имеет одну и ту же структуру, которую легко расшифровать. Каждая команда имеет свой идентификатор (число от 0 до 255), а также может не иметь вовсе, либо иметь ограниченное числом 256^4 количество аргументов.

При отправке инструкции на микроконтроллер первым байтом всегда идет номер (идентификатор) команды. Вторым байтом идет число – количество аргументов. Если у команды нет аргументов, байт нулевой. Пример

команды без аргументов с идентификатором 207:

идентификатор команды

11001111

00000000

количество аргументов

К каждому идентификатору можно привязать выполнение роботом какого-либо действия. Например, включение или выключение компрессора, и т.д. Но бывает, что нужно уточнить параметры выполнения команды какими-то числовыми аргументами. В таком случае второй байт будет ненулевым и он будет указывать на количество идущих после второго байта блоков длиной в 4 байта. В каждом таком четырёхбайтном блоке хранится вещественное число. Пример команды с идентификатором 209 с одним вещественным аргументом:

идентификатор команды вещественный аргумент

11010001 00000001 10110001 11100010 00110100 00100100

количество аргументов

Таким образом можно реализовать команду, которой передаются, например, требуемые скорости вращения колес, или параметры чувствительности датчиков, и т.д.

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. Это заставляет продумывать структуру системы.
- 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 *Numpy* 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:

```
def direct(angles):
    phi1, phi2, phi3 = angles[0], angles[1], angles[2]
    phi4 = four_link_angle(phi3) - (pi/2)
    X = A2 + A3 * cos(phi2) + A6 * sin(phi2) + A8 * sin(phi2 + phi4) + (A9 +
        A10) * cos(phi2 + phi4)
    V = A1 - A3 * sin(phi2) + A6 * cos(phi2) + A8 * cos(phi2 + phi4) - (A9 +
        A10) * sin(phi2 + phi4)
    Y = V * sin(phi1)
    Z = V * cos(phi1)
    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 задачи о четырехзвеннике

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

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

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

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 $c_2 \subset (b, b + h_2)$. Такой результат нас вполне устраивает, поэтому реализация в коде следующая:

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

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] Actuator design for high force proprioceptive control in fast legged locomotion / Sangok Seok, Albert Wang, David Otten, Sangbae Kim // 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. — IEEE, 2012. — oct.
- [3] Morken Knut. Numerical Algorithms and Digital Representation. — The University of Oslo (UiO), 2010.