Титульный

Задание (1)

Задание (2)

Календарный график

РЕФЕРАТ

Выпускная квалификационная работа по теме «Разработка двухколесного самобалансирующегося робота» 109 страниц, 4 раздела, 44 рисунка, 16 источников, 6 приложений.

САМОБАЛАНСИРУЮЩИЙСЯ РОБОТ, РОБОТОТЕХНИКА, ПИД-РЕГУЛЯТОР, ARDUINO, ДВУХКОЛЕСНЫЙ РОБОТ

Цель работы: разработать двухколесного самобалансирующегося робота

В роли объекта исследования выступают двухколесные балансирующие конструкции.

В процессе работы проанализированы различные методы достижения устойчивости балансирующих конструкций; изучены принципы настройки ПИД регулятора.

Результатом работы является двухколесный самобалансирующийся робот.

СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

Arduino - торговая марка аппаратно-программных средств построения и прототипирования простых систем, моделей и экспериментов в области электроники, автоматики, автоматизации процессов и робототехники.

Bluetooth - беспроводной стандарт связи, основанный на радиочастотной технологии, который обеспечивает надежную и энергоэффективную передачу данных между различными электронными устройствами.

I2C - последовательная асимметричная шина для связи между интегральными схемами внутри электронных приборов.

USB - универсальная последовательная шина, стандарт связи, предназначенный для передачи данных и электропитания между компьютером и периферийными устройствами.

ПИД регулятор - алгоритм автоматической регулировки, использующийся для поддержания заданного уровня выходной переменной в системе с обратной связью.

Принципиальная схема - рисунок, служащий для передачи с помощью условных графических и буквенно-цифровых обозначений связей между элементами электрического устройства.

Функциональная схема - рисунок, разъясняющий процессы, протекающие в отдельных функциональных цепях изделия или изделия в целом.

Энкодер - устройство для измерения угла вращения.

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

GND – Ground – заземление.

SCL – Serial Clock Line - линия последовательного тактового сигнала.

SDA - Serial Data Line - линия последовательных данных.

USB - Universal Serial Bus - универсальная последовательная шина.

VCC - Voltage at the Common Collector – напряжение на общем коллекторе.

ПИД регулятор – Пропорционально-Интегрально-Дифференцирующий регулятор.

ШИМ – Широтно-Импульсная Модуляция.

ВВЕДЕНИЕ

Развитие робототехники и автоматизации открывает новые возможности в различных областях, включая промышленность, медицину и бытовую сферу. Одной из перспективных областей в робототехнике является разработка самобалансирующихся роботов. В данной дипломной работе исследуется разработка двухколесного самобалансирующегося робота с использованием управляющего модуля Arduino Uno, датчика MPU6050, а также ПИД регулятора в качестве основного алгоритма балансировки.

Актуальность данной работы обусловлена ростом интереса к робототехнике и возрастающим спросом на автономные системы, способные поддерживать равновесие. Двухколесные самобалансирующиеся роботы находят свое применение в различных областях, включая роботизированную медицину, автономную навигацию и помощников в повседневных задачах. Разработка такого робота с использованием доступных и надежных компонентов, таких как Arduino Uno и MPU6050, представляет интерес как с технической, так и с практической точки зрения.

Объектом исследования являются самобалансирующиеся конструкции.

Предметом исследования являются методы и алгоритмы разработки и настройки двухколесного самобалансирующегося робота с использованием управляющего модуля Arduino Uno, датчика MPU6050, ПИД регулятора.

Целью данной дипломной работы является разработка и реализация двухколесного самобалансирующегося робота на базе Arduino Uno и MPU6050. Главная цель состоит в создании стабильной системы балансировки робота, способной поддерживать равновесие на двух колесах.

Для достижения поставленной цели в работе решаются следующие задачи:

Произвести анализ предметной области, изучить основные принципы работы двухколесных самобалансирующихся роботов и их применение в различных сферах.

Обосновать выбор управляющего модуля Arduino Uno и главного датчика MPU6050 для разработки робота, рассмотреть их технические характеристики и возможности.

Разработать и настроить ПИД-регулятор для стабилизации равновесия робота на основе данных, получаемых от датчика MPU6050.

Реализовать программное обеспечение для управления роботом, включая алгоритмы балансировки и взаимодействия с окружающей средой.

Протестировать разработанный робот и оценить его производительность и стабильность при различных условиях.

Таким образом, данная работа имеет практическое значение, поскольку предоставляет новые знания и опыт в области разработки и реализации двухколесных самобалансирующихся роботов.

1 Исследовательский раздел

1.1 Двухколесные балансирующие роботы

Проектирование самобалансирующихся роботов начинается с выбора конструкции, типа двигателей и датчиков, которые будут использоваться для измерения угла наклона робота. Для балансирующих роботов наиболее распространены двухколесные конструкции, так как они позволяют достичь высокой маневренности и скорости.

Самобалансирующиеся конструкции обычно состоят из двух колес, которые расположены на одной оси и приводятся в движение двигателями. Для измерения угла наклона робота могут использоваться, например, акселерометры или гироскопы. Акселерометры измеряют ускорение робота в направлении оси X и Y, а гироскопы измеряют угловую скорость вокруг оси Z. Это позволяет такой конструкции ориентироваться в пространстве. В целом – проектирование балансирующих роботов — это довольно сложная и интересная инженерная задача, поскольку наблюдения за поведением таких конструкций, а также сами процессы программной настройки могут привести к неожиданным выводам, практическая польза от осознания которых может оказать влияние на развитие в разработке более сложных, важных и дорогостоящих систем.

Одним из простейших примеров самобалансирующегося двухколесного робота является конструкция инженера Стива Хассенплуга – LegWay [1]. Он создал этого робота еще в 2003 году. В нем не используется ни гироскопа, ни акселерометра. В нем использованы лишь два оптических датчика расстояния EOPD (ElectroOptical Proximity Detector) от HiTechnic. Устройство лего-робота достаточно легко понять, взглянув на рисунок 1. Два датчика установлены таким образом, чтобы контролировать расстояние до поверхности, по которой движется робот. Если расстояние до поверхности под датчиками уменьшается или, наоборот, увеличивается, робот производит соответствующее выравнивающее движение при помощи своих двигателей.

Рисунок 1 – Робот LegWay

С периодом, который составляет 50 миллисекунд LegWay пытается пересчитать точку баланса, измеряя текущее расстояние и скорость двигателя. Для движения вперед LegWay фактически запускает двигатели назад, вызывая наклон, который он автоматически корректирует, двигаясь вперед. Когда один датчик находится над линией, он останавливает этот двигатель, и LegWay балансирует, используя только другой двигатель, заставляя его вращаться. Чтобы вращаться на месте, оба двигателя смещаются от центра в противоположные стороны на одинаковую величину, но они все еще корректируют наклон. Если двигатели работают на полной мощности (в любом направлении) более 1 секунды, LegWay предполагает, что он упал, и программа завершается.

Другой пример, в котором уже используется акселерометр – это робот VertiBot. Он довольно миниатюрный и обладает простой конструкцией, при этом используя для балансировки показатели акселерометра. Его корпус и колеса выполнены с помощью 3D-печати. Электронная начинка базируется на плате Arduino Nano [2].

Рисунок 2 – Робот VertiBot

Также в качестве примера всеми известной двухколесной балансирующей конструкции можно привет самокат SegWay (рисунок 3). Двухколесный электрический самокат SegWay, в настоящее время очень популярен. Гироскоп, снабженный двумя колесами, мотором, ручкой, и передовой системой самостоятельного балансирования позволяет передвигаться без особых усилий на скорости, намного превосходящей скорость движения среднестатистического пешехода. На таком электрическом самокате можно перемещаться по труднодоступным для автомобиля местам — лифтам, торговым центрам, складам, многолюдным улицам. Такая маневренность и скорость актуальны не только для частного использования — сегвеи активно задействуются на различных предприятиях, в охранных государственных структурах, торговых организациях, гостиничных комплексах, в местах проведения экскурсий.

Но стоит отметить, что такое устройство имеет два значительных минуса — относительно высокую стоимость (75000 рублей для модели, показанной на рисунке 3 [3]) и неэффективную адаптацию к природным условиям, позволяющую использовать сегвей в условиях нашей страны только летом.

Рисунок 3 - Электросамокат Segway-Ninebot MAX G2

Еще одна похожая конструкция от той же компании SegWay - гироскутер Segway-Ninebot mini S (рисунок 4). Такая конструкция может быть более удобной и компактной для переноски.

Рисунок 4 - Гироскутер Segway-Ninebot mini S

1.2 Моноколесо

Еще один вид балансирующего транспорта, популярного на сегодняшний день – моноколесо (рисунок 5). Моноколесо состоит из мотора-колеса и двигателя, управляющегося на основе показаний гироскопа. Максимальная скорость, до которой можно разогнаться на данном транспортном средстве — 20км/час. В городских условиях этого более, чем достаточно, даже довольного много. Отсутствие ручки позволяет управляющему моноколесом держать в руках планшет, зонтик или другие необходимые предметы. Кроме того, это самый компактный вид балансирующего автотранспорта в мире — моноколесо можно без труда взять в офис, общественный транспорт или положить в багажник. Перевозить и хранить устройство можно в любом месте. Стоит учесть, что освоения принципов управления моноколесом – гораздо более сложный процесс, нежели в случае с электрическим самокатом, поэтому такой вид транспорта подойдет не всем. Также такое транспортное средство от того же производителя отличается высокой стоимостью (85000 рублей [5]).

Рисунок 5 - Моноколесо Ninebot by Segway One Z6 574Wh

1.3 Редукторные моторы

Редукторные моторы — это электрические двигатели, оснащенные редуктором, который позволяет увеличить крутящий момент и снизить скорость вращения. Они широко используются в самобалансирующихся роботах, так как обладают высокой мощностью и точностью в управлении. В самобалансирующихся роботах используется два редукторных мотора, каждый из которых приводит в движение колесо или ролик на одной стороне робота. Это позволяет роботу двигаться в любом направлении, а также поддерживать баланс при изменении скорости и направления движения. Редукторные моторы имеют несколько преимуществ по сравнению с другими типами моторов.

В первую очередь, они обладают высоким крутящим моментом, что позволяет роботу легко перемещаться по неровной поверхности или подниматься на крутые склоны.

Кроме того, они достаточно компактны и легки, что делает их идеальным выбором для использования в мобильных роботах. Однако, редукторные моторы имеют и некоторые недостатки. Например, из-за наличия редуктора, они могут быть более шумными, чем другие типы моторов. Кроме того, редукторный механизм может быть подвержен износу при длительном использовании, что может привести к снижению производительности. В целом, редукторные моторы являются очень эффективным и популярным выбором для использования в самобалансирующихся роботах благодаря своей высокой мощности и точности управления. Однако, при выборе такого мотора необходимо учитывать его особенности и возможные недостатки, чтобы обеспечить оптимальную работу робота.

1.3.1 Мотор JGA25-370

JGA25-370 DC 6V 12V 24V мотор-редуктор представляет собой компактное устройство, состоящее из двигателя постоянного тока и редуктора с передаточным числом 1:25. Он обладает высоким крутящим моментом и низкой скоростью вращения, что делает его идеальным выбором для использования в самобалансирующихся роботах. Для использования JGA25-370 мотор-редуктора в самобалансирующемся роботе необходимо установить два таких мотор-редуктора на колеса робота. Каждый мотор-редуктор будет приводить в движение колесо на одной стороне робота. Если оба колеса будут вращаться с одинаковой скоростью, то робот будет двигаться прямо. Если одно колесо будет вращаться быстрее, чем другое, то робот начнет поворачивать в соответствующую сторону. Одна из особенностей JGA25-370 мотор-редуктора заключается в том, что он имеет низкий коэффициент шума и вибраций, что является важным критерием для использования в самобалансирующихся роботах. Также для управления мотор-редукторами JGA25-370 можно использовать контроллер двигателя постоянного тока, который будет регулировать напряжение питания мотора и следить за скоростью вращения. Это обеспечит точное и мягкое управление роботом и его балансирование. В целом, JGA25-370 DC 6V 12V 24V мотор-редуктор является надежным и эффективным устройством для использования в самобалансирующихся роботах благодаря своей высокой мощности, точности управления и низкому уровню шума и вибраций.

Рисунок 6 – Внешний вид мотора-редуктора JGA25-370

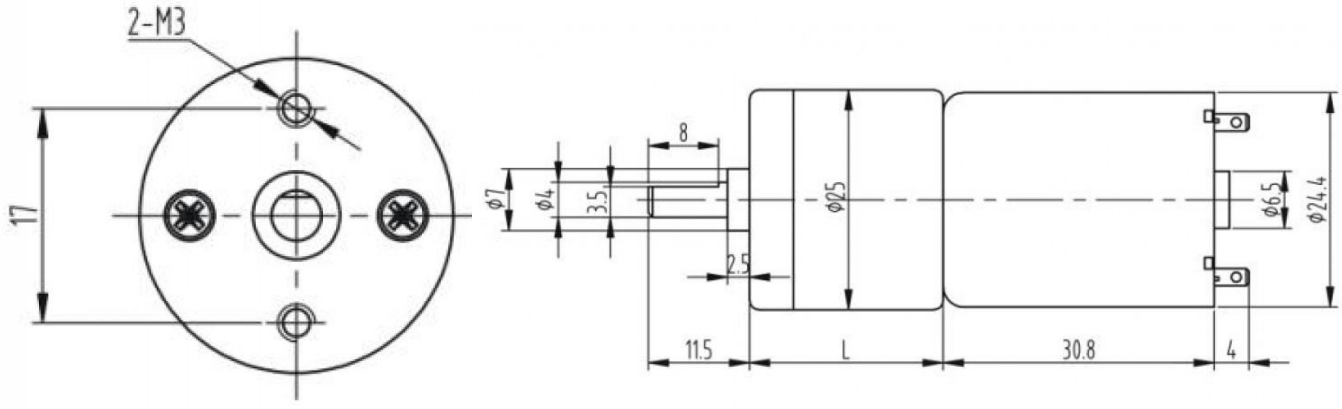

Рисунок 7 – Габариты мотора-редуктора JGA25-370

1.3.2 Мотор 6V12V24V 25GA370

Микродвигатель постоянного тока 6V12V24V 25GA370 — это компактное устройство, состоящее из двигателя постоянного тока и редуктора с передаточным числом 1:25. Он обладает высоким крутящим моментом и низкой скоростью вращения, что делает его идеальным выбором для использования в различных устройствах, таких как роботы, игрушки, автоматические двери и другие. Данный двигатель может работать при напряжении питания от 6 до 24 Вольт, что позволяет использовать его как в небольших устройствах, так и в более мощных системах. Он имеет диаметр корпуса 25 мм и длину 52 мм, что делает его очень компактным и удобным для установки в ограниченных пространствах. Одна из особенностей микродвигателя постоянного тока 6V12V24V 25GA370 заключается в том, что он имеет низкий уровень шума и вибраций, что является важным критерием при использовании в устройствах, где требуется минимизировать шум. Кроме того, данный двигатель обладает высокой мощностью и крутящим моментом, что позволяет использовать его в устройствах с повышенными требованиями к мощности, например, в роботах или автоматических дверях. Для управления Микродвигателем постоянного тока 6V12V24V 25GA370 можно использовать контроллер двигателя постоянного тока, который будет регулировать напряжение питания двигателя и следить за скоростью вращения. Это обеспечит точное и мягкое управление устройством, где используется данный двигатель. В целом, Микродвигатель постоянного тока 6V12V24V 25GA370 является надежным и эффективным устройством, который можно использовать в различных приложениях благодаря своей компактности, мощности, низкому уровню шума и высокой точности управления. В связи с его доступностью и относительно низкой стоимостью в качестве двигателя, используемого в роботе будет взят именно он.

Рисунок 8 – Микродвигатель постоянного тока 6V12V24V 25GA370

Выбор двигателей осуществляется на основе многих параметров различной степени важности. Самое главное и основное – скорость – максимальные обороты в минуту, а также возможность вращать ими колеса подходящего размера. Были выбраны двигатели 25GA370, купленные в комплекте с колесами. Их главный минус – отсутствие прямой обратной связи в следствии отсутствия датчиков оборотов. Но на основе проведенных исследований предполагается, робот будет отлично балансировать на основе показаний акселерометра-гироскопа MPU6050 даже без использования датчиков оборотов. Габариты двигателя показаны на рисунке 9.

Данный двигатель в комплекте с колесами обладает подходящими габаритами, достаточной скоростью, а также при своей низкой стоимости (~312 рублей) является отличным выбором для этого проекта.

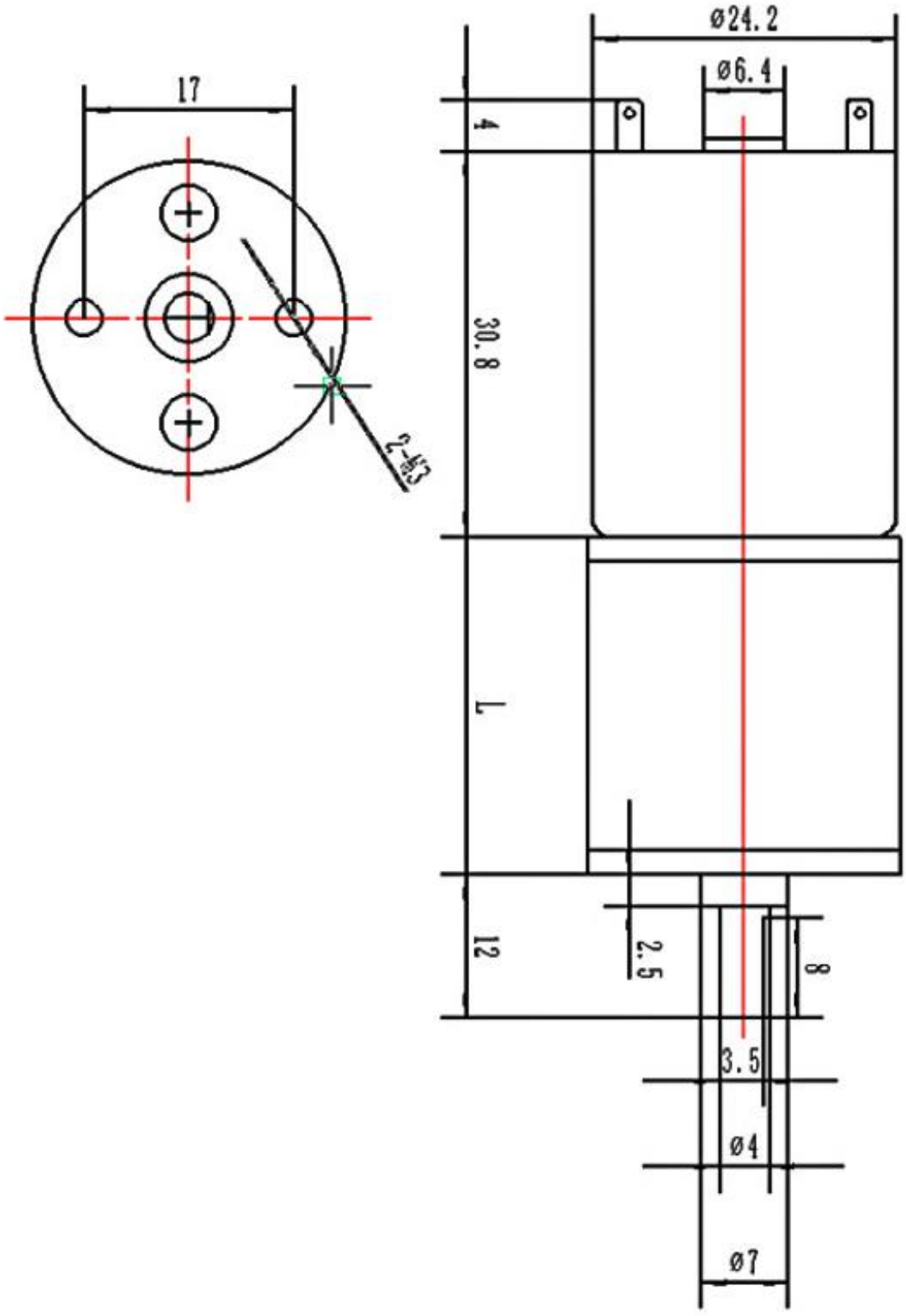

Рисунок 9 – Габариты двигателя 25GA370

1.4 Датчики

1.4.1 Акселерометр

Акселерометр — это электронный датчик, который используется для измерения ускорения объекта в пространстве. Он может показывать как линейное ускорение (направленное вдоль оси), так и гравитационное ускорение (направленное вниз). Эта информация может быть использована для определения ориентации объекта в пространстве.

