Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материала и транспорта
Высшая школа автоматизации и робототехники

Курсовая работа

Дисциплина: Объектно-ориентированное программирование

Тема: Разработка моноколеса

Выполнили студенты гр. 3331506/20101 Соколов М.Д.

Миронов В.В.

Преподаватель Ананьевский М. С.

Санкт-Петербург

2025

Оглавление

1. Введение

1.1. Актуальность и значимость

Современная робототехника и мехатроника активно развиваются в направлении создания все более сложных и автономных систем, способных эффективно взаимодействовать с окружающей средой. Одной из фундаментальных задач в этой области является разработка и исследование нетрадиционных методов передвижения, которые могли бы обеспечить повышенную маневренность, проходимость или энергоэффективность по сравнению с классическими колесными или гусеничными платформами. Системы с динамической стабилизацией, такие как моноколеса, балансирующие роботы и другие одноопорные или малоопорные конструкции, представляют особый интерес, поскольку их управление требует сложных алгоритмов и глубокого понимания динамических процессов.

Разработка моноколеса, использующего для движения радиально выдвигаемые штифты, является актуальной исследовательской задачей. Такой принцип движения потенциально может предложить уникальные преимущества на определенных типах поверхностей или при выполнении специфических маневров. Исследование и реализация алгоритмов управления для подобных систем вносят вклад в общую базу знаний в области робототехники, теории управления и разработки интеллектуальных мехатронных систем. Успешная реализация такого проекта демонстрирует возможность создания работоспособных конструкций на основе нетривиальных кинематических схем и способствует поиску новых инженерных решений.

1.2. Область применения

Несмотря на то, что представленная разработка на данном этапе носит преимущественно исследовательский и экспериментальный характер, потенциальные области применения для систем, основанных на схожих принципах, могут быть достаточно широки:

- **Исследовательские и образовательные платформы:** Моноколесо с активными штифтами может служить отличной платформой для изучения сложной динамики, тестирования алгоритмов управления, обучения студентов основам робототехники, теории автоматического управления и программирования встраиваемых систем.
- **Разработка новых типов мобильных роботов:** В перспективе, при дальнейшей доработке и решении проблем стабилизации и энергоэффективности, подобные системы могут найти применение в создании специализированных роботов для инспекции труднодоступных мест, развлекательных или художественных инсталляций.
- **Изучение альтернативных способов локомоции:** Данный проект способствует исследованию и пониманию преимуществ и недостатков различных методов передвижения, что может привести к созданию гибридных систем или совершенно новых концепций мобильности.
- **Прототипирование и быстрое тестирование идей:** Использование доступных компонентов, таких как микроконтроллеры Arduino и датчики MPU6050, позволяет быстро прототипировать и проверять сложные концепции управления без значительных временных и финансовых затрат.

1.3. Цели и задачи работы

Основной целью данной работы является разработка и программная реализация системы управления для экспериментального прототипа моноколеса, движение которого осуществляется за счет последовательной активации радиально расположенных и выдвигаемых штифтов.

Для достижения поставленной цели были определены следующие ключевые задачи:

1. **Разработать программный модуль для считывания и первичной обработки данных** с инерциального измерительного модуля (IMU MPU6050), включающий получение данных с акселерометра и гироскопа.

2. **Реализовать алгоритм калибровки датчиков IMU** для компенсации смещения нуля гироскопа и определения поправки для системы координат колеса.
3. **Применить фильтр Калмана** для слияния данных акселерометра и гироскопа с целью получения точной и стабильной оценки текущего угла ориентации моноколеса в пространстве и его угловой скорости.
4. **Разработать и обосновать алгоритм определения оптимального угла активации штифта** для создания поступательного движения, базируясь на геометрических параметрах колеса и выбранной стратегии толчка ("загребание").
5. **Реализовать логику последовательной активации актуаторов**, обеспечивающую продвижение колеса в заданном направлении.
6. **Внедрить механизм коррекции движения при откате колеса назад**, предусматривающий активацию соответствующего актуатора для восстановления движения вперед.
7. **Обеспечить управление временем работы актуаторов** для предотвращения их перегрева и оптимизации энергопотребления.
8. **Создать систему отладочного вывода** для мониторинга ключевых параметров системы и анализа ее поведения в реальном времени.

2. Описание Системы и Принцип Действия

2.1. Общее описание моноколеса

Прототип моноколеса, для управления которым разработано данное программное обеспечение, представляет собой конструкцию, состоящую из одного колеса, по ободу которого на равных угловых расстояниях расположены четырнадцать (14) независимых актуаторов. Каждый актуатор способен радиально выдвигать небольшой штифт за пределы основной поверхности катания колеса. Диаметр колеса составляет 120 мм, а каждый штифт может выдвигаться на 10 мм.

В центральной части колеса или на неподвижной относительно него платформе (предполагается, что электроника не вращается вместе с колесом, а отслеживает его вращение) размещается управляющая электроника, включающая микроконтроллер (например, на базе Arduino) и инерциальный измерительный модуль (IMU MPU6050). IMU используется для определения текущего угла наклона (ориентации) колеса и его угловой скорости. Микроконтроллер обрабатывает данные с IMU и, в соответствии с заложенным алгоритмом, подает управляющие сигналы на актуаторы.

Предполагается, что движение моноколеса осуществляется не за счет вращения оси неким двигателем, а исключительно за счет контролируемого и последовательного выдвижения штифтов, которые, упираясь в поверхность, создают необходимый для движения импульс.

2.2. Принцип движения

Основной принцип движения данного моноколеса основан на создании асимметричного взаимодействия с опорной поверхностью посредством выдвигаемых штифтов. Когда штифт выдвигается и контактирует с поверхностью, возникает сила реакции опоры. Если эта сила приложена не строго вертикально под центром масс системы или если она создает момент силы относительно точки контакта обода колеса с поверхностью (или оси вращения), возможно поступательное движение.

В рамках данной работы была выбрана и реализована стратегия движения, условно названная "загребание". Суть этой стратегии заключается в следующем:

1. Система отслеживает текущее угловое положение колеса.
2. Выбирается актуатор, который в данный момент находится позади вертикальной оси, проходящей через центр колеса (т.е. в задней части пятна контакта или непосредственно перед ним, если смотреть по направлению предполагаемого движения).
3. Этот актуатор выдвигает штифт. Поскольку штифт направлен радиально от центра колеса, а его точка активации находится позади нижней точки колеса (Bottom Dead Center - BDC), при контакте с поверхностью штифт будет ориентирован несколько назад и вниз.
4. Сила реакции опоры на этот штифт будет направлена, соответственно, вперед и вверх, к центру колеса.
5. Горизонтальная компонента этой силы реакции, направленная вперед, толкает колесо, придавая ему поступательное ускорение и создавая крутящий момент, способствующий его вращению в нужном направлении.
6. После короткого импульса штифт втягивается, и система готовится активировать следующий актуатор в последовательности, который подойдет к оптимальной для "загребания" позиции.
7. Последовательная активация актуаторов, находящихся в нужной фазе вращения колеса, должна приводить к непрерывному или квазинепрерывному движению. Ключевым моментом является точное определение угла, под которым активация штифта даст максимальную полезную компоненту силы реакции для движения вперед. В программе этот угол рассчитывается на основе геометрии колеса и штифта и составляет приблизительно 329° (если 0° – это самая нижняя точка колеса, а отсчет идет по часовой стрелке). Это соответствует положению штифта примерно на 31° позади вертикали.

3. Кинематическая Схема и Анализ

3.1. Описание кинематической модели

Кинематическая модель моноколеса, рассматриваемая в данной работе, определяется следующими ключевыми геометрическими параметрами и конструктивными особенностями:

- **Диаметр колеса (D):** 120 мм.
- **Длина выдвижения штифта (L):** 10 мм. Этот параметр обозначает максимальное расстояние, на которое кончик штифта выступает за пределы обода колеса при полной активации актуатора.
- **Количество актуаторов (N):** 14. Актуаторы равномерно распределены по ободу колеса.
- **Угловое расстояние между актуаторами ($\alpha_{actuator}$):** Рассчитывается как 360° / N = 360° / 14 ≈ 25.714°. Это означает, что каждый следующий актуатор смещен относительно предыдущего на данный угол.

