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

Курсовая работа

Дисциплина: Объектно-ориентированное программирование

Тема: Разработка алгоритмов управления движением моноколеса с радиальными актуаторами

Выполнили студент гр. 3331506/20101 Миронов В.В.

Преподаватель Ананьевский М. С.

Санкт-Петербург
2025

Оглавление

1. Введение

Управление движением мобильных роботов с нетрадиционной кинематикой, таких как моноколеса с активными элементами взаимодействия с поверхностью, представляет собой сложную и актуальную задачу современной робототехники [1, 5]. После получения достоверной информации о текущем состоянии системы (угле ориентации и скорости вращения), ключевым этапом становится разработка алгоритмов, которые на основе этих данных принимают решения об управляющих воздействиях для достижения целенаправленного движения. В случае моноколеса с радиально выдвигаемыми штифтами, это подразумевает определение оптимального момента, последовательности и параметров активации актуаторов [4].

Актуальность данной работы обусловлена необходимостью создания эффективной стратегии управления для экспериментального прототипа моноколеса, использующего для движения выдвижные штифты. Разработка таких алгоритмов позволит исследовать потенциал данного принципа локомоции и выявить его особенности. Данная работа является логическим продолжением этапа разработки системы сбора и оценки состояния моноколеса.

Целью данной части работы является разработка, программная реализация и предварительное тестирование алгоритмов управления движением моноколеса, основанных на последовательной активации радиальных актуаторов.

Задачи, решаемые в данной части работы:

1. Формализация и реализация выбранной стратегии толчка ("загребание") на основе геометрических параметров системы и расчетного оптимального угла активации.
2. Разработка алгоритма последовательного выбора актуаторов для обеспечения движения в заданном направлении.
3. Реализация механизма коррекции движения при обнаружении отката колеса назад.

4. Внедрение логики пропуска актуатора в случае невозможности его своевременной активации.
5. Обеспечение контроля времени работы актуаторов и минимальных интервалов между их срабатываниями.
6. Анализ поведения системы на основе отладочных данных и выявление ограничений предложенных алгоритмов.

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 [1]. 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 [1, 5]. 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" [1]. 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 [5].

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.

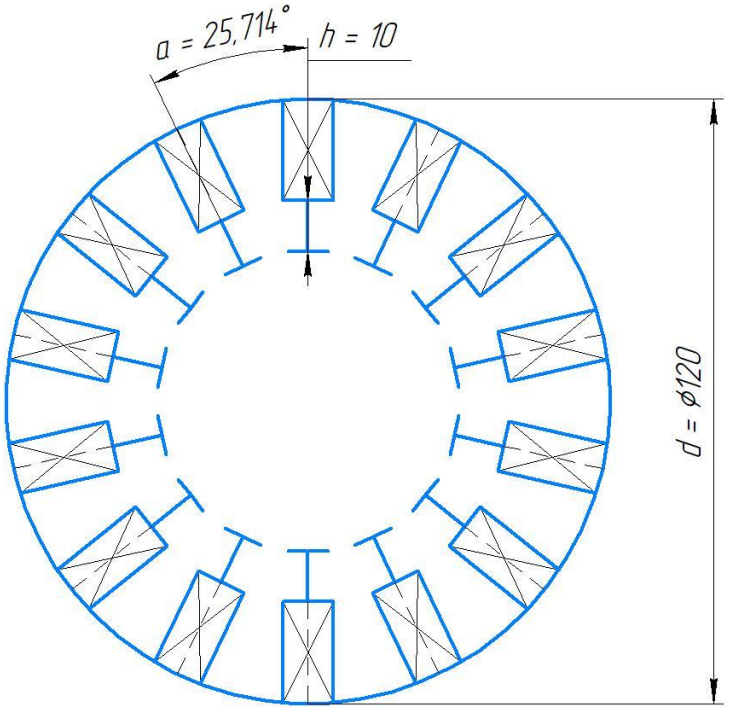

Рисунок 1 – Кинематическая схема моноколеса

4. Электрическая Схема и Компоненты

4.1. Система актуаторов и их управление

В качестве актуаторов, обеспечивающих движение моноколеса, используются четырнадцать (14) соленоидов (втягивающих электромагнитов) модели JF-0520B [4]. 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.

4.2. 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 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.