Акселерометры могут использоваться в самобалансирующихся роботах, чтобы измерять ускорения, вызванные гравитацией и движением робота. Они могут измерять ускорение в трех осях: вдоль оси X (направление вперед/назад), вдоль оси Y (направление влево/вправо) и вдоль оси Z (направление вверх/вниз).

Когда самобалансирующийся робот находится в вертикальном положении, акселерометр измеряет только ускорение, вызванное гравитацией, которое направлено вдоль оси Z. Если робот начинает наклоняться вперед или назад, акселерометр также измеряет ускорение, вызванное движением робота в направлении этой оси. Аналогично, когда робот наклоняется влево или вправо, акселерометр измеряет ускорение, вызванное движением робота в направлении оси Y.

После того как акселерометр измеряет ускорения в трех осях, эта информация используется для определения угла наклона робота. Робот может использовать эти данные для корректировки своего положения и управления двигателями, чтобы сохранять равновесие.

1.4.2 Гироскоп

Гироскоп — это устройство, способное измерять угловую скорость вращения вокруг определенной оси. Он основан на принципе сохранения углового момента, что означает, что гироскоп будет сохранять свою ориентацию в пространстве при отсутствии внешних моментов. В основе работы гироскопа лежит явление, известное как гироскопическая прецессия. Гироскопическая прецессия происходит, когда на гироскоп действует момент силы, перпендикулярный его оси вращения. В результате этого действия ось вращения гироскопа начинает перемещаться в направлении, перпендикулярном к направлению приложения силы. Это приводит к изменению ориентации гироскопа в пространстве. Контроллер анализирует данные от гироскопов и генерирует управляющие сигналы для регулирования скорости вращения колес. Например, если робот начинает наклоняться вперед, гироскопы замечают угловую скорость изменения и передают эту информацию контроллеру, который в ответ увеличивает скорость вращения заднего колеса, чтобы сбалансировать робота и вернуть его в вертикальное положение. Гироскопы также могут использоваться для управления поворотами робота. Они определяют угловую скорость поворота и передают эту информацию контроллеру, который регулирует скорость вращения колес в соответствии с требуемым поворотом. Выводы Использование гироскопов в самобалансирующихся роботах является важным аспектом для достижения вертикальной стабильности и управления движением. Гироскопы обеспечивают информацию о текущем положении и угловой скорости робота, позволяя ему мгновенно реагировать на изменения и поддерживать равновесие. Это делает гироскопы неотъемлемой частью систем самобалансирующихся двухколесных роботов и открывает возможности для разработки более эффективных и точных систем управления.

1.4.3 MPU6050

MPU6050 — это инерционный измеритель, который сочетает в себе гироскоп (датчик угловой скорости) и акселерометр (датчик ускорения). Он может использоваться для измерения ориентации объекта в трех измерениях. В самобалансирующихся роботах MPU6050 часто используется для измерения угла наклона платформы или корпуса робота. Эта информация затем используется для управления двигателями и поддержания равновесия. Как правило, MPU6050 подключается к микроконтроллеру, который анализирует данные, полученные от датчика, и вычисляет необходимые значения для управления двигателями. Это может быть достигнуто с помощью PID-регулятора, который вычисляет ошибку (разницу между желаемым углом наклона и текущим углом) и рассчитывает необходимую скорость двигателей для компенсации этой ошибки. Возможности MPU6050 можно расширить, добавив другие датчики, такие как магнитометр, для улучшения точности ориентации робота.

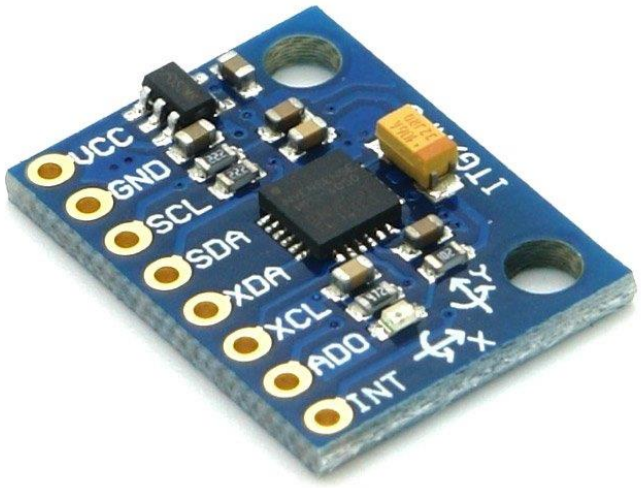

Рисунок 10 – Внешний вид MPU6050

1.4.4 Магнитометр

Магнитометр — это датчик магнитного поля, который измеряет магнитное поле в трех измерениях. В сочетании с гироскопом и акселерометром, магнитометр может использоваться для получения более точной информации об ориентации объекта в пространстве. В самобалансирующихся роботах, например, магнитометр может помочь улучшить точность определения угла наклона корпуса робота. Это особенно полезно в условиях, когда акселерометр может давать неточные результаты из-за вибраций или движений на не стабильной поверхности. К примеру, если робот двигается по местности, где есть электромагнитные помехи (например, от проводов), то магнитометр может помочь корректировать данные, полученные от других датчиков, чтобы получить более точную информацию об ориентации робота в пространстве. Результаты измерений магнитометра могут быть использованы для вычисления магнитного курса робота, что также может быть полезно при навигации или перемещении робота в заданном направлении. Но после изучения принципов его работы более подробно стало очевидно, что применение магнитометра в системах этого проекта нецелесообразно.

1.5 Выбор системы регулирования

Регуляторы используются для управления системами и обеспечения стабильности процессов. Они могут быть использованы в широком спектре приложений, от самобалансирующихся роботов до регулирования температуры в промышленных процессах. Существует несколько различных типов регуляторов, каждый из которых имеет свои особенности и применяется в зависимости от конкретной задачи. Регулятор, или управляющее устройство, — в теории автоматического управления устройство, которое следит за состоянием объекта управления как системы и вырабатывает для неё управляющие сигналы. Регулятор следит за изменением некоторых параметров объекта управления (непосредственно либо с помощью наблюдателей) и реагируют на их изменение с помощью некоторых воздействий в соответствии с заданным качеством управления.

1.5.1 Пропорциональный регулятор (P-регулятор)

Пропорциональный регулятор (P-регулятор) — это один из наиболее простых и широко используемых алгоритмов управления в автоматическом регулировании. Он относится к классу простейших алгоритмов управления, которые имеют линейную функцию передачи.

Суть работы P-регулятора заключается в следующем. Регулятор получает на вход желаемое значение (задание) и текущее значение переменной, которую нужно контролировать. Разность между этими значениями называется ошибкой регулирования. Затем регулятор вычисляет управляющий сигнал, который пропорционален ошибке.

Математическая формула для работы P-регулятора выглядит так:

$$u(t) = Kp * e(t) \quad (1)$$

где u(t) - управляющий сигнал, Kp - коэффициент пропорциональности (параметр регулятора), e(t) - ошибка регулирования.

Чем больше значение ошибки, тем больше будет значение управляющего сигнала. Параметр Kp выбирается экспериментально и определяет чувствительность регулятора к ошибке. Чем больше Kp, тем быстрее будет достигнуто желаемое значение, но при этом может возникнуть перерегулирование (переброс системы), что может привести к еще более большим отклонениям.

Преимущества P-регулятора заключаются в его простоте и надежности. Он легко настраивается и может использоваться для широкого диапазона задач. Однако, он не всегда способен обеспечить точное регулирование системы, так как не учитывает интегральную и производную составляющие ошибки регулирования (I и D, так как не учитывает интегральную и производную составляющие ошибки регулирования (I и D).

1.5.2 Интегральный регулятор (I-регулятор)

Интегральный регулятор (I-регулятор) — это один из алгоритмов управления в автоматическом регулировании, который используется для устранения статических ошибок регулирования. Он использует интегральную составляющую ошибки регулирования для вычисления управляющего сигнала.

Принцип работы I-регулятора заключается в накоплении ошибки регулирования со временем и использовании этой накопленной ошибки для корректировки управляющего сигнала. При этом, чем больше накопленная ошибка, тем больше значение управляющего сигнала.

Математическая формула для работы I-регулятора выглядит так:

$$u(t) = Kp * \int e(t)dt \quad (2)$$

где u(t) - управляющий сигнал, Kp - коэффициент пропорциональности (параметр регулятора), e(t) - ошибка регулирования, $\int$ - оператор интегрирования.

I-регулятор может быть полезен в случаях, когда пропорциональный регулятор (P-регулятор) не может полностью устранить ошибку регулирования.

Например, если возникает постоянная ошибка из-за внешних факторов, таких как дрейф, шум или нелинейность системы.

Преимущества I-регулятора заключаются в его способности устранять статические ошибки регулирования и обеспечивать точное управление системой. Однако, его использование может привести к сильной инерции системы, если коэффициент пропорциональности Kp выбран слишком большим значением. Кроме того, I-регулятор может быть чувствителен к наличию высокочастотного шума в системе.

1.5.3 Дифференциальный регулятор (D-регулятор)

Дифференциальный регулятор (D-регулятор) — это один из алгоритмов управления в автоматическом регулировании, который использует производную составляющую ошибки регулирования для вычисления управляющего сигнала. Он позволяет более точно и быстро реагировать на изменения задания и исключить перерегулирование системы.

Принцип работы D-регулятора заключается в учете скорости изменения ошибки регулирования. При этом, чем быстрее меняется ошибка, тем больше значение управляющего сигнала.

Математическая формула для работы D-регулятора выглядит так:

$$u(t) = Kp * de(t)/dt \quad (3)$$

где u(t) - управляющий сигнал, Kp - коэффициент пропорциональности (параметр регулятора), e(t) - ошибка регулирования, de(t)/dt - производная ошибки регулирования по времени.

D-регулятор может быть полезным в случаях, когда система имеет долгое время инерции или нелинейность и требуется более точное управление. Также он может помочь избежать перерегулирования системы при быстром изменении задания.

Преимущества D-регулятора заключаются в его способности уменьшать перерегулирование и обеспечивать более точное управление системой. Однако,

его использование может привести к повышенной чувствительности к шумам, особенно высокочастотным, что может снизить стабильность системы. Кроме того, при неправильном выборе коэффициента пропорциональности Kp, D-регулятор может усилить колебания системы.

1.5.4 Пропорционально-интегрально-дифференциальный регулятор (PID-регулятор)

Пропорционально-интегрально-дифференциальный регулятор (PID-регулятор) – это алгоритм управления, который позволяет регулировать значение выходной переменной системы таким образом, чтобы она максимально точно соответствовала заданному значению. PID-регулятор является наиболее распространенным и эффективным методом автоматического управления.

PID-регулятор состоит из трех компонентов: пропорциональной, интегральной и дифференциальной составляющих. Каждая из них выполняет свою функцию в процессе регулирования выходной переменной.

Пропорциональная составляющая основывается на принципе пропорциональности между ошибкой регулирования и выходным сигналом. Ошибка регулирования – это разница между заданным значением и фактическим значением выходной переменной. Чем больше ошибка, тем больше должен быть выходной сигнал, чтобы скорректировать ее. Пропорциональная составляющая умножает ошибку на коэффициент пропорциональности Kp, который определяет, насколько сильно должен изменяться выходной сигнал в зависимости от ошибки. Чем больше значение Kp, тем более чувствительной будет система к ошибкам.

Интегральная составляющая учитывает накопленную ошибку регулирования за время работы системы. Она интегрирует ошибку по времени и умножает на коэффициент интегрирования Ki. Это позволяет устранять постоянную ошибку регулирования, которая может возникнуть при использовании только пропорциональной составляющей.

Дифференциальная составляющая учитывает скорость изменения ошибки регулирования. Она дифференцирует ошибку по времени и умножает на коэффициент дифференцирования Kd. Это позволяет предотвратить быстрое изменение выходной переменной, что может привести к нестабильности системы.

Все три составляющие объединяются в единую формулу для расчета выходного сигнала:

$$u(t) = Kp*e(t) + Ki*\int e(t)dt + Kd*(de(t)/dt) \quad (3)$$

где u(t) – выходной сигнал, e(t) – ошибка регулирования, t – время, Kp, Ki, Kd – коэффициенты пропорциональной, интегральной и дифференциальной составляющих соответственно.

PID-регулятор может быть настроен для достижения оптимальной производительности системы. Однако, неправильная настройка коэффициентов может привести к нестабильности или медленной реакции системы. Поэтому важно проводить тщательную настройку PID-регулятора для каждой конкретной системы.

О качестве регулятора судят по реакции системы на единичное воздействие – специальную функцию, скачком принимающую значение, равное

1. И тогда мы получим то, что называется графиком переходного процесса:

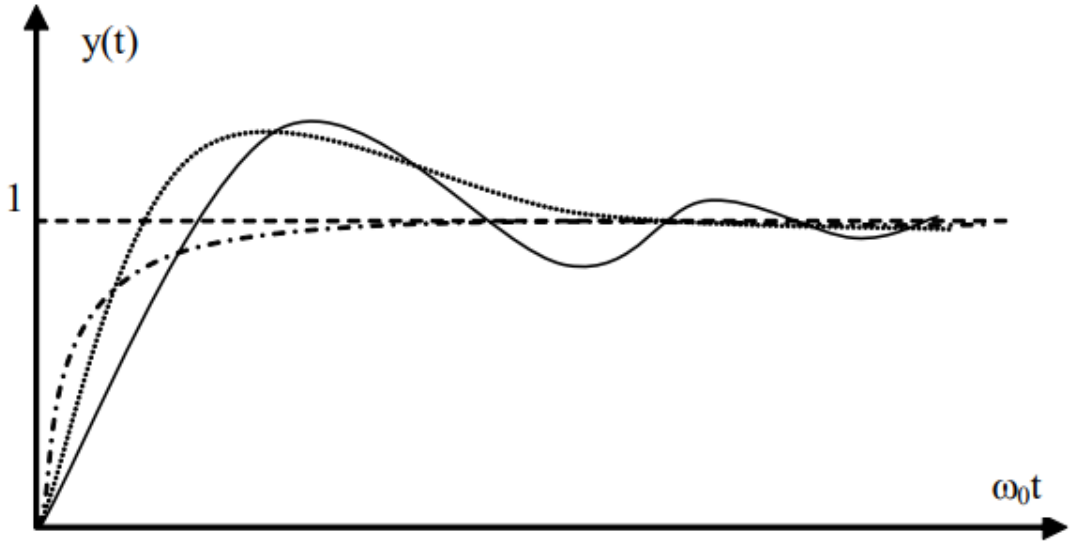

Рисунок 11 - Реакция системы на единичное воздействие

В заключении исследовательского раздела, основываясь на проведенном анализе двухколесных самобалансирующихся роботов, с применением датчиков акселерометра и гироскопа, можно сделать вывод о преимуществах применения ПИД-регулятора для поддержания равновесия и стабильности таких систем.

ПИД-регулятор является широко применяемым методом автоматического управления, используемым для регулирования различных процессов. Он состоит из трех компонентов: пропорциональной, интегральной и дифференциальной составляющих. Каждая из этих составляющих играет свою роль в поддержании требуемого управляющего сигнала, и совместно они обеспечивают стабильное и точное регулирование.

Преимущества применения ПИД-регулятора для двухколесного самобалансирующегося робота, следующие:

Устойчивость: ПИД-регулятор обеспечивает устойчивость системы путем быстрой коррекции ошибки между желаемым и фактическим положением робота. Благодаря интегральной составляющей, регулятор компенсирует постоянную ошибку и предотвращает накопление ошибки со временем.

Точность: ПИД-регулятор обладает высокой точностью регулирования благодаря использованию дифференциальной составляющей, которая реагирует на изменения ошибки в текущем моменте времени. Это позволяет быстро и точно реагировать на внешние возмущения и изменения в положении робота.

Адаптивность: ПИД-регулятор является адаптивным и способен быстро реагировать на изменения в условиях работы. Он может автоматически настраивать коэффициенты пропорциональной, интегральной и дифференциальной составляющих, чтобы поддерживать оптимальное регулирование в различных условиях.

Простота настройки: ПИД-регулятор относительно прост в настройке и настраиваемых параметрах. Хотя процесс настройки может требовать некоторого опыта и экспертизы, существуют методы и алгоритмы, которые помогают определить оптимальные значения коэффициентов ПИД-регулятора для конкретной системы.

Исходя из вышеизложенного, можно заключить, что ПИД-регулятор является идеальным выбором для двухколесного самобалансирующегося робота. Его устойчивость, точность, адаптивность и относительная простота настройки делают его эффективным инструментом для поддержания равновесия и обеспечения стабильной работы робота на основе данных от датчиков акселерометра и гироскопа.

В качестве двигателя был выбран мотор 6V12V24V 25GA370, поскольку он был доступен для покупки, имел относительно низкую стоимость, а также все его характеристики подходят для удовлетворения потребностей проектируемой системы. В качестве основного датчика, за счет показаний которого система сможет находиться в состоянии равновесия был выбран гироскоп-акселерометр MPU6050, поскольку подобный тип датчик используется во многих, проверенных временем балансирующих системах, а также он доступен для покупки и обладает относительно низкой стоимостью.

2 Конструкторский раздел

2.1 Разработка платформы

Проектирование и разработка платформы для двухколесного самобалансирующегося робота являются сложной и многогранной задачей, требующей тщательной проработки всех аспектов процесса.

Одним из первых шагов в данном проекте является определение основных характеристик робота, таких как масса, скорость, габариты и допустимая нагрузка. На основе этих данных необходимо выбрать подходящую платформу, которая обеспечит выполнение поставленных задач.

Важным этапом является проектирование механической части робота, которое должно учитывать требования к устойчивости и маневренности с учетом выбранных компонентов. В процессе проектирования механической части необходимо определить расположение моторов, колес, батарей и других компонентов системы.

Для успешной разработки платформы для двухколесного самобалансирующегося робота необходимо учитывать множество факторов и параметров, которые могут влиять на его работу. Это требует детального изучения технических и технологических аспектов проекта.

Очень важно выбрать подходящий под все требования конструкции материал. Свойства материалов сильно различаются, поэтому выбор того, который будет использоваться должен происходить ответственно.

Для создания подобных роботов необходимо использовать особые материалы, которые обладают определенными свойствами, такими как легкость, прочность и устойчивость.

Необходимо на основе необходимых требований к физическим показателям произвести анализ и выбрать подходящий материал, из которого будет изготавливаться конструкция.

В случае с самобалансирующимся двухколесным роботом необходимо, чтобы конструкция обладала относительно небольшим весом. В этом случае также важнее гибкость, нежели ударопрочность.