Штифты выдвигаются строго радиально от центра колеса. Для анализа движения важно определить эффективный радиус до кончика выдвинутого штифта.

Эффективный радиус до кончика штифта (R_{eff}): Это расстояние от центра колеса до кончика полностью выдвинутого штифта. Рассчитывается как R + L = 60 мм + 10 мм = 70 мм.

Система координат:

Для анализа и управления вводится система координат, связанная с колесом. Примем, что глобальный угол ориентации колеса φ_{wheel} равен 0°, когда условная "нулевая метка" на колесе (например, соответствующая первому актуатору, индекс 0) находится в самой нижней точке (BDC - Bottom Dead Center). Увеличение угла φ_{wheel} соответствует вращению колеса по часовой стрелке (CW), если смотреть на него с определенной стороны.

3.2. Расчет оптимального угла активации штифта

Целью расчета оптимального угла является определение такого положения штифта относительно вертикали, при котором его контакт с опорной поверхностью и последующий толчок (или "загребание") обеспечит максимальную горизонтальную составляющую силы реакции опоры, направленную в сторону движения.

В данной работе реализуется стратегия "загребание". Это означает, что мы активируем штифт, который находится *позади* вертикальной оси колеса (BDC). Штифт, выдвигаясь радиально, упирается в поверхность, будучи направленным несколько назад и вниз. Сила реакции опоры $F_{reaction}$ на кончик штифта будет направлена вдоль оси штифта, то есть радиально к центру колеса.

Рассмотрим геометрию контакта:

- Пусть θ – это угол, который образует ось выдвинутого штифта с вертикалью, проходящей через центр колеса. Положительные значения θ будем отсчитывать по часовой стрелке от вертикали вниз (BDC).
- Чтобы штифт касался земли, когда обод колеса также находится на земле (или очень близко к ней), центр колеса должен находиться на высоте R от земли.
- Кончик выдвинутого штифта будет касаться земли, если выполняется условие:

  $R = R_{eff} * \cos(\theta_{contact})$

  где θ_contact – это предельный угол от вертикали (в любую сторону), при котором кончик выдвинутого штифта может касаться земли, если сам обод также касается земли.

  Из этого уравнения получаем:

$$\cos(\theta_{contact}) = R / R_{eff}$$

$$\cos(\theta_{contact}) = 60 \text{ мм} / 70 \text{ мм} \approx 0.8571428$$

Вычисляем $\theta_{contact}$:

$$\theta_{contact} = \text{acos}(0.8571428)$$

$$\theta_{contact} \approx 0.54105 \text{ радиан}$$

Переводим в градусы:

$$\theta_{contact} \approx 0.54105 * (180.0 / \pi) \approx 31.002°$$

Этот угол $\theta_{contact} \approx 31.0°$ представляет собой максимальное угловое смещение от вертикали (как вперед, так и назад), при котором выдвинутый штифт еще может коснуться земли, если обод колеса также касается ее.

Оптимизация угла для стратегии "загребание":

При стратегии "загребание" мы хотим активировать штифт, находящийся позади BDC. Сила реакции опоры $F_{reaction}$ направлена вдоль штифта к центру колеса. Эту силу можно разложить на вертикальную $F_{vertical}$ и горизонтальную $F_{horizontal}$ составляющие.

Если θ – это угол штифта относительно вертикали (BDC), и мы считаем $\theta < 0$ для штифтов позади BDC (т.е. против часовой стрелки от BDC), то:

$$F_{horizontal} = F_{reaction} * \sin(|\theta|)$$

(если θ отсчитывается от вертикали в обе стороны как положительное) Или, если θ отсчитывается по часовой стрелке от BDC, и мы рассматриваем штифт на угле $(360° - |\theta_{BDC}|)$:

$$F_{horizontal} = F_{reaction} * \sin(\alpha_{push})$$

где α_{push} — это угол между направлением штифта и *горизонталью*.

Для максимальной горизонтальной силы $F_{horizontal}$ (при заданной $F_{reaction}$) нам нужно максимизировать $\sin(|\theta|)$ или $\sin(\alpha_{push})$.

Максимальное значение $|\theta|$ для контакта с землей, когда штифт находится позади BDC, как раз и равно $\theta_{contact} \approx 31.0°$. При этом угле штифт будет максимально отклонен назад, все еще касаясь земли. Это и будет наш оптимальный угол для "загребания", так как он обеспечивает наибольший "рычаг" для горизонтальной составляющей силы при радиальном упоре.

Таким образом, оптимальный угол активации для штифта, если считать от BDC против часовой стрелки, составляет $-\theta_{contact} \approx -31.0°$. В принятой нами системе координат, где 0° – это BDC и углы растут по часовой стрелке, этот оптимальный угол для активации штифта будет:

$$OPTIMAL\_GLOBAL\_ANGLE = 360° - \theta_{contact}$$

$$OPTIMAL\_GLOBAL\_ANGLE = 360° - 31.002° \approx 328.998°$$

В программе это значение используется как OPTIMAL_PUSH_ANGLE_FROM_BOTTOM_DEG и установлено на ~329.0°.

Вывод:

Анализ кинематической схемы моноколеса с учетом его геометрических параметров (диаметр 120 мм, вылет штифта 10 мм) и выбранной стратегии движения "загребание" показывает, что оптимальным является активация штифта, когда он находится под углом приблизительно 31.0° позади вертикальной оси колеса. В глобальной системе координат колеса (где 0° – низ, углы по CW) это соответствует угловому положению актуатора около 329.0°. Именно этот угол используется в алгоритме управления для принятия решения об активации актуаторов.

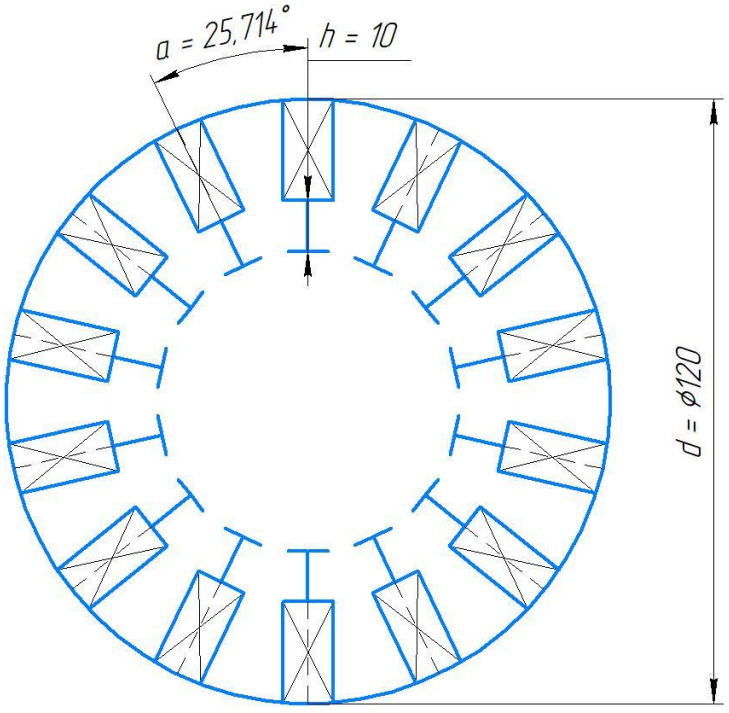

Рисунок 1 – Кинематическая схема моноколеса

4. Электрическая Схема и Компоненты

Электрическая схема системы управления моноколесом разработана для обеспечения сбора данных о положении колеса, их обработки и последующего управления четырнадцатью актуаторами. Ключевыми компонентами схемы являются микроконтроллер, инерциальный измерительный модуль, силовые ключи для управления актуаторами и система питания.