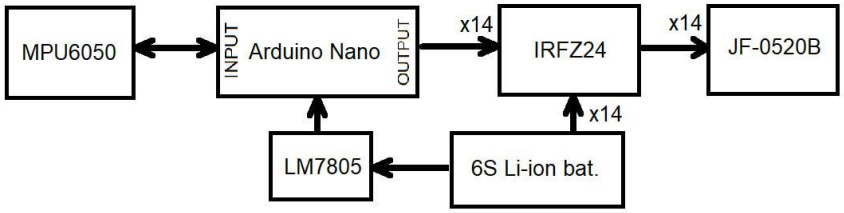

Рисунок 2 – Структурная электрическая схема моноколеса

5. Программная Реализация и Алгоритмы Управления

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.

Полный код программы представлен в приложении 1.

7. Заключение

В рамках данной части работы была успешно завершена разработка и программная реализация комплекса алгоритмов управления движением для экспериментального прототипа моноколеса с радиальными актуаторами. Основной задачей являлось создание логической основы для целенаправленной активации штифтов с целью инициирования и поддержания движения.

Были решены следующие ключевые задачи:

- Формализована и имплементирована выбранная стратегия толчка "загребание", основанная на расчете оптимального угла активации, исходя из геометрических параметров системы. Это позволило определить целевые условия для срабатывания каждого актуатора.
- Разработан и программно реализован алгоритм последовательного выбора актуаторов, обеспечивающий их поочередную активацию для теоретического продвижения колеса в заданном направлении. Введена переменная next_actuator_to_fire_idx для отслеживания плановой последовательности.
- Создан механизм коррекции движения, который при детектировании отката колеса назад инициирует попытку активации "предыдущего" в последовательности актуатора, с целью противодействия нежелательному движению и восстановления предполагаемого направления.
- Внедрена логика пропуска актуатора, которая позволяет системе не "зависать" на ожидании актуатора, уже прошедшего свою оптимальную зону активации, и переходить к следующему в последовательности.
- Реализовано управление временными параметрами работы актуаторов, включая максимальное время их удержания во включенном состоянии и учет минимального интервала между срабатываниями.

Таким образом, создана полная алгоритмическая база, необходимая для управления движением моноколеса на основе полученных ранее (в первой части исследования) данных о его состоянии. Разработанное программное обеспечение готово к этапу всестороннего тестирования на физическом прототипе,

анализа его реальной производительности, выявления ограничений и определения путей дальнейшей оптимизации как самих алгоритмов, так и аппаратной части моноколеса. Результаты такого тестирования и обсуждение эффективности предложенных алгоритмов будут представлены в последующих работах.

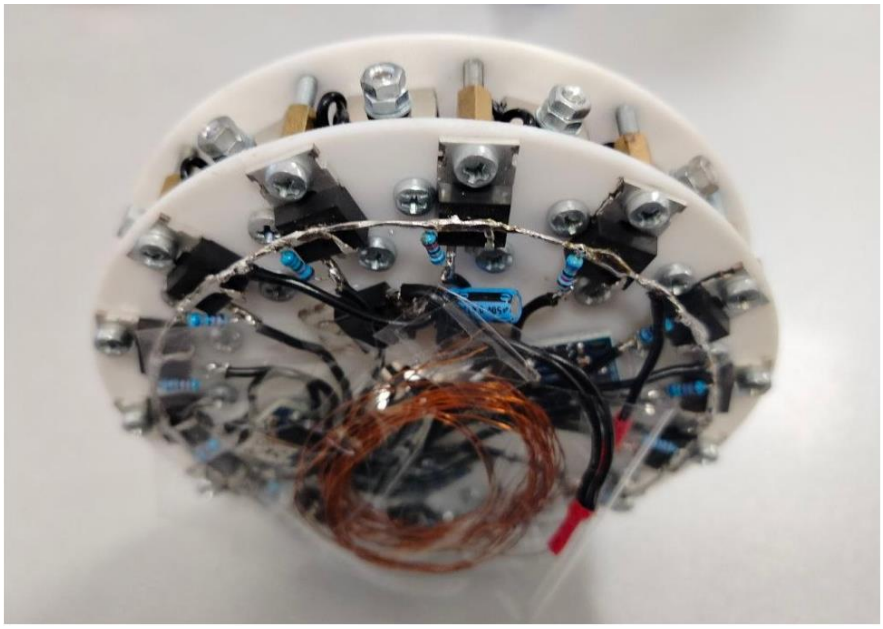

Рисунок 3 –Лабораторный прототип моноколеса

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://iar-duino.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

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);
}
```