Для концептуального дизайна также необходимо, чтобы материал был прозрачным и было лучше видно микросхемы. Также это обеспечивает удобство использования, подключения, отслеживания целостности конструкции. Это, помимо всего, позволяет создать довольно эффектный дизайн.

В целом органическое стекло обладает достаточно высокой прочностью, это позволяет создавать платформы для роботов с большой нагрузкой без риска повреждений и разрушений. Органическое стекло отличается легкостью, что упрощает транспортировку и монтаж роботов, а также настройку PID регулятора и всей системы в целом.

Роботы, особенно те, которые работают на неровной поверхности, часто испытывают вибрации. Органическое стекло обладает устойчивостью к вибрациям, что уменьшает вероятность сбоев в работе робота.

Органическое стекло устойчиво к коррозии, что гарантирует долгий срок службы платформы и робота в целом.

Все эти преимущества делают органическое стекло идеальным материалом для создания платформы самобалансирующегося робота. Оно обеспечивает прочность, легкость и устойчивость к вибрациям, а также может быть использовано для создания эффектного дизайна. Кроме того, органическое стекло имеет высокую устойчивость к коррозии, что гарантирует долгий срок службы робота. Также пластины из органического стекла были доступны на складе. Поэтому платформа изготавливается из него.

Был произведен расчет площади поверхности необходимых пластин на основе размеров платы микроконтроллера, микродвигателей постоянного тока 6V12V24V 25GA370, а также примерного расположения датчиков.

Из органического стекла было вырезано две пластины, которые соответствуют двум «этажам» робота. Также был произведен расчет необходимого для подключения всех устройств количества отверстий. Шаблоны резки и сверловки показаны на рисунках 12 и 13.

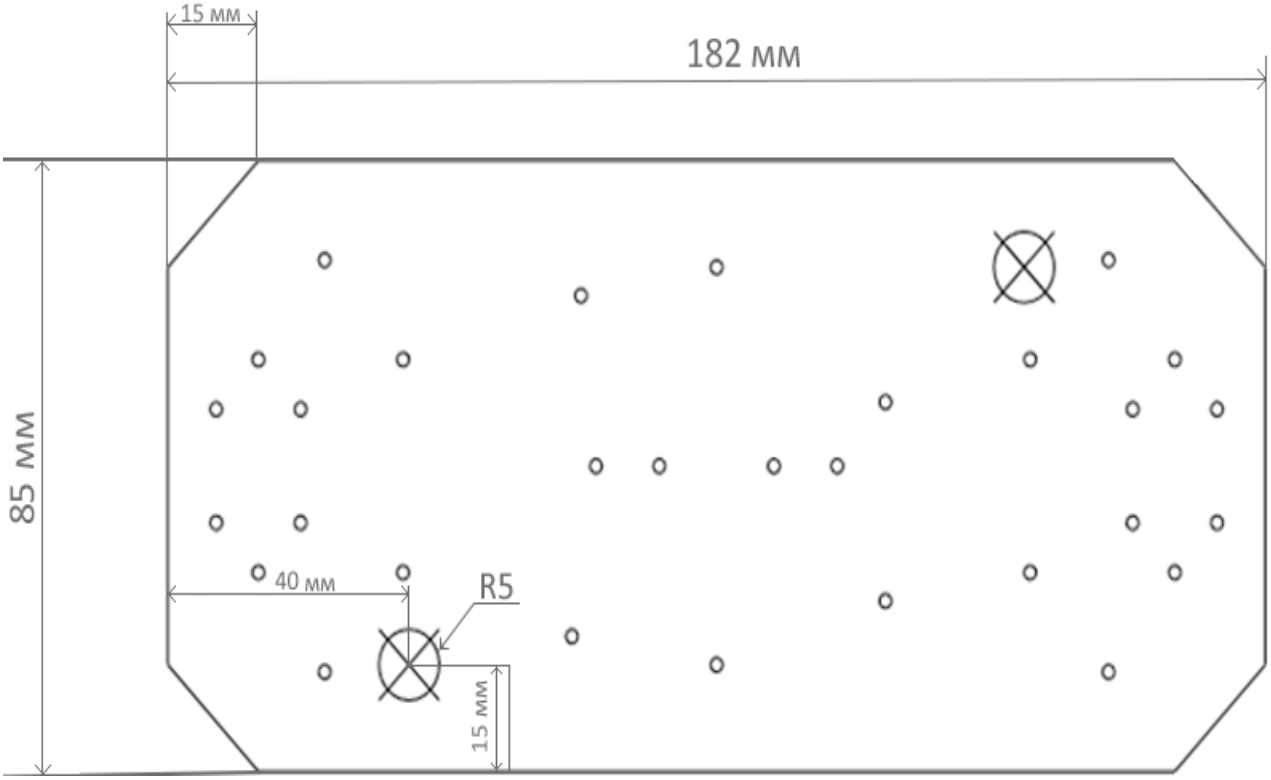

Рисунок 12 – Шаблон резки и сверловки платформы первого этажа

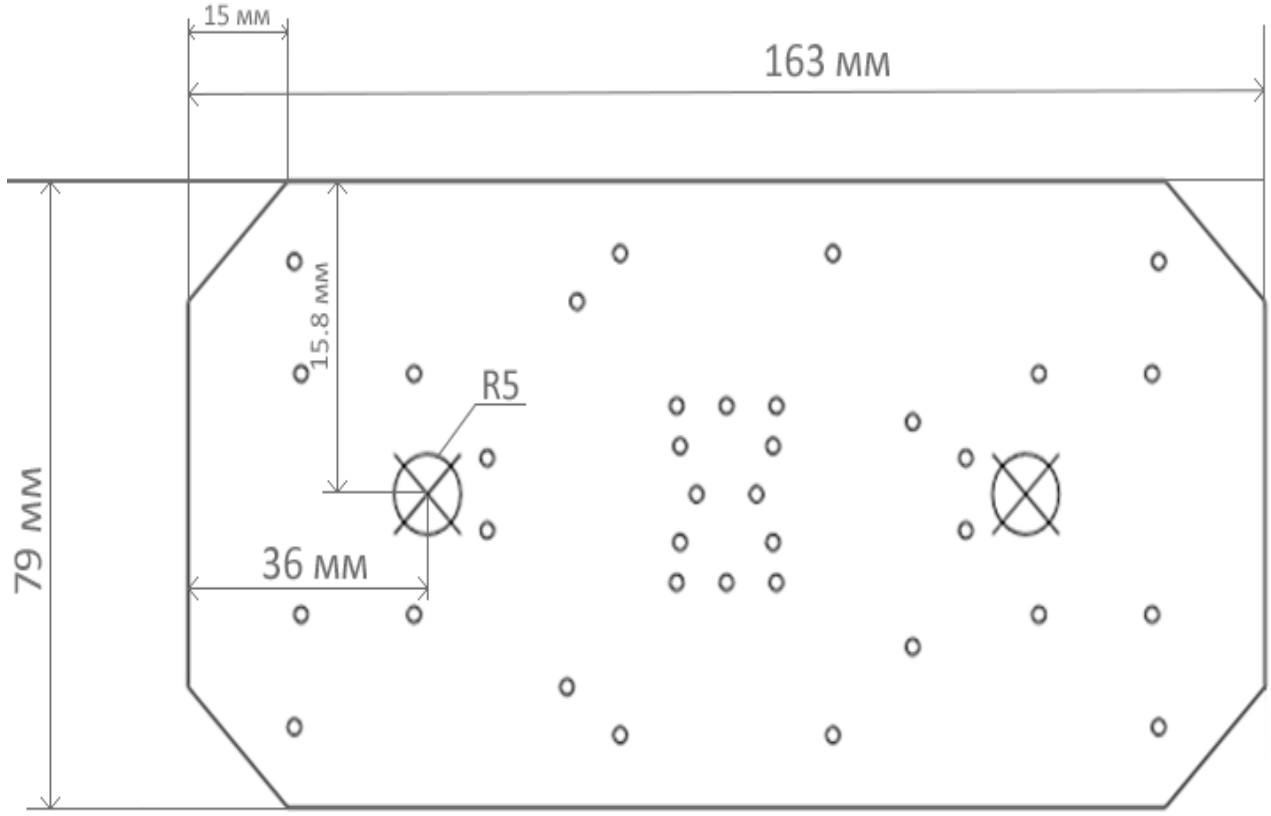

Рисунок 13 – Шаблон резки и сверловки платформы второго этажа

Внешний вид изготовленных платформ показан на рисунках 14 и 15 соответственно.

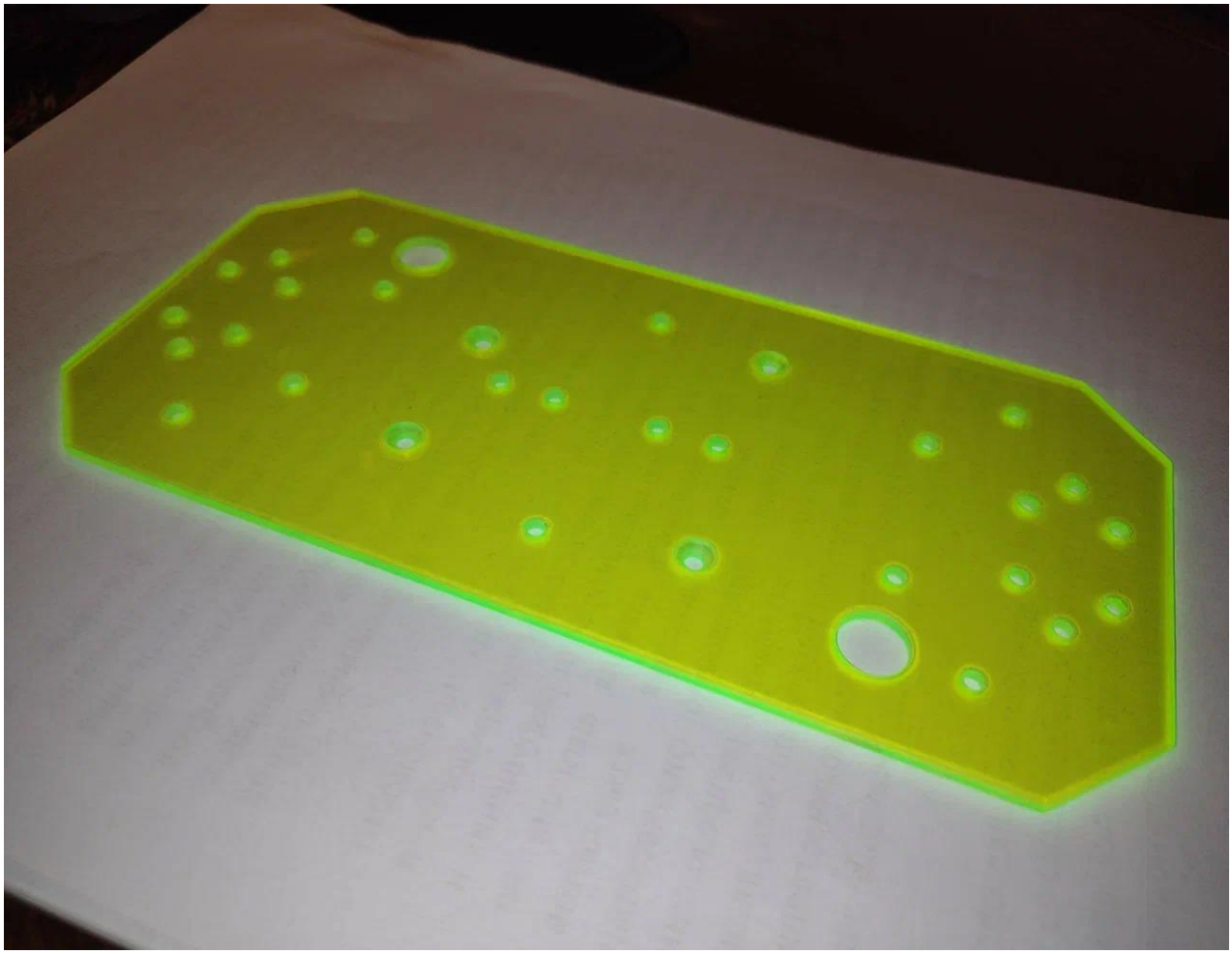

Рисунок 14 – Внешний вид изготовленной платформы первого этажа

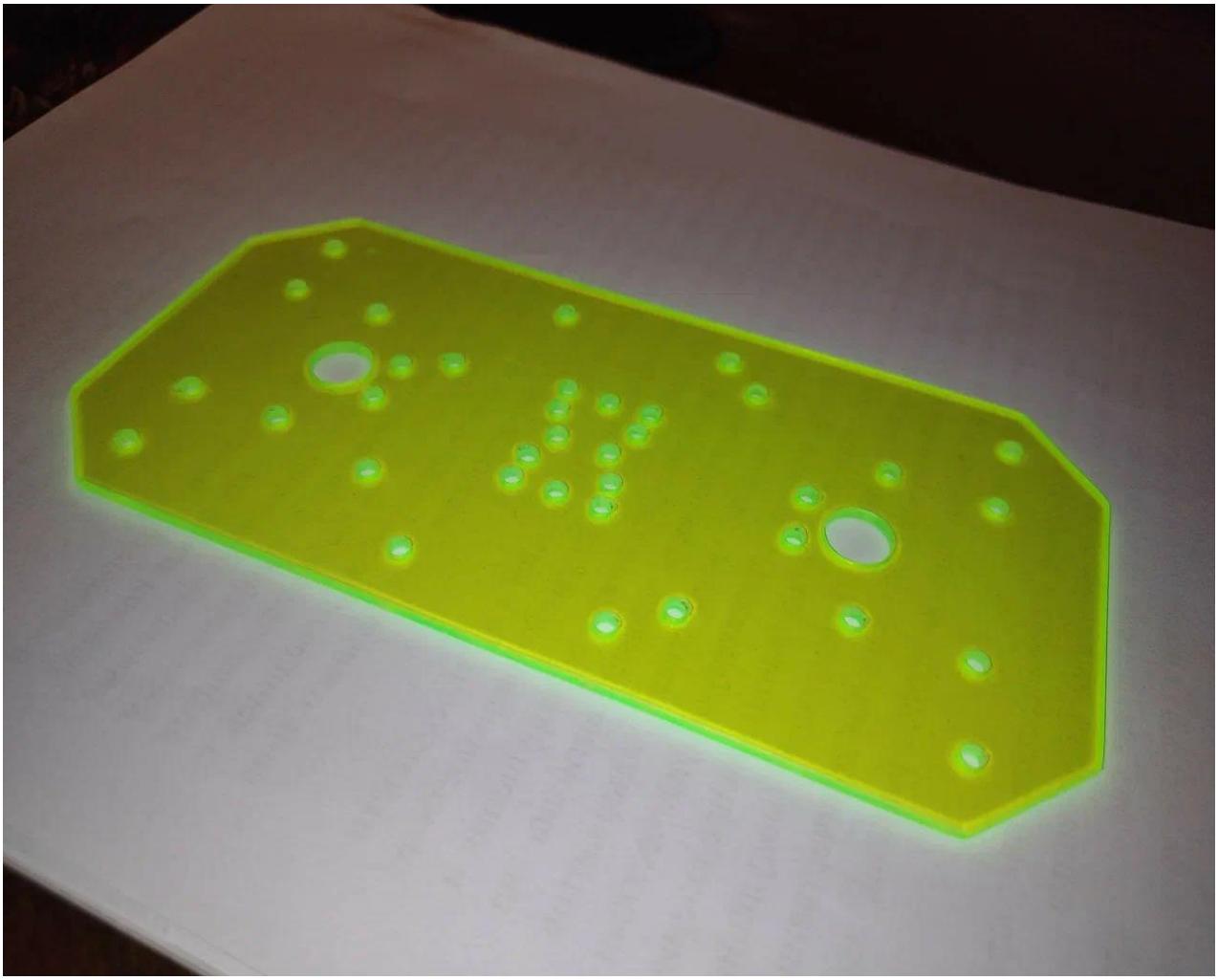

Рисунок 15 – Внешний вид изготовленной платформы второго этажа

Органическое стекло этого цвета обладает особым желтым свечением из-за своих особенностей пропускания света. Дизайн получился довольно особенным с точки зрения функций, которые он может выполнять, а также эстетически привлекательным. Внешний вид платформ, установленных в виде двухэтажной конструкции показан на рисунке 16.

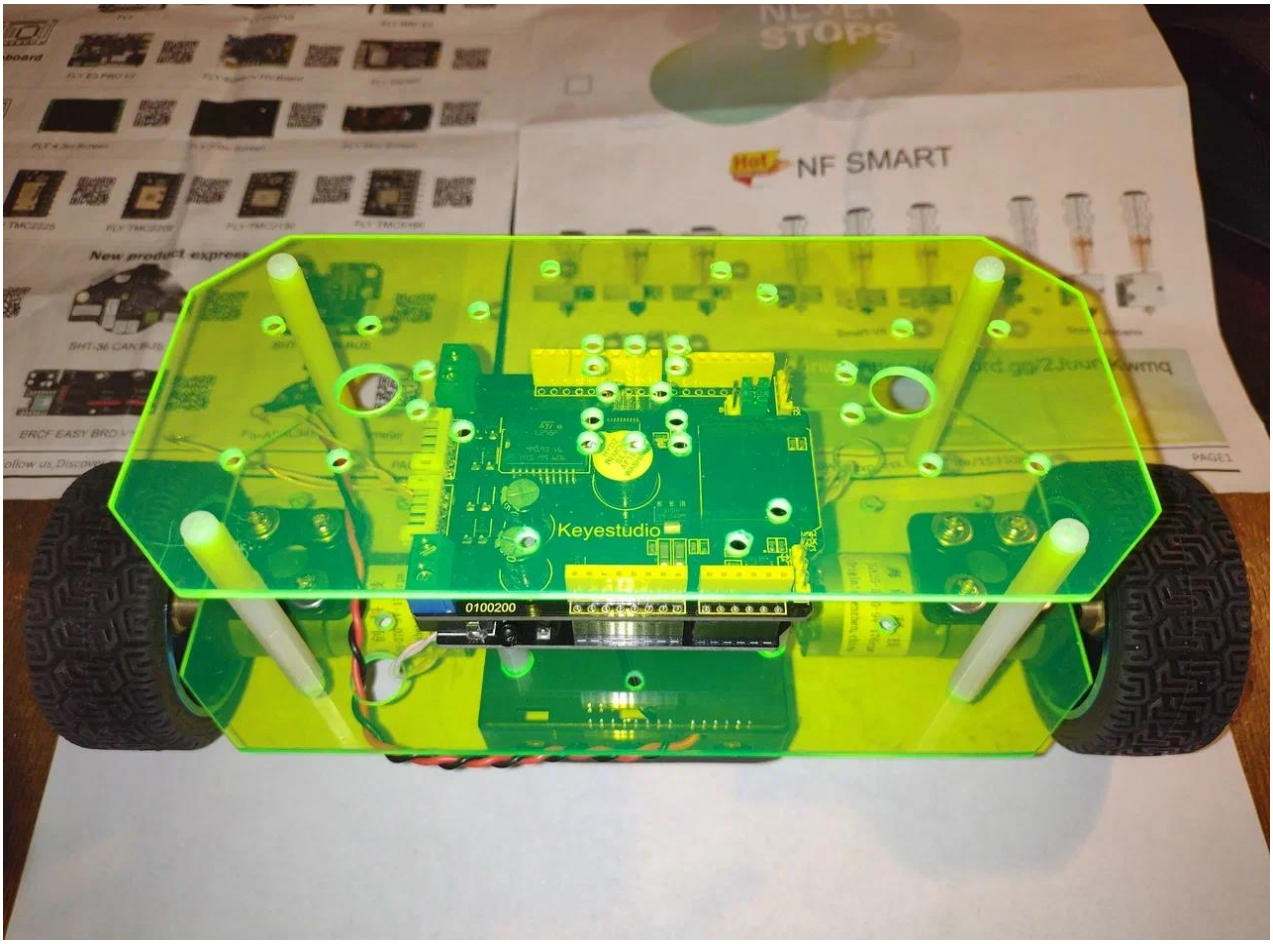

Рисунок 16 – Внешний вид собранной двухэтажной конструкции

2.2 Разработка электрической структурной схемы

В процессе работы для создания двухколесного самобалансирующегося робота была создана электрическая структурная схема, в которой отображены следующие элементы (рисунок 17):

Микроконтроллер Arduino UNO: Этот микроконтроллер является центральным устройством управления и обработки данных. Он подключается ко всем остальным компонентам и обеспечивает контроль и координацию работы системы.