4.1. Микроконтроллер

Центральным управляющим устройством системы является микроконтроллер Arduino Nano. Этот выбор обусловлен его компактными размерами, достаточным количеством цифровых и аналоговых выводов, простотой программирования и широкой доступностью. Arduino Nano выполняет следующие основные функции:

- Инициализация и опрос датчиков инерциального измерительного модуля (IMU).
- Обработка полученных данных с IMU, включая фильтрацию и расчет параметров ориентации колеса.
- Реализация основного алгоритма управления, определяющего момент и последовательность активации актуаторов.
- Формирование управляющих сигналов для силовых ключей, коммутирующих актуаторы.
- Обеспечение отладочного вывода через последовательный порт.

4.2. Инерциальный измерительный модуль (IMU)

Для определения ориентации и угловой скорости моноколеса используется инерциальный измерительный модуль MPU6050. Этот модуль объединяет в себе трехосевой гироскоп и трехосевой акселерометр, а также встроенный цифровой процессор обработки движения (DMP), хотя в данной реализации DMP не используется напрямую, а данные считываются и обрабатываются микроконтроллером.

MPU6050 подключается к Arduino Nano по интерфейсу I²C (Inter-Integrated Circuit), используя стандартные выводы A4 (SDA) и A5 (SCL).

- **Акселерометр** используется для определения вектора гравитации, что позволяет вычислить угол наклона колеса в статическом или медленно движущемся состоянии.
- **Гироскоп** измеряет угловую скорость вращения колеса вокруг его осей. В данной системе критически важна угловая скорость вокруг оси, перпендикулярной плоскости колеса, для определения скорости и направления вращения.

Данные с этих двух сенсоров объединяются с помощью фильтра Калмана для получения робастной и точной оценки угла ориентации и угловой скорости колеса.

4.3. Система актуаторов и их управление

В качестве актуаторов, обеспечивающих движение моноколеса, используются четырнадцать (14) соленоидов (втягивающих электромагнитов) модели JF-0520B. Эти соленоиды при подаче на них напряжения втягивают сердечник, который механически связан со штифтом, выдвигая его за пределы обода колеса.

Управление каждым соленоидом осуществляется индивидуально с помощью N-канальных MOSFET-транзисторов IRFZ24. Для управления N-канальным MOSFET в режиме ключа его затвор (Gate) подключается к цифровому выводу Arduino Nano через токоограничивающий резистор (например, 100-220 Ом), а исток (Source) подключается к общей земле ("минусу") системы. Сток (Drain) транзистора подключается к одному из выводов соленоида, а другой вывод соленоида – к положительной шине питания актуаторов (от 6S Li-ion аккумулятора).

Для защиты микроконтроллера и транзистора от ЭДС самоиндукции, возникающей в катушке соленоида при его выключении, параллельно каждому соленоиду (между его выводами) установлен обратный диод.

Управляющие сигналы для затворов MOSFET-транзисторов подаются со следующих выводов Arduino Nano:

- Цифровые выводы: D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13 (12 актуаторов).
- Аналоговые выводы, используемые как цифровые: A0, A1 (2 актуатора). Итого 14 управляющих сигналов для 14 актуаторов.

4.4. Система питания

Питание всей системы осуществляется от 6S Li-ion аккумулятора. Такой аккумулятор обеспечивает номинальное напряжение около 22.2В (3.7В на банку * 6 банок) при полном заряде до 25.2В.

- **Питание актуаторов (соленоидов JF-0520B):** Соленоиды, вероятно, рассчитаны на напряжение 24В. Питание от 6S Li-ion аккумулятора подходит напрямую.
- **Питание Arduino Nano:** Микроконтроллер Arduino Nano требует стабилизированного напряжения 5В. Поскольку напряжение 6S Li-ion аккумулятора значительно выше, для питания Arduino Nano используется линейный стабилизатор напряжения на 5В. Вход стабилизатора подключается к выходу 6S Li-ion аккумулятора, а выход стабилизатора – к выводу "5V".
- **Питание MPU6050:** Модуль MPU6050 обычно питается напряжением 3.3В.

Общая земля: Все компоненты системы (Arduino Nano, MPU6050, истоки MOSFET-транзисторов, "минус" аккумулятора, "минус" соленоидов через транзисторы) должны быть соединены с общей землей (GND) для корректной работы схемы.

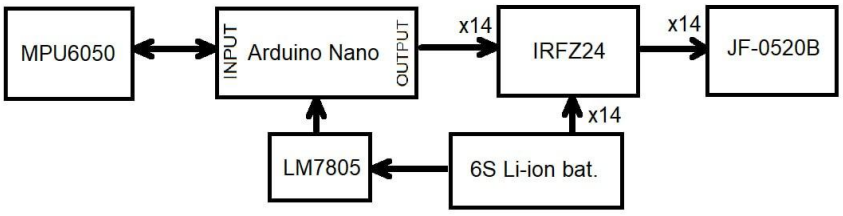

Рисунок 2 – Структурная электрическая схема моноколеса

5. Программная Реализация и Алгоритмы Управления

Программное обеспечение для управления моноколесом разработано в среде Arduino IDE на языке C++. Оно реализует комплекс алгоритмов для считывания и обработки сенсорных данных, определения состояния колеса и управления актуаторами для достижения движения.

5.1. Получение и обработка данных с IMU

Основой для определения ориентации моноколеса служат данные с инерциального измерительного модуля MPU6050.

5.1.1. Считывание "сырых" данных акселерометра и гироскопа

Данные с MPU6050 считываются по протоколу I2C. Функция read_imu_data_raw() запрашивает у датчика значения ускорений по осям X, Y, Z и угловых скоростей вокруг этих осей. Для данной задачи используются ускорения по осям Y и Z (для расчета угла наклона в плоскости колеса) и угловая скорость вокруг оси X (предполагается, что ось X гироскопа совпадает с осью вращения колеса). (Реализация считывания представлена в приложении 1).

5.1.2. Калибровка смещения гироскопа

Гироскопы подвержены дрейфу нуля, то есть могут показывать ненулевую угловую скорость даже в состоянии покоя. Для компенсации этого эффекта при инициализации системы (setup_mpu6050()) производится калибровка: в течение некоторого времени (определяемого GYRO_CALIBRATION_SAMPLES) считываются показания гироскопа, и вычисляется среднее значение смещения. Это смещение (mpu_gyro_x_offset_raw) затем вычитается из последующих измерений угловой скорости. (Реализация калибровки представлена в приложении 1).

5.1.3. Фильтр Калмана для слияния данных

Показания акселерометра точны в статике и на малых скоростях, но подвержены влиянию линейных ускорений. Показания гироскопа точны для измерения быстрых изменений угла, но интегрирование его показаний со временем приводит к накоплению ошибки (дрейфу). Для получения точной и стабильной оценки угла ориентации колеса применяется комплементарный фильтр Калмана. Фильтр рекурсивно оценивает состояние системы (угол kalman_angle_state и смещение гироскопа kalman_bias_state), используя предсказания на основе показаний гироскопа и коррекцию на основе показаний акселерометра. (Реализация фильтра Калмана представлена в приложении 1).

Параметры фильтра q_angle_kalman, q_bias_kalman (шумы процесса) и r_measure_kalman (шум измерения акселерометра) подбираются экспериментально для достижения оптимального баланса между скоростью реакции и гладкостью оценки угла.

5.2. Определение состояния колеса

5.2.1. Калибровка системы координат "IMU-колесо"

Поскольку начальная ориентация IMU относительно колеса неизвестна, выполняется процедура калибровки (calibrate_imu_wheel_coordinate_system()). Пользователь устанавливает колесо так, чтобы первый актуатор (индекс 0) был направлен вертикально вниз. Система в течение некоторого времени усредняет показания отфильтрованного угла IMU. Разница между этим усредненным углом и целевым нулевым углом (0° для нижней точки) сохраняется как смещение imu_to_wheel_coordinate_offset_deg. (Реализация калибровки представлена в приложении 2)

5.2.2. Расчет текущего глобального угла, угловой скорости и направления движения

Эти параметры вычисляются в функции update_wheel_state() и сохраняются в структуре current_wheel_state. (Реализация расчета текущего глобального угла представлена в приложении 2).

5.3. Алгоритм управления актуаторами

Основная логика управления актуаторами реализована в функции process_actuator_logic().

