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

# Курсовая работа

Дисциплина: Объектно-ориентированное программирование

Тема: Разработка и программная реализация системы сбора данных и оценки состояния моноколеса с IMU

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

Преподаватель Ананьевский М. С.

Санкт-Петербург
2025

## Оглавление

# 1. Введение

Современные мобильные робототехнические системы, особенно с нетрадиционными принципами движения, требуют точного и надежного определения их текущего состояния в пространстве для эффективного управления. Моноколесные платформы, в силу своей динамической нестабильности и специфики взаимодействия с окружением, предъявляют особо высокие требования к качеству получаемых данных об ориентации и скорости [2, 3, 5]. Инерциальные измерительные модули (IMU), объединяющие акселерометры и гироскопы, являются ключевыми сенсорами для решения этой задачи, однако их показания подвержены шумам и дрейфу, что требует применения специализированных алгоритмов обработки и фильтрации.

Актуальность данной работы заключается в необходимости создания робастной подсистемы сбора и обработки сенсорных данных для экспериментального прототипа моноколеса, движение которого предполагается осуществлять за счет выдвигаемых штифтов [1]. Качественная оценка состояния колеса является критически важной основой для последующей реализации любых алгоритмов управления движением.

Целью данной части работы является разработка и программная реализация системы получения точных данных об угловой ориентации и скорости вращения моноколеса с использованием IMU MPU6050 и микроконтроллера Arduino Nano.

Задачи, решаемые в данной части работы:

1. Организация считывания "сырых" данных с акселерометра и гироскопа MPU6050.
2. Реализация процедур калибровки смещения нуля гироскопа и системы координат "IMU-колесо".
3. Применение фильтра Калмана для комплексирования данных акселерометра и гироскопа с целью получения точной и помехоустойчивой оценки угла и угловой скорости колеса [5].

4. Обеспечение отладочного вывода для верификации работы подсистемы сбора данных.

# 2. 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.

# 3. 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.

## 3.1. 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.

## 3.2. 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 [2, 5].

## 3.3. 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.

# 4. 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.

## 4.1. Получение и обработка данных с IMU

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

### 4.1.1. 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).

### 4.1.2. 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).

### 4.1.3. 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 [2, 5]. 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.

## 4.2. Определение состояния колеса

### 4.2.1. 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)

### 4.2.2. 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).

Полный текст программы представлен в приложении 4.

# 5. Заключение

В рамках данной части работы была успешно разработана и программно реализована система сбора данных и оценки состояния для экспериментального моноколеса. Были решены задачи по аппаратному сопряжению и программному взаимодействию с инерциальным измерительным модулем MPU6050, включая процедуры его инициализации и калибровки.

Ключевым элементом системы стала реализация фильтра Калмана, который позволил эффективно объединять показания акселерометра и гироскопа, обеспечивая получение стабильной и достаточно точной оценки текущего угла ориентации и угловой скорости моноколеса. Проведенная калибровка системы координат "IMU-колесо" позволила корректно соотносить показания датчика с физическим положением колеса.

Созданная подсистема формирует надежную информационную базу о состоянии объекта управления. Получаемые данные об угле и скорости являются необходимым и достаточным входным сигналом для разработки и реализации алгоритмов управления движением моноколеса, что будет рассмотрено в следующей части исследования. Дальнейшие улучшения данной подсистемы могут включать использование более продвинутых IMU или применение более сложных алгоритмов фильтрации для повышения точности в условиях интенсивных динамических нагрузок.

# 6. 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

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_SYSTEM_CALIBRATION_SAMPLES;
 imu_to_wheel_coordinate_offset_deg = normalize_angle_deg(0.0f - avg_filtered_raw_angle_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_coordinate_offset_deg, 2);
  Serial.print(F("Тестовый глобальный угол колеса с коррекцией: ")); Serial.println(normalize_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);
}
```