Гироскоп-акселерометр MPU6050: Данный модуль используется для измерения угловых скоростей и ускорений робота. Он предоставляет необходимую информацию для определения текущего положения и ориентации робота.

Двигатели, использующие полномостовой драйвер коллекторных моторов L298P: Двигатели отвечают за приведение колес робота в движение. Драйвер L298P обеспечивает управление и подачу электрического тока на двигатели в соответствии с командами от микроконтроллера. Использование такого драйвера позволяет управлять двигателями без использования дополнительных программных средств.

Активный пьезоэлектрический излучатель: Данный излучатель используется для генерации звуковых сигналов. В данном случае он подает сигнал о том, что робот начал работать. Может работать только в двух режимах – излучает звук или не излучает.

3 аккумулятора Lilon 18650 11.1 В: Эти аккумуляторы предоставляют питание для работы всей системы. Они обеспечивают достаточную емкость и напряжение для длительного функционирования робота. Подобраны в соответствии с энергозатратностью системы.

Bluetooth модуль HC-06: Данный модуль обеспечивает беспроводную связь между роботом и другими устройствами, такими как смартфоны или компьютеры. Он позволяет передавать данные или команды для удаленного управления и мониторинга робота.

Датчик распознавания жестов PAJ7620: Этот датчик используется для распознавания жестов и команд от пользователя. Он позволяет роботу взаимодействовать с окружающей средой и выполнять определенные действия в ответ на жесты. Благодаря нему робот может управляться лишь жестами, без использования дополнительных устройств.

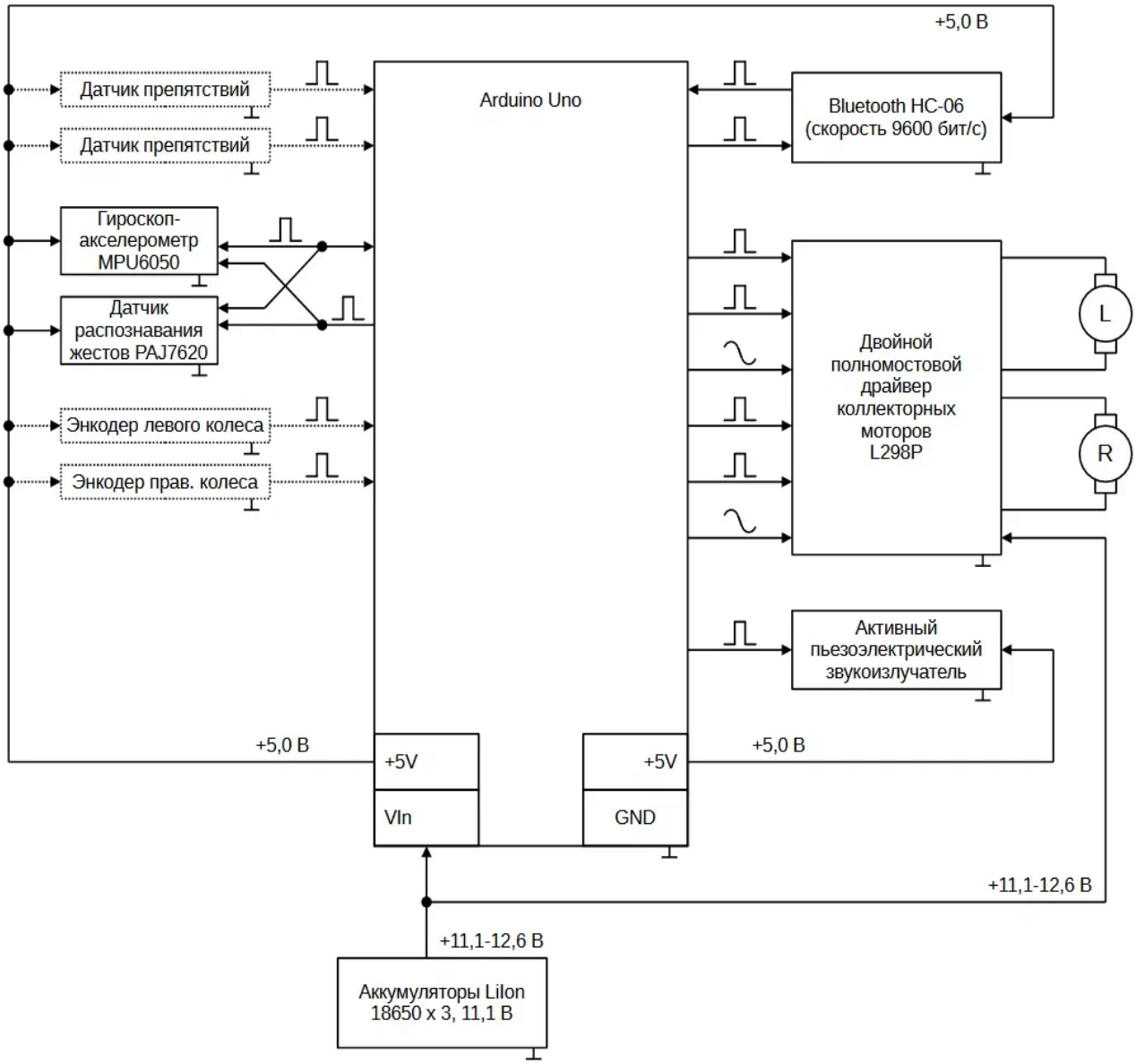

Рисунок 17 – Электрическая структурная схема робота

2.3 Разработка электрической функциональной схемы

В электрической функциональной схеме двухколесного самобалансирующегося робота (рисунок 18) компоненты взаимодействуют следующим образом:

Микроконтроллер Arduino UNO принимает данные об управляющей функции с гироскопа-акселерометра MPU6050 для определения текущего положения и ориентации робота, также получает управляющую функцию от ПИД регулятора на основе показаний гироскопа-акселерометра MPU6050. Выступает в роли главного управляющего устройства и обеспечивает функционирование системы в целом.

Управляющие сигналы передаются на двойной полномостовой драйвер L298P, который контролирует скорость и направление вращения каждого из двигателей.

Робот использует информацию от гироскопа-акселерометра и алгоритмы балансировки на основе управляющей функции ПИД регулятора для поддержания равновесия. Микроконтроллер регулирует скорость вращения каждого из двигателей, чтобы компенсировать наклон и поддерживать стабильное положение робота.

Активный пьезоэлектрический излучатель управляется микроконтроллером для генерации звуковых сигналов при включении системы.

Благодаря Bluetooth модулю HC-06, робот может устанавливать беспроводное соединение с другими устройствами и принимать команды удаленного управления и передавать данные для мониторинга.

Датчик распознавания жестов PAJ7620 [6] позволяет пользователю взаимодействовать с роботом с помощью жестов. Датчик передает данные микроконтроллеру, который обрабатывает эти данные и принимает соответствующие действия и решения в ответ. Также на схемы показаны опции, а именно: датчик препятствий, который может быть установлен и подключен. Можно использовать алгоритм, по которому робот сможет строить свой

маршрут исходя из условий окружающей среды, избегая препятствия. Потенциал для дальнейшей разработки.

Также в качестве опцией можно выбрать двигатели с энкодерами, которые обеспечат прямую обратную связь между роботом и управляющим устройством. Они могут считывать обороты колес и регулировать управляющую функцию учитывая эти показания. В данной системы они не были использованы в связи с дороговизной и недоступностью (стоимость двигателей с энкодерами в 10 раз превышает стоимость двигателей без них [7]). Добиться устойчивости системы можно без использования таких двигателей, однако использование таких двигателей может еще ближе приблизить систему к идеальному, абсолютно устойчивому положению на протяжении всей работы. Потенциал для дальнейшей разработки системы. Система подразумевает возможность установки таких двигателей, а также ранее упомянутых датчиков.

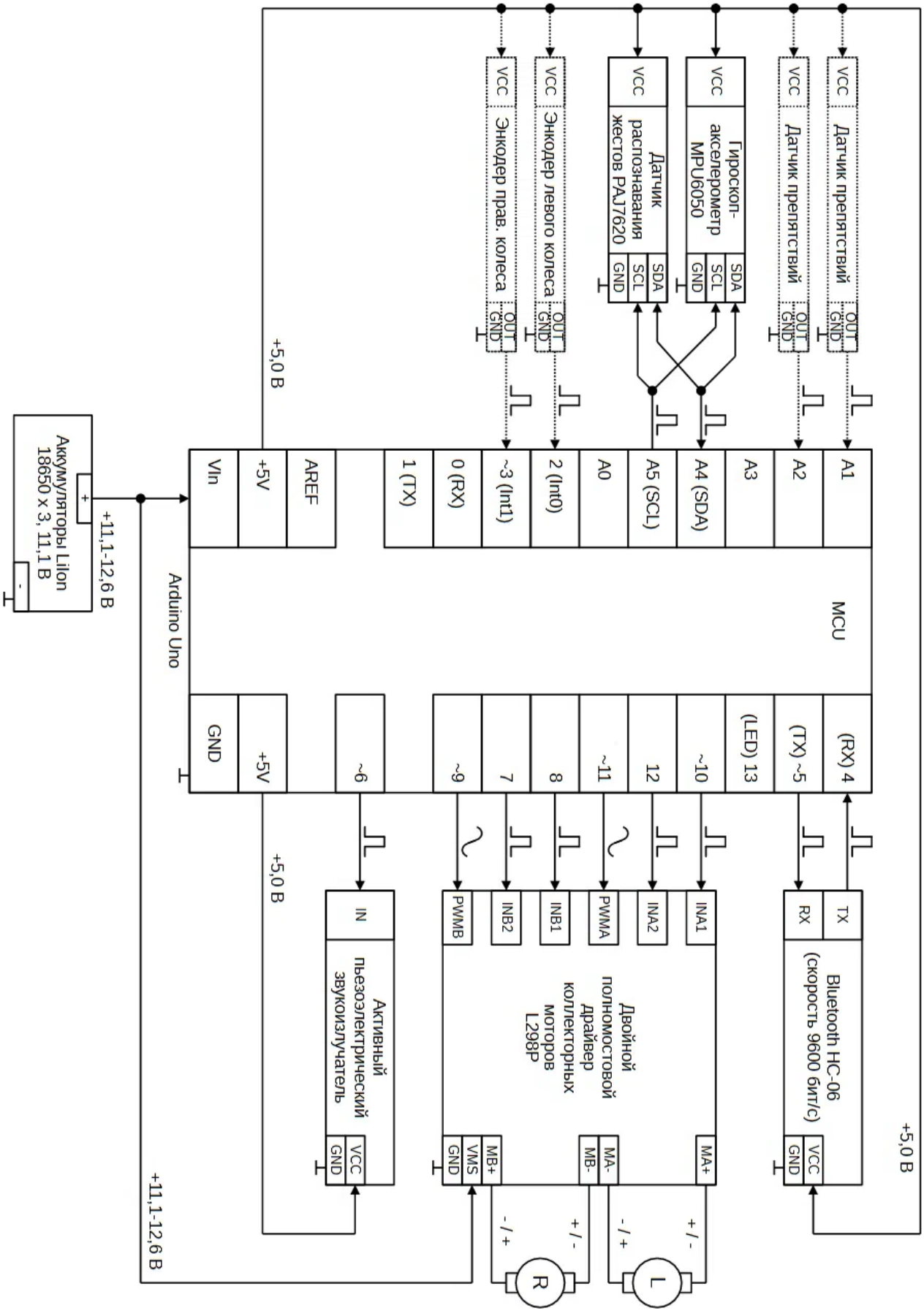

Рисунок 18 – Электрическая функциональная схема робота

2.4 Настройка ПИД регулятора

Для каждой системы, управляющейся ПИД регулятором необходимо подбирать уникальные коэффициенты для функции регулирования, однако существует несколько методов подбора коэффициентов, работоспособность которых по отношению к конкретной системе можно проверить лишь непосредственным опытом.

После того, как код скомпилировался и загрузился в плату можно приступать к настройке.

Далее необходимо закомментировать строки кода MOTOR1(PWM); MOTOR2(PWM); для того, чтобы двигатели не работали.

После этого поставить робота в вертикальное положение.

Подключиться к Arduino через USB, включить робота и в Arduino IDE открыть последовательный порт (частота этого порта составляет 115200).

В новом окне появятся значения угла поворота относительно оси Y (гироскопа). При отклонении робота от вертикальной оси в окне отображаются отклонения угла. Если значения не меняются — наклонить робота по оси X, если значения начали меняться, то в файле gyro_accel.cpp в строках 40-52 поменять значения по оси X с 0 на 90, по оси Y с 90 на 0.

После этого повторить процедуру.

Раскомментировать строки кода MOTOR1(PWM); MOTOR2(PWM); после этого при отклонении от нулевого положения робот будет получать управляющие функции, зависящие от P, I, D коэффициентов из прошлого пункта. Теперь можно приступать к калибровке ПИД регулятора. Коэффициенты регулируются в управляющем файле main.cpp. Они находятся на строках 34-36 кода соответственно (рисунок 19).

```
double Kp = 7;
double Kd = 0.3;
double Ki = 46;
```

Рисунок 19 – Форма задания коэффициентов в коде

2.4.1 Настройка ПИД регулятора методом Циглера-Никольса

Для начала необходимо обнулить все коэффициенты регулятора, то есть пропорциональный, интегральный и дифференциальный (рисунок 20).

```
double Kp = 0;
double Kd = 0;
double Ki = 0;
```

Рисунок 20 – Обнуленные коэффициенты ПИД регулятора

После обнуления необходимо постепенно увеличивать пропорциональный коэффициент и следить за тем, как на это реагирует система. При определенном значении пропорционального коэффициента возникнут незатухающие колебания регулируемой величины.

При достижении незатухающих колебаний фиксированной величины необходимо зафиксировать пропорциональный коэффициент - Kp, при котором эти колебания были достигнуты. Примем это значение за Кк – коэффициент колебаний. С точки зрения конкретно этой системы под незатухающими колебаниями регулируемой величины можно понимать резкие движения робота с наклоном сначала в одну сторону, затем в другую. При этом робот может ударяться о поверхность. Кроме того, необходимо замерить период колебаний системы, примем это значение за ТТ. В случае этой системы Кк принял значение 11.72, а период колебаний составил 0.34 секунды.

Из полученного коэффициента Кк рассчитывается пропорциональный коэффициент ПИД-регулятора:

$$Kp = 0.6 * Kк \quad (4)$$

Kp = 0.6 * 11.7 = 7.02 – высчитанный пропорциональный коэффициент.

Расчет интегрального коэффициента (Ki):

$$Ki = (2 * Kp) / TT = (2 * 7.02) / 0.34 = 41.29 \quad (5)$$

Расчет дифференциального коэффициента (Kd):

$$K_d = (K_p*TT)/8 = (7.02 * 0.34)/8 = 0.3 \quad (6)$$

2.4.2 Настройка PID регулятора при неработоспособности метода Циглера-Никольса

Начать необходимо также с обнуления всех коэффициентов.

После этого произвести увеличение пропорционального коэффициента, но теперь не с целью наблюдения появления колебаний, а для фиксации поведения системы при каждом значении пропорционального коэффициента. Отличным вариантом будет построение графика величины, которую необходимо стабилизировать, для каждого значения коэффициента (рисунок 19). Для этого необходимо открыть Serial Plotter в контекстном меню для отображения графиков в среде разработки Arduino IDE (Рисунок 21).

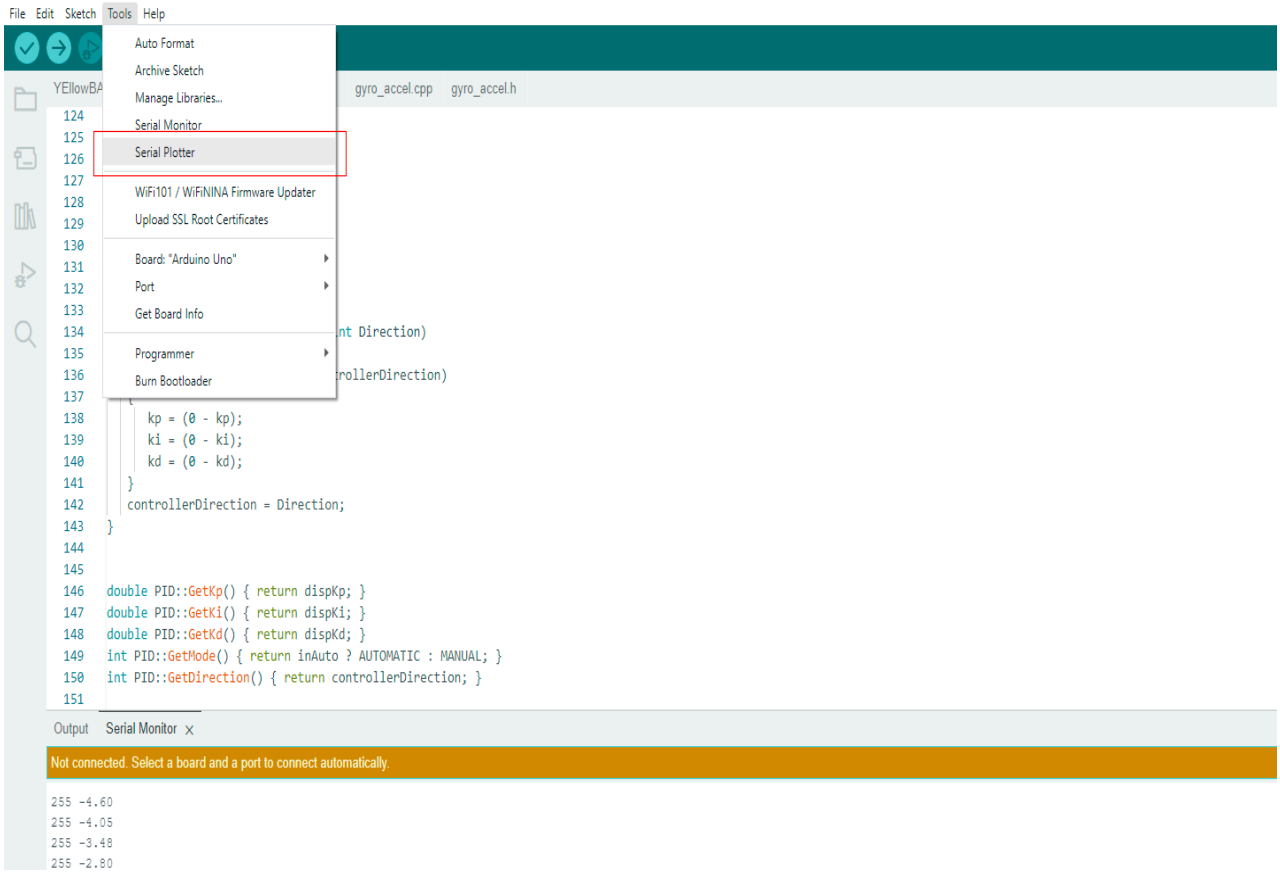

Рисунок 21 – Вызов Serial Plotter в Arduino IDE

После вызова Serial Plotter на экране появятся графики зависимостей.