5.3.1. Логика выбора актуатора для активации

Система стремится активировать актуаторы последовательно (от 0-го до 13-го и далее по кругу). Переменная next_actuator_to_fire_idx хранит индекс следующего ожидаемого актуатора.

- **Нормальный режим (движение вперед или покой):** actuator_to_evaluate_idx присваивается значение next_actuator_to_fire_idx.
- **Режим отката (current_wheel_state.direction == -1):** Для противодействия откату выбирается актуатор, предшествующий ожидаемому

(Реализация логики выбора актуатора представлена в приложении 3).

5.3.2. Проверка условия активации

Для выбранного actuator_to_evaluate_idx рассчитывается его текущий глобальный угол на колесе. Затем этот угол сравнивается с целевым углом OPTIMAL_PUSH_ANGLE_FROM_BOTTOM_DEG. Если актуатор находится в пределах окна ACTIVATION_WINDOW_DEG от целевого угла, он активируется. (Реализация проверки условия активации представлена в приложении 3).

5.3.3. Логика пропуска актуатора

Если планово ожидаемый актуатор не попал в окно активации и уже проехал оптимальную зону (его diff_actuator_to_target_deg стал отрицательным и меньше -1.0f), то next_actuator_to_fire_idx инкрементируется, чтобы система не "зависала" на ожидании уже пропущенного

актуатора. (Реализация логики пропуска актуатора представлена в приложении 3).

5.3.4. Управление временем работы актуатора

После активации актуатор остается включенным на время, определенное MAX_ACTUATOR_ON_TIME_MS (например, 500 мс). Это реализовано в функции handle_actuator_timeout(), которая проверяет время с момента активации и отключает актуатор, если оно превышено. (Реализация управления временем работы представлена в приложении 3).

Такая структура обеспечивает хорошую читаемость и позволяет легко модифицировать отдельные части алгоритма. Минимальный интервал между срабатываниями (MIN_INTERVAL_BETWEEN_SUCCESSFUL_FIRES_MS) также учитывается перед попыткой активации нового актуатора.

6. Пример Работы Программы

Для иллюстрации работы разработанного программного обеспечения рассмотрим гипотетический сценарий последовательной активации актуаторов и реакции системы на изменение условий. В данном примере предполагается, что активна стратегия "загребание", где оптимальный угол для толчка составляет приблизительно 329.0° (или -31.0° относительно нижней точки колеса, BDC), а окно активации +/- 7.0°.

Сценарий 1: Начало движения и последовательная активация

1. **Инициализация:** После включения и завершения калибровок (calibrate_imu_wheel_coordinate_system() и setup_mpu6050()), система готова к работе. Переменная next_actuator_to_fire_idx установлена в 0 (ожидается актуатор #1). Предположим, колесо неподвижно, current_wheel_state.direction = 0.
2. **Ожидание Актуатора #1:** Система начинает отслеживать глобальный угол актуатора #1 (индекс 0). Его глобальный угол равен текущему глобальному углу колеса current_wheel_state.global_angle_deg.
    - Если колесо провернуть так, что current_wheel_state.global_angle_deg приближается к 329.0°, то актуатор #1 войдет в окно активации (329.0° +/- 7.0°, т.е. от 322.0° до 336.0° — *Примечание: это неверно, окно вокруг 329, значит от 322 до 336, но если актуатор имеет смещение 0, то его угол и есть угол колеса. Цель 329.0, значит, когда угол колеса 329.0, актуатор 0 в целевой позиции*).
    - Допустим, current_wheel_state.global_angle_deg становится 330.0°. Разница с OPTIMAL_PUSH_ANGLE_FROM_BOTTOM_DEG (329.0°) составляет 1.0°. Это меньше ACTIVATION_WINDOW_DEG (7.0°).
3. **Активация Актуатора #1:** Функция process_actuator_logic() детектирует попадание в окно. Актуатор #1 (пин ACTUATOR_PINS[0]) активируется (digitalWrite(..., HIGH)).

Устанавливается current_active_actuator_index = 0 и actuator_activation_start_time_ms. Переменная next_actuator_to_fire_idx обновляется на 1 (ожидается актуатор #2).

4. **Таймаут Актуатора #1:** Актуатор #1 остается включенным в течение MAX_ACTUATOR_ON_TIME_MS (например, 500 мс). По истечении этого времени функция handle_actuator_timeout() отключает его (digitalWrite(..., LOW)), сбрасывает current_active_actuator_index в -1 и обновляет last_successfully_fired_actuator_idx = 0.
5. **Ожидание Актуатора #2:** Теперь система ожидает актуатор #2 (индекс 1). Его смещение на колесе составляет actuator_offsets_on_wheel_deg[1] ≈ 25.7°.
    - Предположим, толчок от актуатора #1 придал колесу вращение по часовой стрелке (CW). current_wheel_state.direction становится 1. Глобальный угол колеса current_wheel_state.global_angle_deg начинает уменьшаться (поскольку 0° - низ, а CW - это увеличение угла, но если 0° - это нулевая метка внизу, и CW увеличивает угол, то для "загребания" на ~329° колесо должно вращаться так, чтобы эта "нулевая метка" шла от 0° к 359°, 358° и т.д. чтобы актуатор сдвинулся вперед. Либо, если мы говорим о глобальном положении самого актуатора, то оно должно приближаться к 329°).
    - *Давайте переформулируем для ясности вращения*: Если мы хотим, чтобы актуатор, находящийся на физической позиции ~329° (позади BDC), толкнул колесо, то само колесо должно вращаться так, чтобы этот актуатор подошел к этой позиции. Если актуатор #1 только что отработал на ~329°, то для того, чтобы актуатор #2 (смещенный на +25.7° относительно #1) оказался на ~329°, колесо должно провернуться против часовой стрелки примерно на 25.7°. Но мы хотим движения CW.
    - *Корректное рассуждение для "загребания" (цель ~329°)*:

- Актуатор #1 (смещение 0°) сработал, когда current_wheel_state.global_angle_deg был ~329°.
- Колесо получило импульс и вращается CW. current_wheel_state.global_angle_deg увеличивается (например, от 329° к 0°, затем к 1°, 2°...).
- Мы ожидаем актуатор #2 (смещение ~25.7°). Его глобальный угол normalize_angle_deg(current_wheel_state.global_angle_deg + 25.7°).
- Чтобы актуатор #2 оказался на ~329°, current_wheel_state.global_angle_deg + 25.7° ≈ 329° (или 329° + 360° и т.д.). Это означает, что current_wheel_state.global_angle_deg ≈ 303.3°.
- Таким образом, после срабатывания актуатора #1 (когда нулевая метка была на 329°), колесо должно провернуться CW так, чтобы нулевая метка переместилась с 329° на 303.3° (пройдя через 0°). Это оборот почти на 334°. Это неверно.

Давайте пересмотрим логику вращения для "загребания". Если мы "загребаем" актуатором, который находится на глобальной позиции ~329° (позади BDC), и хотим двигаться CW (по часовой стрелке, увеличение current_wheel_state.global_angle_deg):

1. Актуатор X находится на глобальной позиции A_x.
2. Мы ждем, пока A_x станет ~329°.
3. Актуатор X срабатывает. Колесо получает импульс CW. current_wheel_state.global_angle_deg увеличивается.
4. Следующий актуатор Y (смещенный на +25.7° относительно X на раме колеса) теперь будет иметь глобальный угол A_y = normalize_angle_deg(current_wheel_state.global_angle_deg + offset_Y). Мы ждем, пока A_y достигнет ~329°.