Для того, чтобы графики отображались в контекстном меню Serial Plotter необходимо выбрать скорость 115200 (рисунок 22).

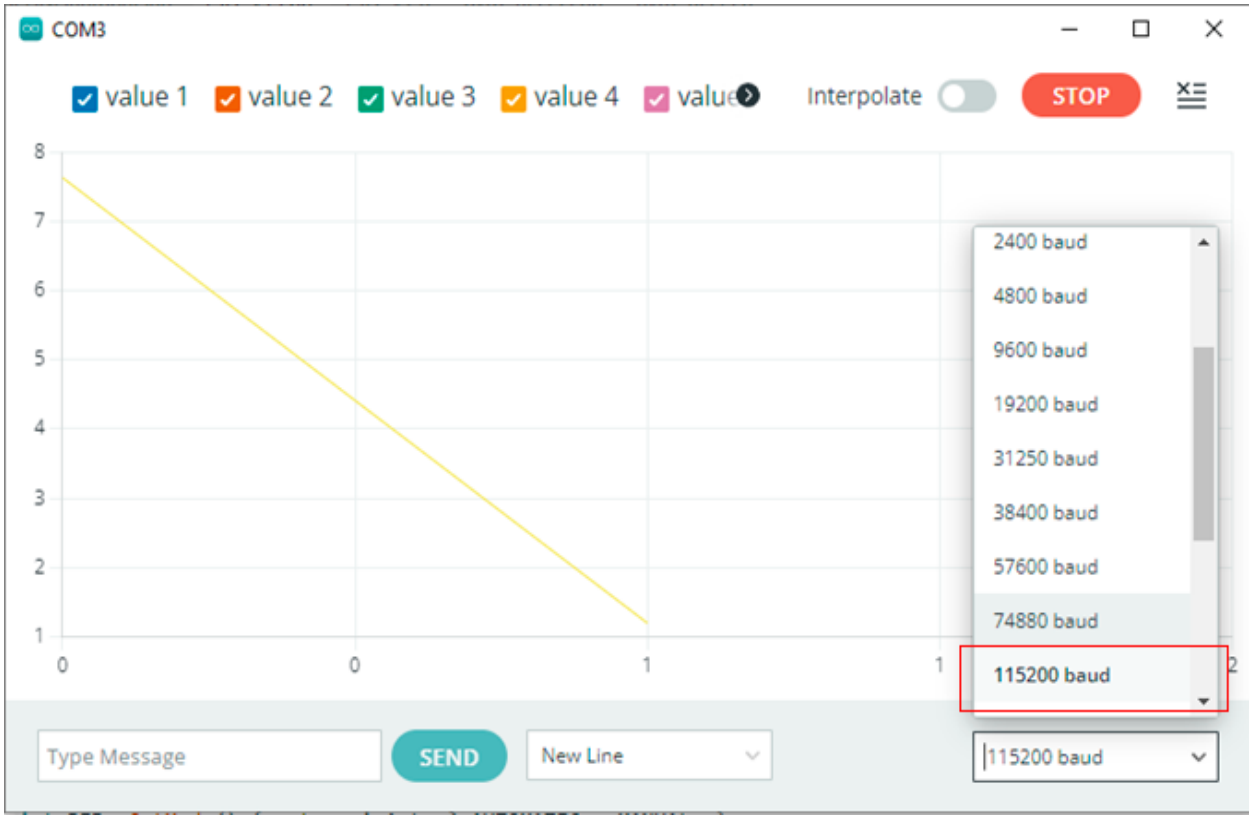

Рисунок 22 – Выбор скорости передачи данных в контекстном меню

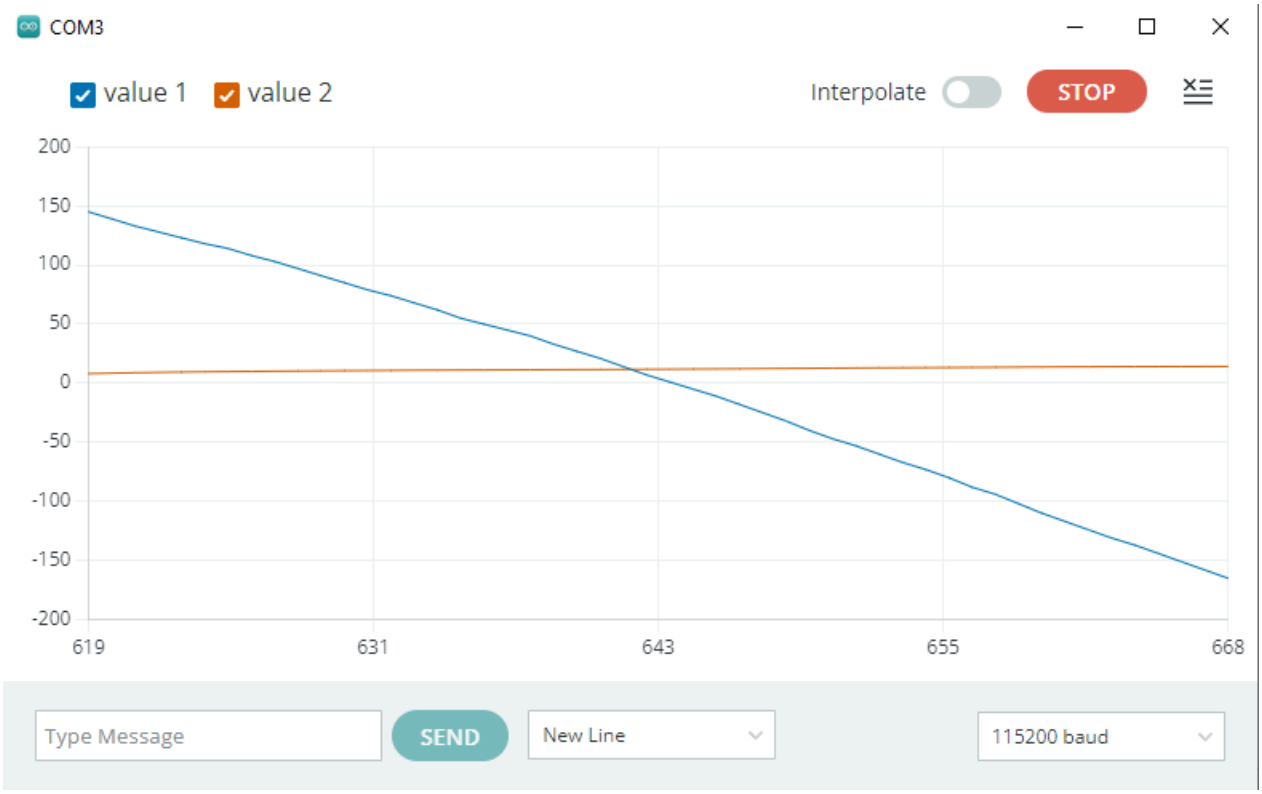

Рисунок 23 – Отображение графиков зависимостей

Если наглядно видно, что система слишком медленно реагирует на изменение показаний и слишком плавно выходит на оптимальные значения, то необходимо увеличить пропорциональный коэффициент. Если же система начинает слишком сильно колебаться относительно нужной величины, то это значит, что пропорциональный коэффициент слишком велик. Его необходимо уменьшить и перейти к настройке других составляющих.

2.4.3 Общие рекомендации по настройке PID регулятора

Понимая, как работает ПИД-регулятор в целом, и представляя, как должна работать настраиваемая система, можно довольно-таки быстро и точно настроить коэффициенты регулятора. Особенно, если есть возможность построить графические зависимости и визуально отслеживать поведение системы при изменениях коэффициентов регулирующей функции.

Вот некоторые правила, которые могут помочь при настройке ПИД-регулятора:

1. Увеличение пропорционального коэффициента приводит к увеличению быстродействия, но снижению устойчивости системы.
2. Увеличение дифференциальной составляющей также приводит к значительному увеличению быстродействия.
3. Дифференциальная составляющая призвана устранить затухающие колебания, возникающие при использовании лишь пропорциональной составляющей.
4. Интегральная составляющая устраняет остаточное рассогласование системы при настроенных пропорциональной и дифференциальной составляющих.

2.5 Разработка алгоритма работы программы

Данный код описывает реализацию алгоритма системы стабилизации на основе гироскопа и акселерометра с использованием ПИД-регулятора. Код разделен на несколько разделов: подключение необходимых библиотек и объявление констант и глобальных переменных, настройка пинов для моторов и ПИД-регулятора, определение функций управления моторами. Основной код программы определен в функциях setup() и loop(), которые будут описаны далее. Схема работы общей структуры программы показана на рисунке 24.

В начале программы подключаются необходимые библиотеки: Wire – стандартная библиотека, используется для связи микроконтроллера с устройствами и модулями через интерфейс I2C. PID_v1 для работы с ПИД-регулятором и gyro_accel.h для работы с гироскопом-акселерометром MPU6050 (рисунок 23).

```
#include <Wire.h>
#include "PID_v1.h"
#include "gyro_accel.h"
```

Рисунок 23 – Подключение необходимых библиотек

Затем определяются константы и глобальные переменные, необходимые для вычислений и управления системой стабилизации. Константы dt, rad2degree и Filter_gain используются в вычислениях угла наклона. Переменные angle_x_gyro, angle_y_gyro, angle_z_gyro хранят значения углов наклона от гироскопа, а переменные angle_x_accel, angle_y_accel, angle_z_accel - от акселерометра. Переменные angle_x, angle_y и angle_z хранят отфильтрованные значения углов наклона (рисунок 26).

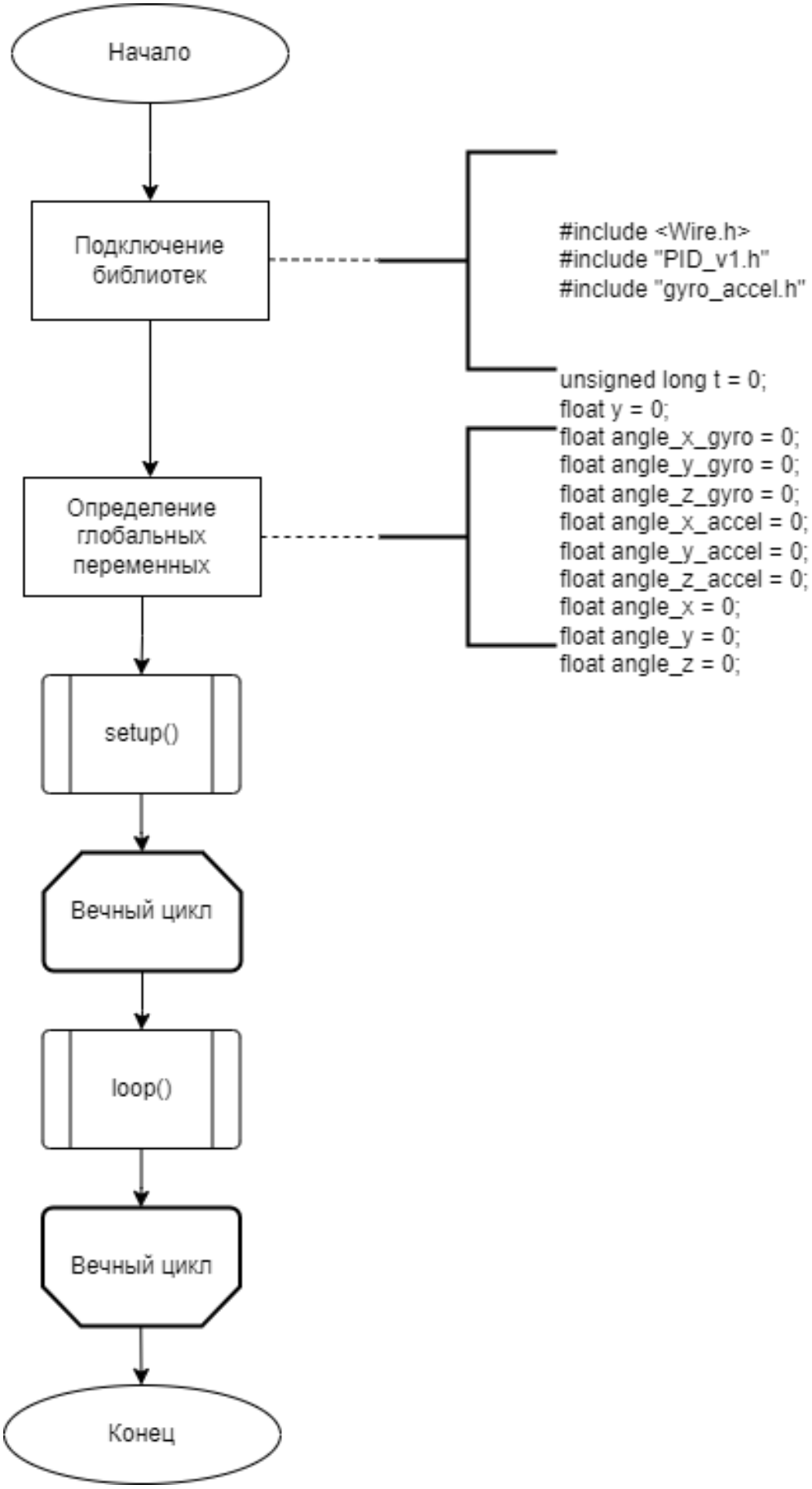

Рисунок 24 – Схема работы системы

Определены константы dt, rad2degree и Filter_gain, которые используются для вычисления угла наклона. Также задан порт для звукоизлучателя (рисунок 25).

```
#define dt 20
#define rad2degree 57.3
#define Filter_gain 0.95
#define SND_PIN 6
```

Рисунок 25 – Определение констант и порта для звукоизлучателя

#define dt 20 – задает интервал времени в промежутках 20 миллисекунд.

#define rad2degree 57.3 – множитель, позволяет переводить радианы в градусы. 1 радиан равен 57.3 градусам.

#define Filter_gain 0.95 - переменная для фильтра angle = angle_gyro*Filter_gain + angle_accel*(1-Filter_gain).

#define SND_PIN 6 – определение порта для работы со звукоизлучателем.

```
unsigned long t = 0;
float y = 0;
float angle_x_gyro = 0;
float angle_y_gyro = 0;
float angle_z_gyro = 0;
float angle_x_accel = 0;
float angle_y_accel = 0;
float angle_z_accel = 0;
float angle_x = 0;
float angle_y = 0;
float angle_z = 0;
```

Рисунок 26 – Определенные глобальные переменные

Далее происходит настройка пинов для моторов. Определены массивы PWM1, Umotor_1 и Umotor_2, которые содержат номера пинов для управления моторами с помощью ШИМ.

```
char PWM1[] = {11, 9};       // Номера пинов для ШИМ управления
char Umotor_1[] = {10, 12}; // Подключение первого двигателя
char Umotor_2[] = {7, 8};
```

Рисунок 27 – Настройка пинов для моторов

Определены переменны ПИД регулятора.

```
double originalSetpoint = -0.6;
double movingAngleOffset = 0.1;
double input, output;
double Kp = 7;
double Kd = 0.3;
double Ki = 46;
```

Рисунок 28 – Определение переменных ПИД регулятора

originalSetpoint - требуемое значение стабилизации, Kp, Kd и Ki - коэффициенты ПИД регулятора, принцип определения которых был описан в разделе алгоритм настройки ПИД регулятора.

Создается объект ПИД регулятора с передачей указателей на переменные input, output и originalSetpoint, также туда передаются коэффициенты ПИД регулятора. Режим работы установлен на "DIRECT".

```
PID pid(&input, &output, &originalSetpoint, Kp, Ki, Kd, DIRECT);
```

Рисунок 29 – Инициализация ПИД регулятора

2.5.1 Разработка алгоритма работы функции setup()

В функции setup происходит настройка пинов для управления моторами, инициализация последовательной связи с компьютером на скорости 115200, инициализация шины I2C, сброс настроек датчика MPU6050, настройка фильтра низких частот, калибровка гироскопа и акселерометра, настройка ПИД регулятора и ограничение выходных значений PID. Также настраивается порт для управления активным звукоизлучателем.

```
void setup()
{
  for (char pin = 0; pin <= 2; pin++)
    pinMode(PWM1[pin], OUTPUT);
  for (char pin1 = 0; pin1 <= 2; pin1++)
    pinMode(Umotor_1[pin1], OUTPUT);
  for (char pin2 = 0; pin2 <= 2; pin2++)
    pinMode(Umotor_2[pin2], OUTPUT);
  Serial.begin(115200);
  Wire.begin();
  MPU6050_ResetWake();
  MPU6050_SetGains(0, 1);
  MPU6050_SetDLPF(0);
  MPU6050_OffsetCal();
  MPU6050_SetDLPF(6);
  Serial.print("\tangle_y_accel");
  Serial.print("\tangle_y");
  Serial.println("\tLoad");
  t = millis();
  pid.SetMode(AUTOMATIC);
  pid.SetSampleTime(10);
  pid.SetOutputLimits(-255, 255); // Ограничение крайних значений PID
  // Настройка порта для управления пищалкой
  pinMode(SND_PIN, OUTPUT);
  digitalWrite(SND_PIN, HIGH);
  delay(30);
  digitalWrite(SND_PIN, LOW);
  delay(60);
  digitalWrite(SND_PIN, HIGH);
  delay(120);
  digitalWrite(SND_PIN, LOW);
}
```

Рисунок 30 – Функция setup()

Serial.begin(115200); - скорость работы последовательного порта (UART) Arduino (микроконтроллера AT mega328P). Этой строкой инициализируются порты 0 и 1 Arduino для передачи данных, а не для работы с сигналами. Дальше Arduino взаимодействует с преобразователем логических уровней (CH340, MAX2232, CP2102, ATmega16U и т.д.), который переводит сигналы ТТЛ (TTL) интерфейса RS232 в сигналы, «понятные» USB хосту, и обратно. Стандартные

скорости подразумевают передачу какого-то количества бит за секунду, их называют бодами. То есть скорость измеряется в бодах (последовательных битах) за секунду. Перечень скоростей, бод/с: 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600 и 115200. Самое главное в этом процессе (выбора скорости) чтобы и передатчик, и приёмник работали с одинаковой скоростью. Можно выбрать и нестандартную (как анти отладочный приём), но тогда не будет совместимости.

Wire.begin(); - инициализация библиотеки Wire и подключение к шине I2C в качестве ведущего или ведомого устройства. Вызывается один раз.

MPU6050_ResetWake(); - сбрасывает настройки гироскопа-акселерометра MPU6050 по умолчанию.

MPU6050_SetGains(0, 1); - настраивает максимальные значения шкалы измерений гироскопа и акселерометра.

MPU6050_SetDLPF(0); - настройка фильтра низких частот.

MPU6050_OffsetCal(); - вызов функции калибровки акселерометра и гироскопа.

MPU6050_SetDLPF(6); - настройка фильтра низких частот.

pid.SetOutputLimits(-255, 255); - ограничение крайних значений ПИД регулятора.

pinMode(SND_PIN, OUTPUT); - настройка порта для управления активным звукоизлучателем.

Схема работы алгоритма функции setup() показана на рисунке 31.

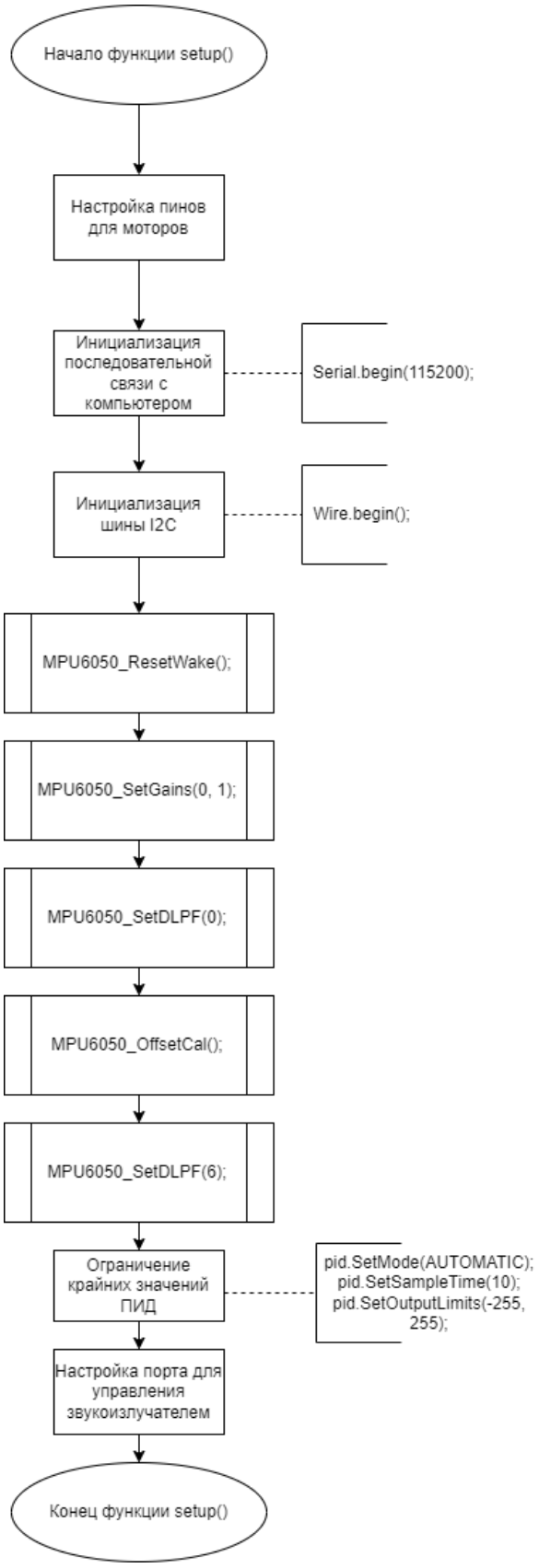

Рисунок 31 – Схема работы функции setup()

2.5.2 Разработка алгоритма работы функции loop()

В функции loop() сосредоточен основной код, управляющий процессом блаансировки. Сначала считываются данные с гироскопа-акселерометра MPU6050, а затем вычисляется угловая скорость angle_y_gyro с помощью гироскопа и угол поворота angle_y_accel с помощью акселерометра. Затем угол поворота относительно оси OY обрабатывается и фильтруется с использованием переменной Filter_gain. Полученное значение угла наклона передается в ПИД регулятор через переменную input, после чего вычисляется управляющий сигнал с помощью функции Compute. Значение выходного сигнала output передается на двигатели с помощью функций MOTOR1 и MOTOR2. Полный код функции loop() показан на рисунке 32.

```
void loop()
{
  t = millis();
  MPU6050_ReadData();
  angle_y_gyro = (gyro_y_scalled * ((float)dt / 1000) + angle_y);
  angle_y_accel = -atan(accel_x_scalled / (sqrt(accel_y_scalled *
accel_y_scalled + accel_z_scalled * accel_z_scalled))) *
(float)rad2degree;
  angle_y = Filter_gain * angle_y_gyro + (1 - Filter_gain) *
angle_y_accel;
  Serial.print(angle_y);
  Serial.println("\t");
  Serial.println(((float)(millis()-t)/(float)dt)*100);
  while ((millis() - t) < dt)
  {
  }
  input = angle_y;
  pid.Compute();
  int PWM = output;
  Serial.print(PWM);
  Serial.print(" ");
  MOTOR1(PWM);
  MOTOR2(PWM);
}
```

Рисунок 32 – Код функции loop()

Схема работы алгоритма функции loop() показана на рисунке 33.

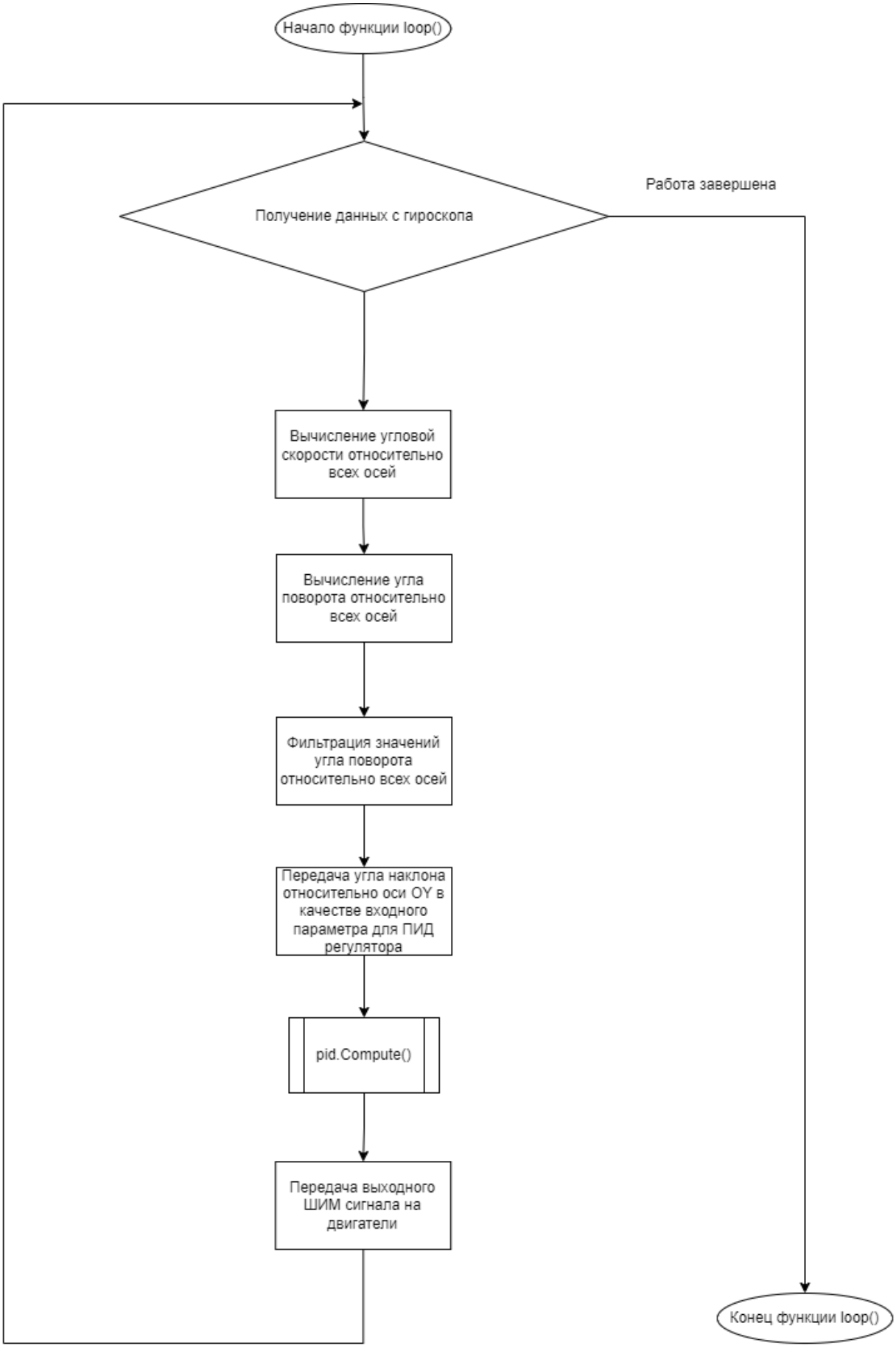

Рисунок 33 – Схема работы функции loop()

2.5.3 Разработка алгоритма работы функции pid.Compute()

Код функции pid.Compute() показан на рисунке 34. Функция вычисляет значение управляющего сигнала, с помощью которого происходит контроль двигателей. В переменной output хранится текущее значение управляющего сигнала.

```
void PID::Compute()
{
   if (!inAuto)
      return;
   unsigned long now = millis();
   unsigned long timeChange = (now - lastTime);
   if (timeChange >= SampleTime)
   {
      double input = *myInput;
      double error = *mySetpoint - input;
      ITerm += (ki * error);
      if (ITerm > outMax)
         ITerm = outMax;
      else if (ITerm < outMin)
         ITerm = outMin;
      double dInput = (input - lastInput);

      double output = kp * error + ITerm - kd * dInput;

      if (output > outMax)
         output = outMax;
      else if (output < outMin)
         output = outMin;
      *myOutput = output;

      lastInput = input;
      lastTime = now;
   }
}
```

Рисунок 34 – Код функции pid.Compute()

На первом шаге происходит проверка режима работы. Сначала происходит проверка переменной inAuto, которая указывает, включен ли режим автоматического регулирования. Если режим не включен (inAuto равно false), функция просто возвращает управление и завершается.

Затем функция вычисляет текущую временную метку, используя функцию millis(), которая возвращает количество миллисекунд, прошедших с момента запуска программы. Затем вычисляется временная разница (timeChange) между текущим временем и последним временем обновления.

Происходит вычисление ошибки: функция получает текущее значение входного сигнала (input) и вычисляет ошибку (error) путем вычитания желаемого значения (mySetpoint) из входного значения.

Вычисление интегральной составляющей: функция обновляет интегральную составляющую (ITerm) путем умножения ошибки на коэффициент интегрального усиления (ki) и добавления этого значения к предыдущему значению ITerm.

Ограничение интегральной составляющей: если значение ITerm превышает максимальное значение (outMax), оно устанавливается в равное outMax. Аналогично, если значение ITerm меньше минимального значения (outMin), оно устанавливается равным outMin. Это гарантирует, что интегральная составляющая остается в пределах заданных границ.

Вычисление дифференциальной составляющей: функция вычисляет изменение входного сигнала (dInput), вычитая предыдущее значение входа (lastInput) из текущего значения input. Дифференциальная составляющая (kd * dInput) используется для компенсации изменения входного сигнала.

Вычисление выходного значения: функция вычисляет выходное значение регулятора путем суммирования пропорциональной составляющей (kp * error), интегральной составляющей

Ограничение выходного значения: если значение output превышает максимальное значение (outMax), оно устанавливается равным outMax. Если значение output меньше минимального значения (outMin), оно устанавливается равным outMin. Это гарантирует, что выходной сигнал остается в пределах заданных границ.

Завершение функции: функция завершается после обновления переменных состояния. Схема работы функции pid.Compute() показана на рисунке 35.

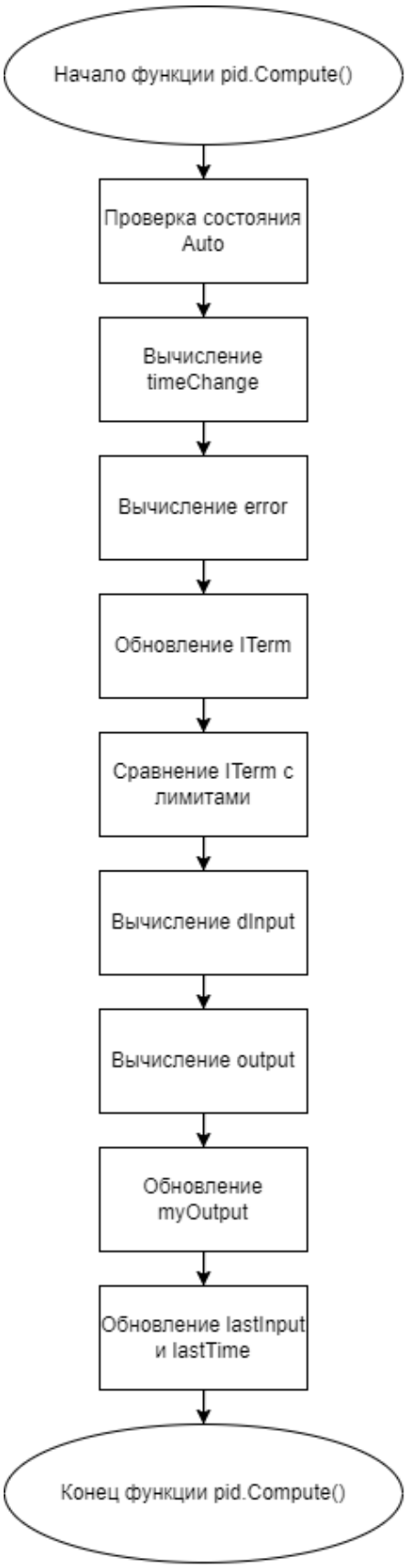

Рисунок 35 – Схема работы функции pid.Compute()

2.5.4 Разработка алгоритма работы функции MPU6050_OffsetCal()

Функция MPU6050_OffsetCal() отвечает за калибровку акселерометра-гироскопа MPU6050. Программный код этой функции представлен на рисунке 36.

```
void MPU6050_OffsetCal()
{
  Serial.println("Calibrating gyroscope .... do not move the hardware
.........");
  int x = 0, y = 0, z = 0, i;
  MPU6050_ReadData();
  MPU6050_ReadData();
  x = gyro_x;
  y = gyro_y;
  z = gyro_z;
  for (i = 1; i <= 1000; i++)
  {
    MPU6050_ReadData();
    x = (x + gyro_x) / 2;
    y = (y + gyro_y) / 2;
    z = (z + gyro_z) / 2;
    Serial.print(".");
  }
  Serial.println(".");
  gyro_x_OC = x;
  gyro_y_OC = y;
  gyro_z_OC = z;
  Serial.print("gyro_x register offset = ");
  Serial.println(x);
  Serial.print("gyro_y register offect = ");
  Serial.println(y);
  Serial.print("gyro_z register offset = ");
  Serial.println(z);
  Serial.println("Calibrating accelerometer .... dont move the hardware
.........");
  x = accel_x;
  y = accel_y;
  z = accel_z;
  for (i = 1; i <= 1000; i++)
  {
    MPU6050_ReadData();
    x = (x + accel_x) / 2;
    y = (y + accel_y) / 2;
    z = (z + accel_z) / 2;
    Serial.print(".");
  }
  Serial.println(".");
  accel_x_OC = x;
  accel_y_OC = y;
  accel_z_OC = z - (float)g * 1000 / accel_scale_fact;
  Serial.print("Accel_x register offset = ");
  Serial.println(x);
  Serial.print("Accel_y register offect = ");
  Serial.println(y);
  Serial.print("Accel_z register offset = ");
  Serial.println(z);
}
```

Рисунок 36 – Код функции MPU6050_OffsetCal()

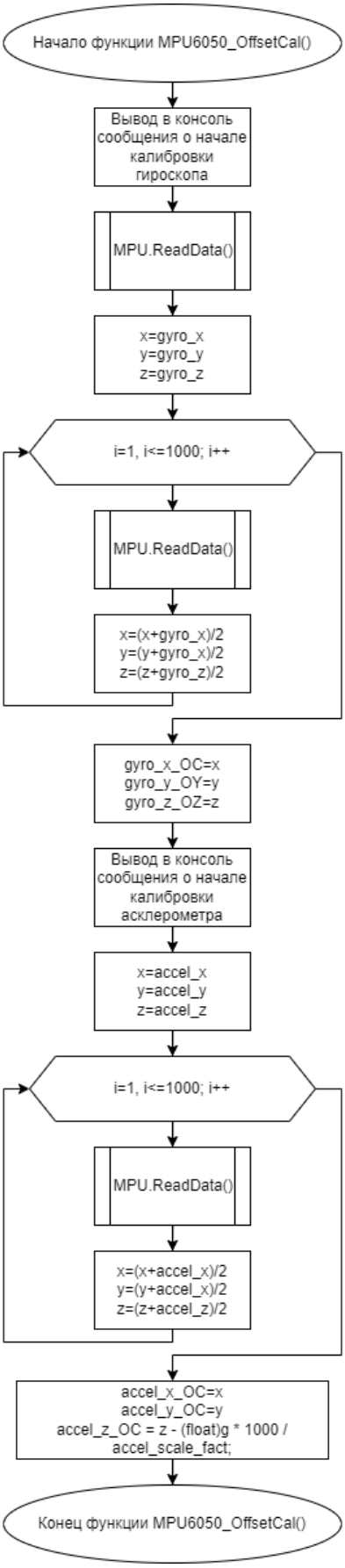

Рисунок 37 – Схема работы функции MPU6050_OffsetCal()

В начале функции выводится сообщение "Calibrating gyroscope .... dont move the hardware ....." для указания пользователю, что необходимо держать модуль неподвижно во время калибровки гироскопа.

Затем происходит инициализация переменных x, y и z для хранения временных значений гироскопа.

Далее вызывается функция MPU6050_ReadData(), которая считывает данные из модуля MPU6050.

После этого происходит повторное считывание данных с MPU6050 с целью усреднения значений гироскопа. В цикле, выполняющемся 1000 раз, значения гироскопа (gyro_x, gyro_y, gyro_z) усредняются с предыдущими значениями (x, y, z).

Затем значения x, y и z сохраняются в переменные gyro_x_OC, gyro_y_OC и gyro_z_OC соответственно. Эти значения представляют собой оффсеты (смещения) регистров гироскопа, которые будут использоваться для коррекции измерений.

Выводятся значения смещения регистров гироскопа в консоль.

Далее выводится сообщение "Calibrating accelrometer .... dont move the hardware ....." для указания пользователю, что необходимо держать модуль неподвижно во время калибровки акселерометра.

Повторяется процесс усреднения значений акселерометра (accel_x, accel_y, accel_z) с целью определения оффсетов регистров акселерометра. Аналогично гироскопу, значения усредняются в цикле, выполняющемся 1000 раз, и сохраняются в переменные accel_x_OC, accel_y_OC и accel_z_OC.

Значение accel_z_OC корректируется на значение гравитационного ускорения g для компенсации смещения оси z.

3 Технологический раздел

3.1 Технология настройки ПИД регулятора

Настройка ПИД регулятора для каждой конкретной системы – трудоемкий процесс, требующий большого количества экспериментов. Есть общие методики настройки ПИД регулятора, однако можно выделить некоторые особенности конкретно для этой системы.

Настройка методом Циглера-Никольса требует достижения периодических колебаний, однако эти колебания не всегда можно явно пронаблюдать. Если рассматривать колебания в рамках двухколесных самобалансирующихся роботов, то здесь важную роль играет множество факторов, которые эти колебания вызывают. Например, мощность двигателей, тонкости настройки остальных модулей системы в целом, правильная настройка и калибровка акселерометров и гироскопов и другие. Плавное увеличение пропорционального – P коэффициента – понятие относительное.

В случае этой системы резкое увеличение P коэффициента может привести к неоднозначным колебаниям, при которых робот будет ударяться о поверхность. Также важно, чтобы поверхность была максимально ровной и не вызывала дополнительных помех балансировке. Амплитуда колебаний в случае, когда пропорциональный коэффициент – P сильно завышен показана на рисунке 39. В случае, когда колебания имеют подобный характер и их период составляет менее 0.7 секунды рекомендуется значительно снизить пропорциональный коэффициент путем уменьшения его в 0.8 раз. Для начала следует обнулить все коэффициенты. В таком случае при запуске системы двигатели робота в движение приводится не будут. Это начальная точка.