Сценарий 2: Реакция на откат

1. **Движение вперед:** Предположим, актуатор #5 (индекс 4) только что отработал, и next_actuator_to_fire_idx = 5 (ожидаем актуатор #6). Колесо двигалось вперед (current_wheel_state.direction = 1).
2. **Обнаружение отката:** Внезапно, из-за препятствия или уклона, колесо начинает катиться назад. update_wheel_state() определяет это, и current_wheel_state.direction становится -1.
3. **Выбор коррекционного актуатора:** process_actuator_logic() видит current_wheel_state.direction = -1. actuator_to_evaluate_idx рассчитывается как (next_actuator_to_fire_idx - 1 + NUM_ACTUATORS) % NUM_ACTUATORS = (5 - 1 + 14) % 14 = 4. То есть, система попытается активировать актуатор #5 (индекс 4), который только что отработал или должен был быть следующим перед откатом. Флаг is_corrective_push устанавливается в true.
4. **Проверка окна для Актуатора #5:** Рассчитывается глобальный угол актуатора #5. Если из-за отката он снова оказался в оптимальной зоне для "загребания" (~329°), он будет активирован.
5. **Активация коррекционного толчка:** Если актуатор #5 активирован, он совершает толчок. next_actuator_to_fire_idx обновляется на (4 + 1) % 14 = 5. Система надеется, что движение вперед восстановлено и продолжит ожидать актуатор #6 в нормальном режиме.

Сценарий 3: Пропуск актуатора

1. **Ожидание актуатора:** Система ожидает next_actuator_to_fire_idx = N. Колесо вращается CW.
2. **Актуатор N не попадает в окно:** По какой-то причине (слишком быстрое вращение, неточность IMU, короткий цикл loop()) актуатор N проходит свою оптимальную позицию ~329°, но его глобальный угол не попал точно в окно +/- 7.0° в момент проверки.
3. **Обнаружение пропуска:** В последующих итерациях loop() глобальный угол актуатора N становится значительно

меньше 329.0° (например, 300°). Разница diff_actuator_to_target_deg становится большой отрицательной величиной (например, -29°).

4. **Инкремент next_actuator_to_fire_idx:** Условие diff_actuator_to_target_deg < -1.0f выполняется. next_actuator_to_fire_idx инкрементируется до (N + 1) % NUM_ACTUATORS. Система начинает ожидать следующий актуатор, не "зависая" на пропущенном.

Эти сценарии иллюстрируют основные аспекты логики программы: последовательную активацию, реакцию на изменение направления движения и обработку ситуаций, когда актуатор не может быть активирован в точно рассчитанный момент. Успешность работы всей системы сильно зависит от точности сенсорных данных, эффективности физического толчка актуатора и динамических свойств самого моноколеса.

Полный текст программы представлен в приложении 4.

7. Заключение

В ходе выполнения данной работы была разработана и программно реализована система управления для экспериментального прототипа моноколеса, движение которого предполагается осуществлять за счет последовательной активации радиально выдвигаемых штифтов.

Основные достигнутые результаты включают:

- Создана программная архитектура, обеспечивающая считывание данных с инерциального измерительного модуля MPU6050.
- Реализованы процедуры калибровки гироскопа и системы координат "IMU-колесо", что позволило корректно определять глобальную ориентацию моноколеса.
- Успешно применен фильтр Калмана для слияния данных акселерометра и гироскопа, обеспечивающий стабильную и относительно точную оценку угла и угловой скорости колеса.
- Разработан и реализован алгоритм выбора и активации актуаторов, основанный на геометрическом расчете оптимального угла для толчка (в частности, протестирована стратегия "загребание"). Алгоритм включает логику последовательного переключения между актуаторами, а также механизм коррекции при обнаружении отката колеса назад.
- Обеспечено управление временем работы актуаторов для предотвращения их перегрева.
- Создана система отладочного вывода, позволившая проводить мониторинг ключевых параметров и анализ поведения системы.

В ходе тестирования было установлено, что программная логика определения углов, выбора актуаторов и реакции на изменение состояния колеса в целом функционирует корректно. Стратегия "загребание" показала себя несколько более перспективной по сравнению с первоначальной идеей "толкать вперед", демонстрируя более частые и последовательные попытки активации.

Тем не менее, основной выявленной проблемой стала недостаточность развиваемого актуаторами импульса для обеспечения устойчивого и продолжительного самостоятельного движения моноколеса. После одного или нескольких толчков колесо, как правило, теряло скорость и останавливалось. Это указывает на необходимость дальнейших работ, в первую очередь, в направлении улучшения механической части (мощность актуаторов, снижение трения, оптимизация геометрии толчка) и, возможно, силовой электроники.

Несмотря на текущие ограничения в достижении стабильного движения, проделанная работа закладывает важную программную и алгоритмическую основу для дальнейших исследований и усовершенствования подобных систем. Полученный опыт и выявленные проблемы являются ценным вкладом в понимание сложностей, связанных с реализацией нетрадиционных способов локомоции. Перспективы проекта лежат в комплексной доработке как аппаратной, так и программной частей, с возможным применением более сложных алгоритмов управления и адаптации.

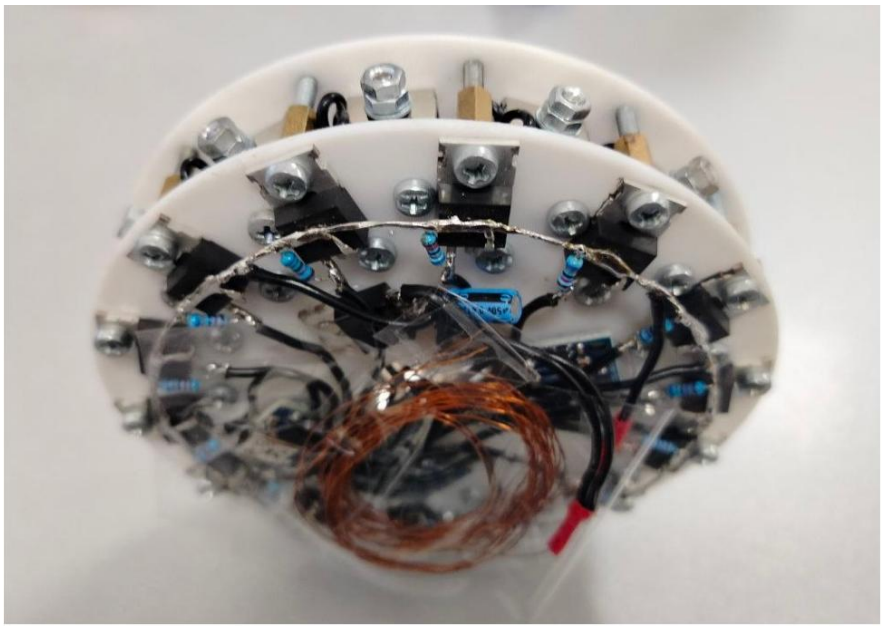

Рисунок 3 –Лабораторный прототип моноколеса

8. Список использованной литературы

1. Гим К.Г., Ким Дж. Рингбот: моноцикл с ногами // IEEE Транзакции по робототех нике. 2024. Т. 40. С. 1890–1905. DOI: 10.1109/TRO.2024.3362326. EDN: TIWZEX.
2. Чжан Ю., Цзинь Х., Чжао Дж. Управление динамическим балансом дву-гироско пического моноцикла на основе контроллера слайдинга // Дат-чики. 2023. Т. 23, №3. С. 1064. DOI: 10.3390/s23031064.
3. Хо М.-Т., Ризал Ю., Чен Ю.-Л. Управление балансом моноцикла // 23-й междуна родный симпозиум по промышленной электронике (ISIE). 2014. С. 1–6. DOI: 10.1109/ISIE.2014.6864782.
4. Соленоид JF-0520B [Электронный ресурс]. URL: https://iarduino.ru/shop/Mehanika/ solenoid-tau-0520.html
5. Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). *Introduction to Autonomous Mobile Robots* (2nd ed.). MIT Press.