После этого, путем увеличения пропорционального коэффициента до большого значения, например, 100 необходимо убедиться, что колеса будут вращаться слишком быстро и робот будет резко заваливаться в сторону.

Рисунок 38 – Вертикальное положение робота

ПИД регулятор должен быть настроен так, чтобы робот поддерживал такое положение с минимальными колебаниями или вообще их отсутствием в случае, если поверхность достаточно ровная.

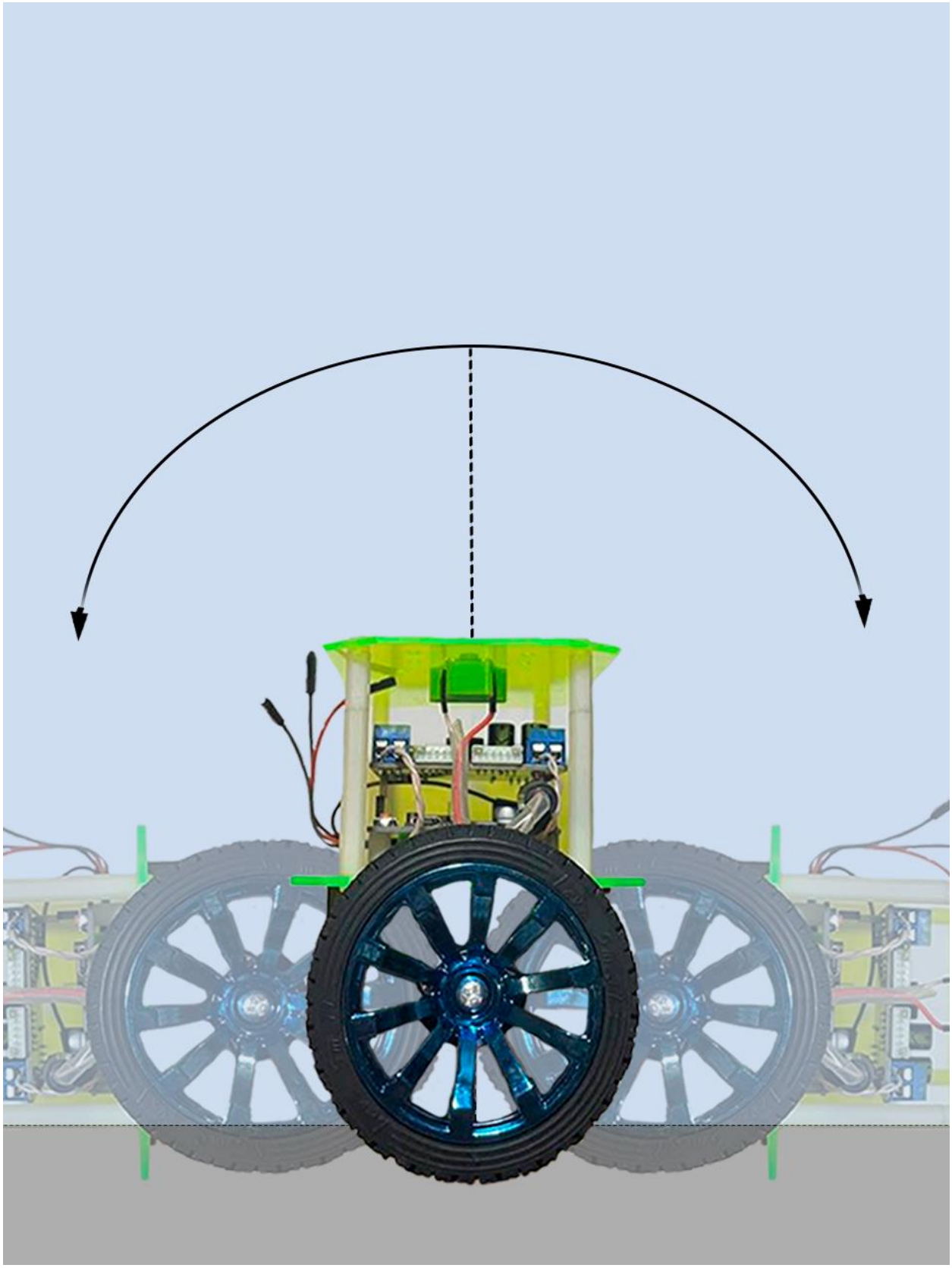

Рисунок 39 – Полная амплитуда колебаний

В случае, если колебания достигли амплитуды, как на рисунке 39 и при этом период этих колебаний меньше, чем 0.8 нужно продолжать уменьшать пропорциональный – P коэффициент. В случае, если колебания подобной амплитуды обрели период, меньший, чем 0.8 секунды – взять этот пропорциональный P коэффициент за коэффициент Кк – коэффициент колебаний. Далее этот коэффициент необходимо умножить на 0.6. В результате должны быть достигнуты колебания с амплитудой, отображенной на рисунке 40. Это промежуточный результат, который является отправной точкой для следующих процедур изменения коэффициентов. После этого, согласно формулам необходимо высчитать все коэффициенты для ПИД регулятора.

Расчет пропорционального коэффициента:

$$Kp = Kk * 0.6 = 11.7 * 0.6 = 7.02 \quad (4)$$

Расчет интегрального коэффициента (Ki):

$$Ki = (2 * Kp) / TT = (2 * 7.02) / 0.34 = 41.29 \quad (5)$$

Расчет дифференциального коэффициента (Kd):

$$K_d = (K_p*TT)/8 = (7.02 * 0.34)/8 = 0.3 \quad (6)$$

При правильно рассчитанных коэффициентах ПИД регулятора в дальнейшем колебаний системы возникать не будет.

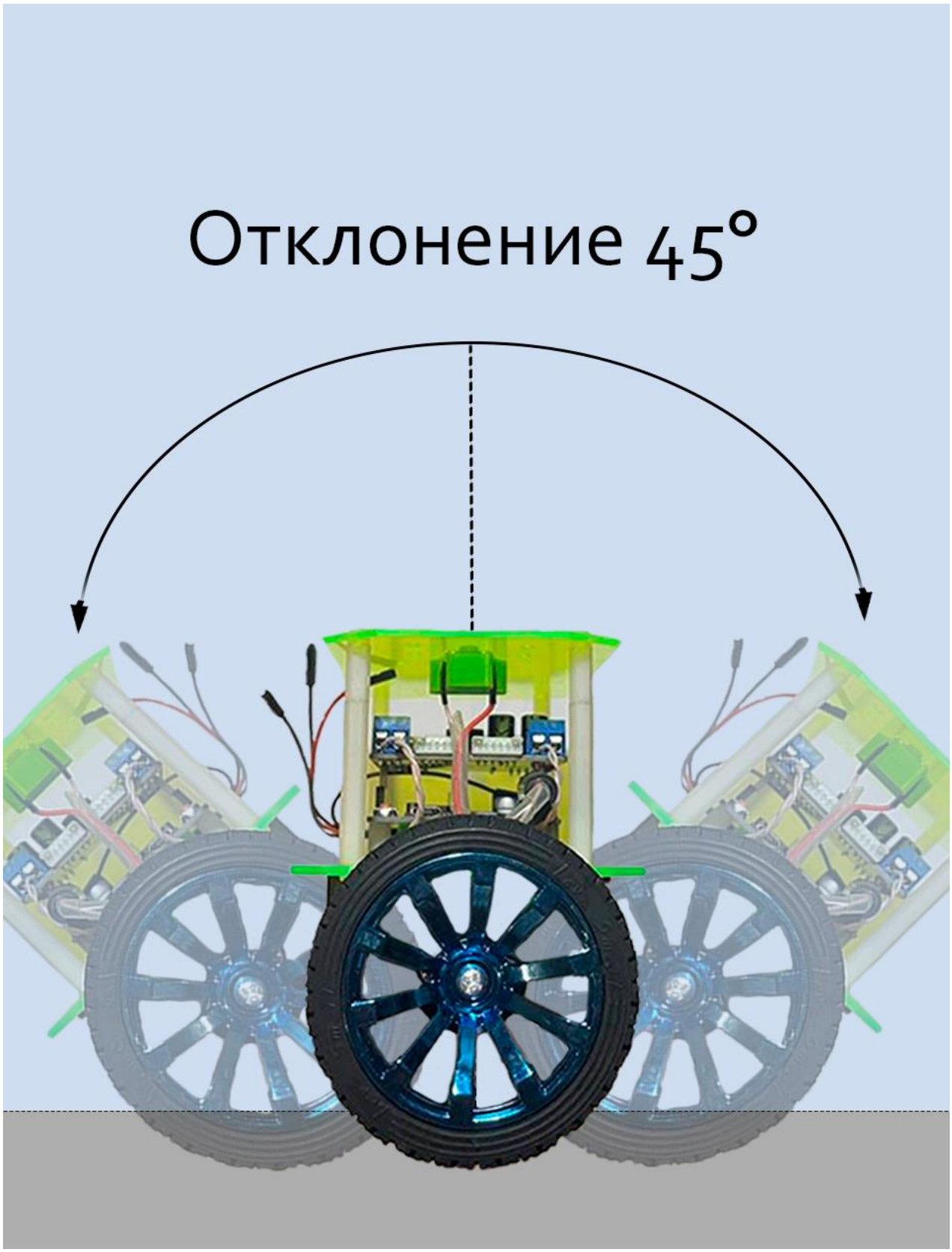

Рисунок 40 – Промежуточная амплитуда колебаний

3.2 Использование датчика PAJ7620 для управления роботом

Датчик PAJ7620 может быть использован для управления двухколесным самобалансирующимся роботом без использования Bluetooth модуля.

Для использования датчика PAJ7620 необходимо подключить его к микроконтроллеру (рисунок 41) и написать соответствующую программу. Программа должна определять, какой жест был сделан, и выполнять соответствующее действие. Например, если жест "вверх", то робот должен двигаться вперед, если жест "влево", то робот должен поворачиваться влево и т.д. Также для работы с датчиком существует библиотека [8].

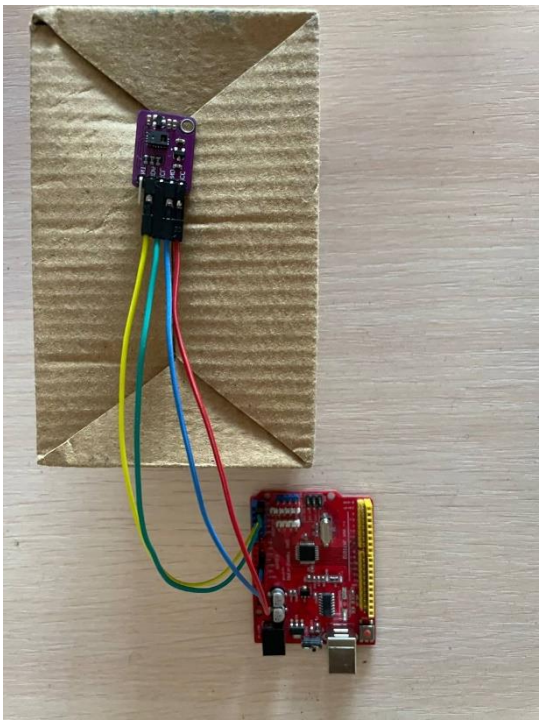

Рисунок 41 - Датчик PAJ7620, подключенный к микроконтроллеру Arduino

Были проведены эксперименты при различной степени освещенности помещения. При этом датчик смог однозначно и четко определить жесты в любых условиях. Схематически жесты показаны на рисунке 42. При этом жест «Up» подразумевает, что рука двигается в вертикальной плоскости снизу вверх, а жест «Forward» подразумевает, что рука движется в горизонтальной плоскости вперед. Жест «Down» подразумевает, что рука движется в вертикальной плоскости сверху вниз, а жест «Backward» подразумевает, что рука движется в горизонтальной плоскости назад.

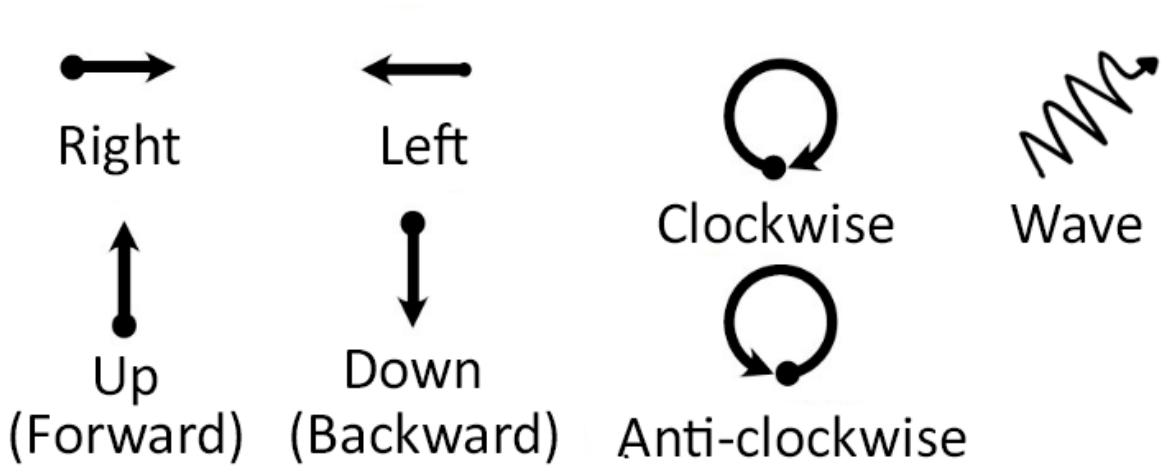

Рисунок 42 - Схематическое обозначение жестов

```
Down
Left
anti-clockwise
Right
Backward
Up
```

Рисунок 43 - Экспериментальные показания датчика PAJ7620

Датчик распознавания жестов PAJ7620 можно использовать для управления устройствами без необходимости касания их экрана или кнопок. Он определяет различные жесты и движения, которые могут быть связаны с определенными действиями, например:

Управление телевизором, плеером и другими устройствами: можно использовать жесты для переключения каналов, громкости и других параметров.

Управление компьютером: можно использовать жесты для запуска приложений, изменения громкости звука, перемещения по страницам и других действий.

Управление умным домом: можно использовать жесты для открытия дверей, включения и выключения света, регулирования температуры и других действий.

Кроме того, датчик PAJ7620 может быть использован для создания интерактивных игр и проектов, например, для управления дронами. В целом, датчик распознавания жестов PAJ7620 может быть полезен во многих областях, где требуется управление устройствами без необходимости касания экрана или кнопок.

Одно из важных преимуществ использования датчиков распознавания жестов – автономность при управлении роботами. Если для управления роботом используется Bluetooth или другие технологии удаленного управления, то для этого необходимо дополнительное устройство – телефон, планшет, пульт и т.д. При использовании же датчика жестов для управления роботом можно избежать необходимости наличия таких управляющих устройств и увеличить таким способом автономность робота.

3.2.1 Сравнение потребления тока датчика PAJ7620 и Bluetooth модуля

Использование Bluetooth для управления роботами в целом более энергозатратное, чем управление с помощью датчика жестов. Это особенно ощутимо, когда речь идет о компактных роботах, в которых энергопотребление каждого модуля или датчика на общем фоне очень значительно, поскольку в таких роботах используются маломощные источники энергии. Также для передачи данных по Bluetooth требуется постоянное подключение, в то время

как датчик жестов может работать на низкой мощности и не требует постоянного подключения. Произведем расчёт:

Для расчета энергопотребления датчика жестов PAJ7620 можно использовать следующую формулу:

$$E = P * t \quad (7),$$

где E - энергопотребление (в Дж), P - мощность потребления энергии датчика (в Вт), t - время работы датчика (в часах).

Согласно спецификации производителя [9], максимальный ток потребления датчика составляет 15 мА, при напряжении питания 3,3 В. Тогда мощность потребления энергии будет равна:

$$P = V * I = 3.3 \text{ В} * 0.015 \text{ А} = 0.0495 \text{ Вт} \quad (8)$$

При расчете временем работы можно пренебречь, поскольку мы подразумеваем, что и система с использованием датчика жестов и система с использованием Bluetooth работают на одинаковом промежутке времени.

Для расчета энергопотребления системы с Bluetooth на модуле HC-06 можно использовать аналогичную формулу:

Средняя мощность потребления энергии Bluetooth модуля составляет примерно 20 мА при напряжении питания 5 В. Тогда мощность потребления энергии согласно формуле 8 будет равна:

$P = V * I = 5 \text{ В} * 0.02 \text{ А} = 0.1 \text{ Вт}$.

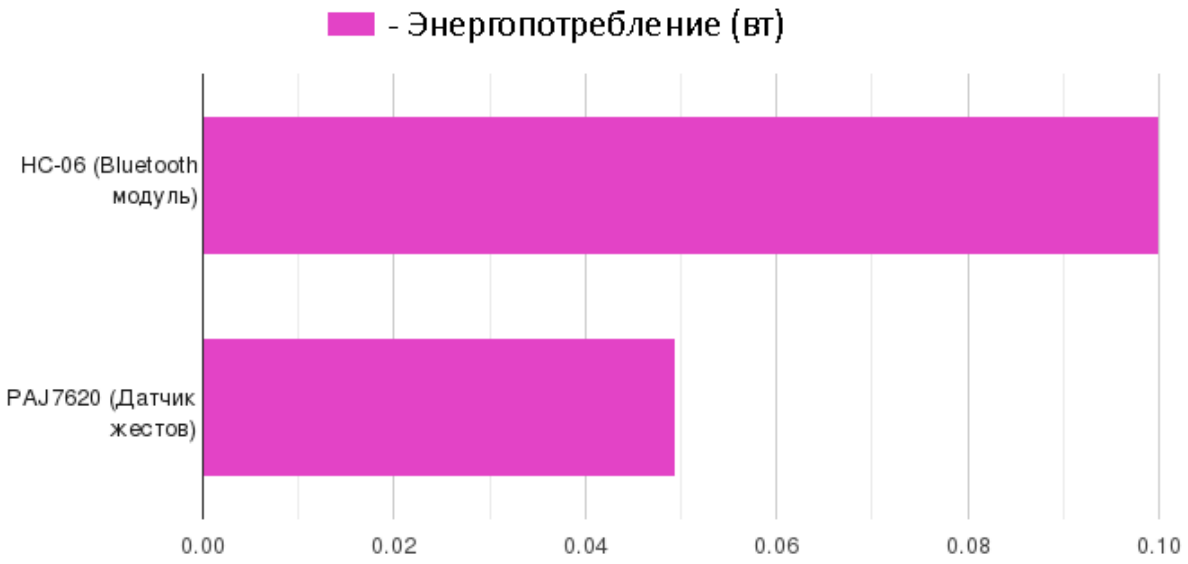

Рисунок 44 - Энергопотребления датчика жестов PAJ7620 и Bluetooth модуля

Таким образом, можно увидеть, что энергопотребление датчика жестов PAJ7620 значительно ниже, чем у системы с Bluetooth на модуле HC-06.

Очень важно также упомянуть, что сравнение потребления датчика жестов PAJ7620 и Bluetooth модуля HC-06 производилось без учета энергопотребления управляющего устройства при использовании системы на основе Bluetooth. Если в эти сравнения добавить к системе, использующей Bluetooth мощность потребления энергии управляющего устройства (например, смартфона), то результаты еще более однозначно покажут преимущества использования систем на основе датчика определения жестов PAJ7620 в рамках энергопотребления и автономности.