Приложение 1

Считывание "сырых" данных акселерометра и гироскопа:

```
// Фрагмент из read_imu_data_raw()
IMUData data;
int16_t acc_y_raw, acc_z_raw, gyro_x_raw_current;

// ... (Код для связи с MPU6050 и чтения регистров) ...
acc_y_raw = (Wire.read() << 8) | Wire.read();
acc_z_raw = (Wire.read() << 8) | Wire.read();
gyro_x_raw_current = (Wire.read() << 8) | Wire.read();

// Расчет "сырого" угла от акселерометра
data.acc_angle_raw = normalize_angle_deg(atan2f((float)acc_z_raw, (float)acc_y_raw + 1e-6f)
* (180.0f / M_PI));
// Угловая скорость с учетом калибровки смещения (mpu_gyro_x_offset_raw вычисляется
ранее)
data.gyro_rate_cal = ((float)gyro_x_raw_current - mpu_gyro_x_offset_raw) / 131.0f; // 131
LSB/°/s для ±250°/s
return data;
```

Калибровка смещения гироскопа:

```
// Фрагмент из setup_mpu6050()
long gyro_sum_raw = 0;
for (int i = 0; i < GYRO_CALIBRATION_SAMPLES; i++) {
   // ... (чтение данных гироскопа) ...
   gyro_sum_raw += raw_gyro_x_value; // Предположим, это сырое значение
   delay(2);
}
mpu_gyro_x_offset_raw = (float)gyro_sum_raw / GYRO_CALIBRATION_SAMPLES;
```

Фильтр Калмана для слияния данных:

```
// Фрагмент из update_kalman_filter()
float predicted_rate = gyro_rate_cal - kalman_bias_state;
kalman_angle_state += dt_sec * predicted_rate; // Предсказание угла

// ... (Обновление ковариационной матрицы P) ...

float measurement_error = shortest_angle_diff_deg(acc_angle_raw, kalman_angle_state); //
Ошибка

// ... (Расчет усиления Калмана K0, K1) ...

kalman_angle_state += k0_kalman_gain * measurement_error; // Коррекция угла
kalman_bias_state  += k1_kalman_gain * measurement_error; // Коррекция смещения гиро-
скопа
```

```
// ... (Обновление ковариационной матрицы P) ...
return normalize_angle_deg(kalman_angle_state);
```

Приложение 2

Калибровка системы координат "IMU-колесо":

```
// Фрагмент из calibrate_imu_wheel_coordinate_system()
// ... (усреднение filtered_raw_angle_at_bottom) ...
imu_to_wheel_coordinate_offset_deg = normalize_angle_deg(0.0f - avg_filtered_raw_angle_at_bottom);
```

Расчет текущего глобального угла, угловой скорости и направления движения:

```
// Фрагмент из update_wheel_state()
// Глобальный угол нулевой метки колеса (0° = низ, CW = положительное)
current_wheel_state.global_angle_deg = normalize_angle_deg(filtered_raw_imu_angle_deg + imu_to_wheel_coordinate_offset_deg);
current_wheel_state.angular_velocity_dps = imu_data.gyro_rate_cal;

// Определение направления движения
if (current_wheel_state.angular_velocity_dps > GYRO_STATIONARY_THRESHOLD_DPS) {
   current_wheel_state.direction = 1;  // Вперед (CW)
} else if (current_wheel_state.angular_velocity_dps < -GYRO_STATIONARY_THRESHOLD_DPS) {
   current_wheel_state.direction = -1; // Назад (CCW, откат)
} else {
   current_wheel_state.direction = 0;  // Неподвижно
}
```

Приложение 3

Логика выбора актуатора для активации:

```
actuator_to_evaluate_idx = (next_actuator_to_fire_idx - 1 + NUM_ACTUATORS) %
NUM_ACTUATORS;
is_corrective_push = true;
```

Проверка условия активации:

```
float current_eval_actuator_global_angle_deg = normalize_angle_deg(cur-
rent_wheel_state.global_angle_deg + actuator_offsets_on_wheel_deg[actuator_to_evalu-
ate_idx]);
float diff_actuator_to_target_deg = shortest_angle_diff_deg(current_eval_actuator_global_an-
gle_deg, target_push_global_angle_deg);
if (fabs(diff_actuator_to_target_deg) < ACTIVATION_WINDOW_DEG) {
   digitalWrite(ACTUATOR_PINS[actuator_to_evaluate_idx], HIGH);
   // ... (обновление состояния активного актуатора) ...
   next_actuator_to_fire_idx = (current_active_actuator_index + 1) % NUM_ACTUATORS; //
Переход к следующему
}
```

Логика пропуска актуатора:

```
else { // Если актуатор НЕ в окне
   if (!is_corrective_push &&
      actuator_to_evaluate_idx == next_actuator_to_fire_idx &&
      diff_actuator_to_target_deg < -1.0f ) { // Условие пропуска
      // ... (логирование пропуска) ...
      next_actuator_to_fire_idx = (next_actuator_to_fire_idx + 1) % NUM_ACTUATORS;
   }
}
```

Управление временем работы актуатора:

```
// Фрагмент из handle_actuator_timeout()
if (current_time_ms - actuator_activation_start_time_ms >= MAX_ACTUA-
TOR_ON_TIME_MS) {
   digitalWrite(ACTUATOR_PINS[current_active_actuator_index], LOW);
   // ... (обновление состояния) ...
}
```

Приложение 4

Полный код программы:

```
#include <Wire.h>
#include <math.h>
#include <Arduino.h>

struct IMUData {
  float acc_angle_raw;
  float gyro_rate_cal;
};

struct WheelState {
  float global_angle_deg;
  float angular_velocity_dps;
  int   direction;
};

#define NUM_ACTUATORS    14
#define MPU6050_ADDR     0x68
#define MAX_ACTUATOR_ON_TIME_MS 500
#define SERIAL_DEBUG     true
#define MIN_INTERVAL_BETWEEN_SUCCESSFUL_FIRES_MS 0
#define GYRO_CALIBRATION_SAMPLES 1000
#define COORDINATE_SYSTEM_CALIBRATION_SAMPLES 200
#define GYRO_STATIONARY_THRESHOLD_DPS 1.5f

const uint8_t ACTUATOR_PINS[NUM_ACTUATORS] = {2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
A0, A1};

const float WHEEL_DIAMETER_CM = 12.0f;
const float WHEEL_RADIUS_CM   = WHEEL_DIAMETER_CM / 2.0f;
const float ROD_EXTENSION_CM  = 1.0f;

const float ANGLE_OFFSET_CALC_DEG = acos(WHEEL_RADIUS_CM / (WHEEL_RA-
DIUS_CM + ROD_EXTENSION_CM)) * (180.0f / M_PI);
const float _temp_optimal_push_angle = 0.0f - ANGLE_OFFSET_CALC_DEG;
const float OPTIMAL_PUSH_ANGLE_FROM_BOTTOM_DEG = (_temp_optimal_push_angle
< 0.0f) ? (_temp_optimal_push_angle + 360.0f) : _temp_optimal_push_angle;
const float ACTIVATION_WINDOW_DEG = 7.0f;

float q_angle_kalman   = 0.003f;
float q_bias_kalman    = 0.0005f;
float r_measure_kalman = 0.01f;

float kalman_angle_state = 0.0f;
float kalman_bias_state  = 0.0f;
float p_kalman[2][2] = { { 1.0f, 0.0f }, { 0.0f, 1.0f } };

float actuator_offsets_on_wheel_deg[NUM_ACTUATORS];
unsigned long last_loop_time_us = 0;
```

```
int current_active_actuator_index = -1;
unsigned long actuator_activation_start_time_ms = 0;

unsigned long last_successful_fire_time_global_ms = 0;
int last_successfully_fired_actuator_idx = -1;
int next_actuator_to_fire_idx = 0;

float imu_to_wheel_coordinate_offset_deg = 0.0f;
bool  initial_calibration_completed = false;
WheelState current_wheel_state;
int previous_wheel_direction = 0;

float normalize_angle_deg(float angle) {
  angle = fmodf(angle, 360.0f);
  if (angle < 0.0f) angle += 360.0f;
  return angle;
}

float shortest_angle_diff_deg(float angle_a, float angle_b) {
  float diff = normalize_angle_deg(angle_a) - normalize_angle_deg(angle_b);
  if (diff > 180.0f) diff -= 360.0f;
  else if (diff <= -180.0f) diff += 360.0f;
  return diff;
}

float mpu_gyro_x_offset_raw = 0.0f;
void setup_mpu6050() {
  Wire.beginTransmission(MPU6050_ADDR); Wire.write(0x6B); Wire.write(0); Wire.endTrans-
mission(true);
  Wire.beginTransmission(MPU6050_ADDR); Wire.write(0x1B); Wire.write(0x00); Wire.end-
Transmission(true);
  Wire.beginTransmission(MPU6050_ADDR); Wire.write(0x1C); Wire.write(0x00); Wire.end-
Transmission(true);

  if (SERIAL_DEBUG) Serial.println(F("Калибровка смещения гироскопа X... Держите
неподвижно."));
  long gyro_sum_raw = 0;
  for (int i = 0; i < 200; i++) {
    Wire.beginTransmission(MPU6050_ADDR); Wire.write(0x43); Wire.endTransmission(false);
    Wire.requestFrom(MPU6050_ADDR, 2, true); Wire.read(); Wire.read(); delay(2);
  }
  for (int i = 0; i < GYRO_CALIBRATION_SAMPLES; i++) {
    Wire.beginTransmission(MPU6050_ADDR); Wire.write(0x43); Wire.endTransmission(false);
    Wire.requestFrom(MPU6050_ADDR, 2, true);
    gyro_sum_raw += (int16_t)((Wire.read() << 8) | Wire.read());
    delay(2);
  }
  mpu_gyro_x_offset_raw = (float)gyro_sum_raw / GYRO_CALIBRATION_SAMPLES;
  if (SERIAL_DEBUG) { Serial.print(F("Калибровка Gyro X завершена. Смещение (raw): "));
Serial.println(mpu_gyro_x_offset_raw); }
}
```

```
IMUData read_imu_data_raw() {
  IMUData data;
  int16_t acc_y_raw, acc_z_raw, gyro_x_raw_current; // local variables in snake_case

  Wire.beginTransmission(MPU6050_ADDR); Wire.write(0x3B); Wire.endTransmission(false);
  Wire.requestFrom(MPU6050_ADDR, 14, true);

  Wire.read(); Wire.read();
  acc_y_raw = (Wire.read() << 8) | Wire.read();
  acc_z_raw = (Wire.read() << 8) | Wire.read();
  Wire.read(); Wire.read();
  gyro_x_raw_current = (Wire.read() << 8) | Wire.read();

  data.acc_angle_raw = normalize_angle_deg(atan2f((float)acc_z_raw, (float)acc_y_raw + 1e-6f)
* (180.0f / M_PI));
  data.gyro_rate_cal = ((float)gyro_x_raw_current - mpu_gyro_x_offset_raw) / 131.0f;
  return data;
}

float update_kalman_filter(float acc_angle_raw, float gyro_rate_cal, float dt_sec) {
  float predicted_rate = gyro_rate_cal - kalman_bias_state;
  kalman_angle_state += dt_sec * predicted_rate;

  p_kalman[0][0] += dt_sec * (dt_sec * p_kalman[1][1] - p_kalman[0][1] - p_kalman[1][0] +
q_angle_kalman);
  p_kalman[0][1] -= dt_sec * p_kalman[1][1];
  p_kalman[1][0] -= dt_sec * p_kalman[1][1];
  p_kalman[1][1] += q_bias_kalman * dt_sec;

  float measurement_error = shortest_angle_diff_deg(acc_angle_raw, kalman_angle_state);
  float s_innovation_covariance = p_kalman[0][0] + r_measure_kalman;
  if (fabs(s_innovation_covariance) < 1e-9) s_innovation_covariance = 1e-9f;

  float k0_kalman_gain = p_kalman[0][0] / s_innovation_covariance;
  float k1_kalman_gain = p_kalman[1][0] / s_innovation_covariance;

  kalman_angle_state += k0_kalman_gain * measurement_error;
  kalman_bias_state  += k1_kalman_gain * measurement_error;

  float p00_temp = p_kalman[0][0];
  float p01_temp = p_kalman[0][1];
  p_kalman[0][0] -= k0_kalman_gain * p00_temp;
  p_kalman[0][1] -= k0_kalman_gain * p01_temp;
  p_kalman[1][0] -= k1_kalman_gain * p00_temp;
  p_kalman[1][1] -= k1_kalman_gain * p01_temp;

  return normalize_angle_deg(kalman_angle_state);
}

void calibrate_imu_wheel_coordinate_system() {
  if (SERIAL_DEBUG) {
```

```
  Serial.println(F("\nКалибровка системы координат IMU-Колесо..."));
  Serial.println(F("Установите колесо так, чтобы АКТУАТОР 1 (пин ACTUATOR_PINS[0])
был направлен ВЕРТИКАЛЬНО ВНИЗ."));
  Serial.println(F("Это положение будет соответствовать 0° глобального угла колеса."));
  Serial.print(F("Ожидание 5 секунд для стабилизации..."));
 }
 delay(5000);
 if (SERIAL_DEBUG) Serial.println(F(" Начало сбора данных."));

 float sum_filtered_raw_angles = 0.0f;
 for (int i = 0; i < 100; i++) {
  IMUData imu = read_imu_data_raw();
  update_kalman_filter(imu.acc_angle_raw, imu.gyro_rate_cal, 0.01f);
  delay(5);
 }
 for (int i = 0; i < COORDINATE_SYSTEM_CALIBRATION_SAMPLES; i++) {
  IMUData imu = read_imu_data_raw();
  sum_filtered_raw_angles += update_kalman_filter(imu.acc_angle_raw, imu.gyro_rate_cal,
0.01f);
  delay(10);
 }
 float avg_filtered_raw_angle_at_bottom = sum_filtered_raw_angles / COORDINATE_SYS-
TEM_CALIBRATION_SAMPLES;
 imu_to_wheel_coordinate_offset_deg = normalize_angle_deg(0.0f - avg_filtered_raw_an-
gle_at_bottom);

 if (SERIAL_DEBUG) {
  Serial.print(F("Средний отфильтрованный \"сырой\" угол IMU (когда актуатор 1 внизу):
")); Serial.println(avg_filtered_raw_angle_at_bottom, 2);
  Serial.print(F("Рассчитанное смещение для СК колеса: ")); Serial.println(imu_to_wheel_co-
ordinate_offset_deg, 2);
  Serial.print(F("Тестовый глобальный угол колеса с коррекцией: ")); Serial.println(normal-
ize_angle_deg(avg_filtered_raw_angle_at_bottom + imu_to_wheel_coordinate_offset_deg), 2);
  Serial.println(F("Калибровка СК завершена."));
 }
 initial_calibration_completed = true;
}

void update_wheel_state(float dt_sec) {
  IMUData imu_data = read_imu_data_raw();
  float filtered_raw_imu_angle_deg = update_kalman_filter(imu_data.acc_angle_raw,
imu_data.gyro_rate_cal, dt_sec);

  current_wheel_state.global_angle_deg = normalize_angle_deg(filtered_raw_imu_angle_deg +
imu_to_wheel_coordinate_offset_deg);
  current_wheel_state.angular_velocity_dps = imu_data.gyro_rate_cal;

  previous_wheel_direction = current_wheel_state.direction;
  if (current_wheel_state.angular_velocity_dps > GYRO_STATIONARY_THRESHOLD_DPS)
{
      current_wheel_state.direction = 1;
```

```
  } else if (current_wheel_state.angular_velocity_dps < -GYRO_STATIONARY_THRESH-
OLD_DPS) {
    current_wheel_state.direction = -1;
  } else {
    current_wheel_state.direction = 0;
  }
}

void handle_actuator_timeout(unsigned long current_time_ms) {
  if (current_active_actuator_index != -1) {
    if (current_time_ms - actuator_activation_start_time_ms >= MAX_ACTUA-
TOR_ON_TIME_MS) {
      digitalWrite(ACTUATOR_PINS[current_active_actuator_index], LOW);
      if (SERIAL_DEBUG) {
        Serial.print(F("Актуатор #")); Serial.print(current_active_actuator_index + 1);
        Serial.println(F(" ОТКЛЮЧЕН по таймауту."));
      }
      last_successful_fire_time_global_ms = current_time_ms;
      last_successfully_fired_actuator_idx = current_active_actuator_index;
      current_active_actuator_index = -1;
    }
  }
}

void process_actuator_logic(unsigned long current_time_ms) {
  if (current_active_actuator_index != -1 || !initial_calibration_completed ||
    (current_time_ms - last_successful_fire_time_global_ms < MIN_INTERVAL_BE-
TWEEN_SUCCESSFUL_FIRES_MS)) {
    return;
  }

  int actuator_to_evaluate_idx = -1;
  bool is_corrective_push = false;

  if (current_wheel_state.direction == -1) {
    actuator_to_evaluate_idx = (next_actuator_to_fire_idx - 1 + NUM_ACTUATORS) %
NUM_ACTUATORS;
    is_corrective_push = true;
    if (SERIAL_DEBUG && previous_wheel_direction != -1) {
      Serial.print(F("ОТКАТ! Попытка коррекции пред. (#")); Serial.print(actuator_to_eval-
uate_idx + 1);
      Serial.print(F(") относ. ожид. (#")); Serial.print(next_actuator_to_fire_idx + 1); Se-
rial.println(F(")"));
    }
  } else {
    actuator_to_evaluate_idx = next_actuator_to_fire_idx;
    is_corrective_push = false;
    if (SERIAL_DEBUG) {
      if (current_wheel_state.direction == 1 && previous_wheel_direction != 1) {
        Serial.print(F("Движение ВПЕРЕД: ожидаем акт. #")); Serial.println(actua-
tor_to_evaluate_idx + 1);
      } else if (current_wheel_state.direction == 0 && previous_wheel_direction != 0) {
```

```
            Serial.print(F("СТОИМ: ожидаем акт. #")); Serial.println(actuator_to_evaluate_idx +
1);
        }
      }
    }

    if (actuator_to_evaluate_idx != -1) {
      float current_eval_actuator_global_angle_deg = normalize_angle_deg(cur-
rent_wheel_state.global_angle_deg + actuator_offsets_on_wheel_deg[actuator_to_evalu-
ate_idx]);
      float target_push_global_angle_deg = OPTIMAL_PUSH_ANGLE_FROM_BOT-
TOM_DEG;
      float diff_actuator_to_target_deg = shortest_angle_diff_deg(current_eval_actua-
tor_global_angle_deg, target_push_global_angle_deg);

      if (fabs(diff_actuator_to_target_deg) < ACTIVATION_WINDOW_DEG) {
        digitalWrite(ACTUATOR_PINS[actuator_to_evaluate_idx], HIGH);
        current_active_actuator_index = actuator_to_evaluate_idx;
        actuator_activation_start_time_ms = current_time_ms;

        if (SERIAL_DEBUG) {
          Serial.print(is_corrective_push ? F("КОРР. ") : F(""));
          Serial.print(F("АКТИВИРОВАН АКТУАТОР #")); Serial.print(current_active_actua-
tor_index + 1);
          Serial.print(F(" (инд ")); Serial.print(current_active_actuator_index);
          Serial.print(F(") Глоб.угол: ")); Serial.print(current_eval_actuator_global_angle_deg,
1);
          Serial.print(F("° (цель ~")); Serial.print(target_push_global_angle_deg,1);
          Serial.print(F("° diff "));Serial.print(diff_actuator_to_target_deg,1);
          Serial.print(F("°). Напр: ")); Serial.print(current_wheel_state.direction);
          int next_after_this_fire = (current_active_actuator_index + 1) % NUM_ACTUATORS;
          Serial.print(F(". След.ожид: #")); Serial.println(next_after_this_fire + 1);
        }
        next_actuator_to_fire_idx = (current_active_actuator_index + 1) % NUM_ACTUA-
TORS;
      } else {
        if (!is_corrective_push &&
          actuator_to_evaluate_idx == next_actuator_to_fire_idx &&
          diff_actuator_to_target_deg < -1.0f ) {

          int old_expected = next_actuator_to_fire_idx;
          next_actuator_to_fire_idx = (next_actuator_to_fire_idx + 1) % NUM_ACTUATORS;
          if (SERIAL_DEBUG) {
            Serial.print(F("Акт.#")); Serial.print(old_expected + 1);
            Serial.print(F(" пропустил окно (угол ")); Serial.print(current_eval_actua-
tor_global_angle_deg, 1);
            Serial.print(F("°, цель ")); Serial.print(target_push_global_angle_deg, 1);
            Serial.print(F("°, diff ")); Serial.print(diff_actuator_to_target_deg, 1);
            Serial.print(F("°). Нов.ожид.#"));Serial.println(next_actuator_to_fire_idx+1);
          }
        }
      }
```

```
    }
}

void print_debug_info(unsigned long current_time_ms) {
    static unsigned long last_serial_debug_print_ms = 0;
    if (SERIAL_DEBUG && (current_time_ms - last_serial_debug_print_ms >= 500)) {
        Serial.print(F("T:")); Serial.print(current_time_ms / 1000.0f, 1);
        Serial.print(F(" Угол:")); Serial.print(current_wheel_state.global_angle_deg, 1);
        Serial.print(F("° Скор:")); Serial.print(current_wheel_state.angular_velocity_dps, 1);
        Serial.print(F("°/с Напр:")); Serial.print(current_wheel_state.direction);
        Serial.print(F(" Посл.сраб(#индекс):")); Serial.print(last_successfully_fired_actuator_idx
+1); Serial.print(F("(")); Serial.print(last_successfully_fired_actuator_idx); Serial.print(F(")"));
        Serial.print(F(" Ожид.#")); Serial.print(next_actuator_to_fire_idx+1);
        if (current_active_actuator_index != -1) {
            Serial.print(F(" АКТ.#:")); Serial.print(current_active_actuator_index+1);
        }
        Serial.println();
        last_serial_debug_print_ms = current_time_ms;
    }
}

void setup() {
  Serial.begin(115200);
  Wire.begin();

  if (SERIAL_DEBUG) {
    Serial.println(F("\n=== Инициализация Системы Моноколеса ==="));
    Serial.println(F("Загребание (~329.0°)"));
  }

  setup_mpu6050();

  for (int i = 0; i < NUM_ACTUATORS; i++) {
    pinMode(ACTUATOR_PINS[i], OUTPUT);
    digitalWrite(ACTUATOR_PINS[i], LOW);
    actuator_offsets_on_wheel_deg[i] = normalize_angle_deg(i * (360.0f / NUM_ACTUA-
TORS));
  }
  if (SERIAL_DEBUG) {
    Serial.print(F("Угловые смещения актуаторов на колесе (°): "));
    for(int i=0; i<NUM_ACTUATORS; ++i) { Serial.print(actuator_offsets_on_wheel_deg[i],1);
Serial.print(F(" "));}
    Serial.println();
  }

  IMUData initial_imu_data = read_imu_data_raw();
  kalman_angle_state = initial_imu_data.acc_angle_raw;
  kalman_bias_state = 0.0f;
  current_wheel_state.direction = 0;

  current_active_actuator_index = -1;
  last_successful_fire_time_global_ms = 0;
```

```
  last_successfully_fired_actuator_idx = -1;
  next_actuator_to_fire_idx = 0;

  previous_wheel_direction = 0;
  last_loop_time_us = micros();

  calibrate_imu_wheel_coordinate_system();

  if (SERIAL_DEBUG) {
    Serial.print(F("Оптимальный угол для толчка (глобальный): ")); Serial.println(OPTI-
MAL_PUSH_ANGLE_FROM_BOTTOM_DEG, 1);
    Serial.print(F("Окно активации: +/- ")); Serial.print(ACTIVATION_WINDOW_DEG,1); Se-
rial.println(F(" град."));
    Serial.print(F("Время работы актуатора: ")); Serial.print(MAX_ACTUA-
TOR_ON_TIME_MS); Serial.println(F(" мс"));
    Serial.println(F("Инициализация завершена. Задержка 3 сек перед запуском..."));
  }
  delay(3000);
}

void loop() {
  unsigned long current_time_ms = millis();
  unsigned long current_time_us = micros();
  float dt_sec = (current_time_us - last_loop_time_us) / 1000000.0f;
  last_loop_time_us = current_time_us;

  if (dt_sec <= 0.0f || dt_sec > 0.1f) {
    dt_sec = 0.01f;
  }

  update_wheel_state(dt_sec);
  handle_actuator_timeout(current_time_ms);
  process_actuator_logic(current_time_ms);
  print_debug_info(current_time_ms);
}
```