Также преимущество использования датчика жестов для управления роботом заключаются в том, что это более естественный и интуитивный способ управления. Человек обычно легко освоит этот метод управления, потому что он соответствует естественным движениям рук и тела. Это делает управление роботом более эффективным и удобным для пользователя.

Датчик жестов позволяет роботу получать информацию о действиях пользователя без необходимости постоянной связи с внешним источником. Это

может быть полезно в ситуациях, когда робот должен работать в условиях, где связь с другими устройствами невозможна или ограничена. Таким образом, использование датчика жестов для управления роботом может быть более эффективным и удобным способом управления, который также может помочь улучшить автономность робота.

В целом, датчик PAJ7620 — это очень полезное устройство для создания интерактивных роботов, которые могут общаться с человеком на языке жестов. Он имеет высокую точность распознавания жестов, малый размер и низкую потребляемую мощность, что делает его идеальным компонентом для будущего продолжения разработки в рамках системы двухколесного самобалансирующегося робота.

4 Экономический раздел

Для расчета стоимости разработки необходимо определить зарплату разработчика, а также стоимость всех компонентов.

Зарплата разработчика 13000 рублей в месяц. Разработка длилась 3 месяца, зарплата разработчика за 3 месяца – 13000 * 3 = 39000 рублей.

Для расчета стоимости компонентов необходимо привести цену каждого компонента.

Стоимость комплекта – двигатели, колеса с резиновыми шинами, переходники на валы двигателей – 594 рублей за комплект, 2 комплекта – 594 * 2 – 1188 рублей [10].

Стоимость щита с модулем MPU6050 и драйвером двигателей – 1059 рублей [11].

Стоимость комплекта Радиоконструктор Arduino uno R3.0 + шнур USB + PLS40 (ATMEGA328P + ATMEGA16U2) в ближайшем магазине – 1990 рублей [12].

Датчик распознавания жестов PAJ7620 – 320 рублей [6].

Лист органического стекла 30х30х0.3мм – 685 рублей [13].

Итоговая стоимость всех компонентов – 5242 рубль.

Общая стоимость проекта – 5242 + 39000 = 44242 рубля.

ЗАКЛЮЧЕНИЕ

В рамках работы был разработан двухколесный самобалансирующийся робот. В частности, были выполнены следующие задачи:

- Проанализированы существующие варианты реализации самобалансирующихся конструкций – двухколесных конструкций, моноколеса.
- Робот был собран, запрограммирован. Был настроен и откалиброван ПИД регулятор, способный поддерживать систему в равновесии.
- Было произведено тестирование двухколесного самобалансирующегося робота в различных условиях.

В дальнейшем планируется доработка и усовершенствование двухколесного самобалансирующегося робота, а также его использование на кафедре ИТАС ПНИПУ в рамках учебных целей. Таким образом, данная работа может быть полезной для студентов, преподавателей и посетителей кафедры, а также представляет интерес для дальнейших исследований в области робототехники.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Steve`s LegWay [Электронный ресурс] URL: http://www.teamhassenplug.org/robots/legway/ (Дата обращения 04.05.2023)

2. Робот VertiBot [Электронный ресурс] URL: https://24gadget.ru/1161053923-dvuhkolesnyy-samobalansiruyuschiysya-robot-video.html (Дата обращения 05.05.2023)

3. Segway-Ninebot [Электронный ресурс] URL: https://24gadget.ru/1161053923-dvuhkolesnyy-samobalansiruyuschiysya-robot-video.html (Дата обращения 05.05.2023)

4. Segyay-Ninebot mini S [Электронный ресурс] URL: https://segway-ninebot.ru/catalog/transport/giroskuter_ninebot_mini_s/ (Дата обращения 05.05.2023)

5. Двигатель с редуктором JGA25-370 [Электронный ресурс] URL: https://shop.robotclass.ru/item/274 (Дата обращения 05.05.2023)

6. Модуль датчика с распознаванием жестов PAJ7620 [Электронный ресурс] URL: https://aliexpress.ru/item/1005004350347548.html (Дата обращения 05.05.2023)

7. Двигатель JGB37-520 [Электронный ресурс] URL: https://aliexpress.ru/item/1005002816305762 (Дата обращения 05.05.2023)

8. Руководство PAJ7620. [Электронный ресурс] URL: https://roboparts.ru/upload/iblock/16a/16a366b2d45bcd6867d842f7a12d8dfb.pdf (Дата обращения 02.05.2023)

9. Датчик PAJ7620 техническая спецификация. [Электронный ресурс] URL: https://www.epsglobal.com/MediaLibrary/EPSGlobal/Products/files/pixart/PAJ7620F2.pdf?ext=.pdf (Дата обращения 12.05.2023)

10. Двигатели 25GA370 [Электронный ресурс] URL: https://aliexpress.ru/item/1005002885789611 (Дата обращения 05.05.2023)

11. Keyestudio быстроразъемный щит моторного привода V2 для робота Arduino [Электронный ресурс] URL: https://aliexpress.ru/item/33046129388 (Дата обращения 04.04.2023)

12. Радиоконструктор Arduino uno R3.0 + шнур USB + PLS40 (ATMEGA328P + ATMEGA16U2) [Электронный ресурс] URL: https://elecomp.ru/radiokonstruktor-arduino-uno-r3-0-shnur-usb-pls40-atmega328p-atmega16u2/ (Дата обращения 04.04.2023)

13. Органическое стекло, OZON [Электронный ресурс] URL: https://www.ozon.ru/product/polistirol-orgsteklo-beloe-listovoe-razmerom-30h30-sm-tolshchinoy-3-mm-476260165 (Дата обращения 01.04.2023)

14. Денисенко В.В. ПИД регуляторы: принципы построения и модификации. [Электронный ресурс] URL: https://www.cta.ru/cms/f/342946.pdf (Дата обращения 17.04.2023)

15. Отчет о научно-исследовательской работе. Структура и правила оформления [Текст]: ГОСТ 7.32-2001.

16. Соммер У. Программирование микроконтроллерных плат Arduino. – СПБ: БХВ-Петербург, 2012. – 256 с.

ПРИЛОЖЕНИЕ А
Техническое задание

УТВЕРЖДАЮ

Заведующий кафедрой ИТАС

_____Р.А. Файзрахманов

«____»_____2023

Разработка самобалансирующегося двухколесного робота

техническое задание

На 19 листах

Действует с 06.02.23

СОГЛАСОВАНО

Старший преподаватель кафедры ИТАС

_____А.Б. Федоров

«_____»_____2023

РАЗРАБОТАЛ

Студент группы КС-19-1б

_____А.А. Плотников

«_____»_____2023

Пермь 2023

СОДЕРЖАНИЕ

1 Общие сведения

Настоящее Техническое задание на разработку двухколесного самобалансирующегося робота выполнено в соответствии с ГОСТ 34.602-89 «Техническое задание на создание автоматизированной системы» и ГОСТ 19.201-78 «Единая система программной документации». В настоящем Техническом задании описаны общие требования к системе в целом.

1.1 Полное наименование системы и ее условное обозначение

Двухколесный самобалансирующийся робот.

1.2 Номер договора (контракта)

Не предусмотрено.

1.3 Наименования организации-заказчика и организаций-участников работ

1.3.1 Заказчик

Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет» (ПНИПУ), электротехнический факультет, кафедра «Информационные технологии и автоматизированные системы».

Старший преподаватель кафедры ИТАС А.Б. Федоров

Адрес фактический: г. Пермь, ул. Профессора Поздеева, 7.

Телефон: +7(342)219-83-19, доб. 126

1.3.2 Участник

Студент группы КС-19-1б:

Плотников Антон Андреевич

Телефон: +7(902)4786314

1.4 Перечень документов, на основании которых создается система

Не предусмотрено.

1.5 Плановые сроки начала и окончания работы по созданию системы

Начало работ по разработке робота февраль 2023 года, конец работ по разработке робота май 2023 года.

1.6 Источники и порядок финансирования работ

Собственные средства заказчика.

1.7 Порядок оформления и предъявления заказчику результатов работ по созданию системы

1.7.1 Выполнение работ по разработке системы

К результатам труда разработчика относится:

− оригинальное программное обеспечение;

− физическая реализация системы двухколесного самобалансирующегося робота;

− необходимые методические материалы для создания и настройки двухколесного самобалансирующегося робота;

Результаты работы предоставляются заказчику:

Полностью функционирующий двухколесный само балансирующийся робот.

Проектная документация должна быть разработана в соответствии с ГОСТ 34.201-89 и ГОСТ ЕСПД.

Работы по созданию системы должны осуществляться в порядке, установленном в разделе 5 настоящего Технического задания.

1.7.2 Приемка результатов работ

Приемка результатов работ должна проводиться в порядке, установленном в разделе 6 настоящего Технического задания.

2 Назначение и цели создания системы

2.1 Назначение системы

Двухколесный самобалансирующийся робот разрабатывается для решения таких задач, как:

1) Привлечение абитуриентов;

2) Изучение поведения двухколесных балансирующих конструкций на различных типах

Поверхностей;

3) Проведение мастер-классов по настройке ПИД регулятора.

2.2 Цели создания системы

Целью разработки является создание двухколесного самобалансирующегося робота, с помощью которого можно будет проводить тестирования устойчивости двухколесных балансирующих конструкций на различных типах поверхностей, проводить мастер классы по настройке ПИД регулятора, а также привлекать абитуриентов.

3 Характеристика объекта автоматизации

3.1 Краткие сведения об объекте автоматизации

Объектом автоматизации является процесс балансировки двухколесного робота.

3.2 Сведения об условиях эксплуатации объекта автоматизации и характеристиках окружающей среды.

Использование двухколесного робота должно происходить в лабораторных условиях, при которых ограничено попадание на конструкцию робота воды, снега и других нежелательных элементов. Робот должен использоваться под присмотром человека на специально подготовленных поверхностях.

4 Требования к системе

4.1 Требования к системе в целом;

Общие требования:

- выполнение всех функций, предусмотренных настоящим документом в пункте 4.2;

- выполнение требований к автоматизированной системе, предусмотренных настоящим документом в пунктах 4.1.1 – 4.1.13 соответственно.

4.1.1 Требования к структуре и функционированию системы

4.1.1.1 Перечень подсистем, их назначение и основные характеристики

Самобалансирующийся робот должен представлять собой систему, включающую в себя подсистемы:

- программного обеспечения;

- технического обеспечения.

4.1.1.2 Требования к способам и средствам связи для информационного обмена между компонентами системы

Управление самобалансирующимся роботом осуществляется с помощью кнопки подачи питания.

Установка и обновление программного обеспечения двухколесного самобалансирующегося робота происходит с помощью компьютера, установленной на него среды разработки Arduino, а также MicroUSB кабеля.

4.1.1.3 Требования к совместимости системы

Для запуска среды разработки Arduino необходим компьютер, который поддерживает эту среду разработки.

4.1.1.4 Требования к режимам функционирования системы

Не предусмотрены.

4.1.1.5 Требования по диагностированию системы

Не предусмотрены.

4.1.1.6 Перспективы развития, модернизации системы

Для самобалансирующегося двухколесного робота в рамках разработки были также подобраны возможные опции, которые можно добавить в будущем. Самобалансирующийся двухколесный робот – платформа для будущих разработок.

4.1.2 Требования к численности и квалификации персонала системы

Не предусмотрено.

4.1.3 Показатели назначения

4.1.3.1 Степень приспособляемости системы к изменению процессов и методов управления, к отклонениям параметров объекта управления

Не предусмотрено.

4.1.3.2 Допустимые пределы модернизации и развития системы

Не предусмотрено.

4.1.3.3 Вероятностно-временные характеристики, при которых сохраняется целевое назначение системы.

Не предусмотрено.

4.1.4 Требования к надежности

4.1.4.1 Состав и количественные значения показателей надежности для системы в целом или ее подсистем

Не предусмотрено.

4.1.4.2 Перечень аварийных ситуаций, по которым должны быть регламентированы требования к надежности, и значения соответствующих показателей

Не предусмотрено.

4.1.4.3 Требования к надежности технических средств и программного обеспечения;

Из требований к надежности технических средств и ПО можно выделить:

- ПО должно быть пригодным для эксплуатации и безошибочно выполнять требуемые функции;

- Робот должен работать стабильно.

Для обеспечения стабильной работы робота необходимо следить за правильностью его эксплуатации.

4.1.4.4 Требования к методам оценки и контроля показателей надежности на разных стадиях создания системы в соответствии с действующими нормативно-техническими документами

Не предусмотрено.

4.1.5 Требования к безопасности

Не предусмотрено.

4.1.6 Требования к эргономике и технической эстетике

Двухколесный самобалансирующийся робот должен выглядеть «презентабельно», чтобы его было можно демонстрировать на экскурсиях.

4.1.7 Требования к транспортабельности для подвижных АС

Не предусмотрено.

4.1.8 Требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы

Не предусмотрено.

4.1.9 Требования по сохранности информации при авариях

Не предусмотрено.

4.1.10 Требования к защите от влияния внешних воздействий

Для защиты от внешних воздействий необходимо придерживаться требований к условиям эксплуатации.

4.1.11 Требования к патентной частоте

Не предусмотрено.

4.1.12 Требования по стандартизации и унификации

Не предусмотрено.

4.1.13 Дополнительные требования

Не предусмотрено.

4.2 Требования к функциям (задачам), выполняемым системой

Робот должен удерживать баланс на ровных поверхностях за счет управляющей функции, формирующейся ПИД регулятором.

4.2.1 Перечень функций, задач или их комплексов (в том числе обеспечивающих взаимодействие частей системы), подлежащих автоматизации

Балансировка робота.

4.2.2 Временной регламент реализации каждой функции, задачи (или комплекса задач)

Возможна корректировка в процессе разработки робота.

4.2.3 Требования к качеству реализации каждой функции (задачи или комплекса задач), к форме представления выходной информации, характеристики необходимой точности и времени выполнения, требования одновременности выполнения группы функций, достоверности выдачи результатов

Возможна корректировка в процессе разработки робота.

4.2.4 Перечень и критерии отказов для каждой функции, по которой задаются требования по надежности.

Возможна корректировка в процессе разработки робота.

4.3 Требования к видам обеспечения

4.3.1 Требования к математическому обеспечению системы

Математическое обеспечение должно удовлетворять следующим требованиям:

- адекватность данных;
- точность.

4.3.2 Требования информационному обеспечению системы

Не предусмотрено.

4.3.3 Требования к лингвистическому обеспечению системы

Не предусмотрено.

4.3.4 Требования к программному обеспечению системы

Система, поддерживающая среду разработки Arduino.

4.3.5 Требования к техническому обеспечению

Система, поддерживающая среду разработки Arduino.

4.3.6 Требования к метрологическому обеспечению

Не предусмотрено.

4.3.7 Требования к организационному обеспечению

Не предусмотрено.

4.3.8 Требования к методическому обеспечению

Не предусмотрено.

5 Состав и содержание работ по созданию (развитию) системы

5.1 Перечень документов, по ГОСТ 34.201-89, предъявляемых по окончании соответствующих стадий и этапов работ

Не предусмотрено.

5.2 Вид и порядок проведения экспертизы технической документации (стадия, этап, объем проверяемой документации, организация-эксперт)

Не предусмотрено.

5.3 Программу работ, направленных на обеспечение требуемого уровня надежности разрабатываемой системы (при необходимости)

Не предусмотрено.

5.4 Перечень работ по метрологическому обеспечению на всех стадиях создания системы с указанием их сроков выполнения и организаций-исполнителей (при необходимости)

Не предусмотрено.

6 Порядок контроля и приемки системы

Приемка этапа заключается в рассмотрении и оценке проведенного объема работ и предъявленной технической документации в соответствии с требованиями настоящего технического задания. Ответственность за организацию и проведение приемки системы должен нести заказчик. Приемка системы должна производиться по завершению приемки всех задач системы.

Заказчик должен предъявлять систему ведомственной приемочной комиссии, при этом он обязан обеспечить нормальные условия работы данной комиссии в соответствии с принятой программой приемки. Завершающим этапом при приемке системы должно быть составление акта приемки.

6.1 Виды, состав, объем и методы испытаний системы

Не предусмотрено.

6.2 Общие требования к приемке работ по стадиям

Не предусмотрено.

6.3 Статус приемочной комиссии

Не предусмотрено.

7 Требования к составу и содержанию работ по подготовке объекта разработки к вводу в действие

Для обеспечения ввода объекта в действие провести комплекс мероприятий:

- приобрести компоненты технического и программного обеспечения, заключить договора на их лицензионное использование;
- завершить работы по установке технических средств;
- завершить работы по установке программных средств.

8 Требования к документированию

Проектная документация должна быть разработана в соответствии с ГОСТ 34.201-89 и ГОСТ ЕСПД. Отчетные материалы должны включать в себя текстовые материалы (представленные в виде бумажной копии и на цифровом носителе в формате PDF) и графические материалы.

9 ИСТОЧНИКИ РАЗРАБОТКИ

В качестве источников по разработке данного технического задания были использованы:

- ГОСТ 34.602-89 «Техническое задание на создание АС»;

- ГОСТ 19.201-78 «Единая система программной документации»;

- документация из курсовых и дипломных работ студентов ПНИПУ на аналогичную тему.

СОСТАВИЛ

Наименование организации, предприятия	Должность исполнителя	Фамилия имя, отчество	Подпись	Дата
ПНИПУ	Студент	Плотников Антон Андреевич		

СОГЛАСОВАНО

Наименование организации, предприятия	Должность исполнителя	Фамилия имя, отчество	Подпись	Дата
ПНИПУ	Старший преподаватель кафедры ИТАС	Федоров Андрей Борисович		

ПРИЛОЖЕНИЕ Б

Листинг файла «main.ino»

```
#include <Wire.h>       // шина I2C
#include "PID_v1.h"     // PID
#include "gyro_accel.h" // mpu6050
// вычисления угла наклона, постоянные
#define dt 20            // промежуток времени в милисикундах
#define rad2degree 57.3  // переменная для перевода из радиан в градусы
#define Filter_gain 0.95 // переменная для фильтра angle =
angle_gyro*Filter_gain + angle_accel*(1-Filter_gain)
#define SND_PIN 6 // пищалка
// Глобальные переменные
unsigned long t = 0;
float y = 0;
float angle_x_gyro = 0;
float angle_y_gyro = 0;
float angle_z_gyro = 0;
float angle_x_accel = 0;
float angle_y_accel = 0;
float angle_z_accel = 0;
float angle_x = 0;
float angle_y = 0;
float angle_z = 0;
// Настройка пинов для моторов
char PWM1[] = {11, 9};      // Номера пинов для ШИМ управления
char Umotor_1[] = {10, 12}; // Подключение первого двигателя
char Umotor_2[] = {7, 8};   // Подключение первого двигателя
// Переменные для ПИД регулятора
double originalSetpoint = -0.6; // Требуемое значение стабилизации
double movingAngleOffset = 0.1;
double input, output;
double Kp = 7;  // 7 - лучшее значение, подобранное мной
(предположительно - лучше не сделать)
double Kd = 0.3; // 0.3 - аналогично
double Ki = 46; // здесь 46, диапазон +-1.5
// Вызов ПИД регулятора
PID pid(&input, &output, &originalSetpoint, Kp, Ki, Kd, DIRECT);
// Функция для управления первым мотором
void MOTOR1(int PWM_1)
{
  if (PWM_1 > 0)
  {
    digitalWrite(Umotor_1[0], HIGH);
    digitalWrite(Umotor_1[1], LOW);
    analogWrite(PWM1[0], PWM_1);
  }
  else
  {
    digitalWrite(Umotor_1[0], LOW);
    digitalWrite(Umotor_1[1], HIGH);
    analogWrite(PWM1[0], abs(PWM_1));
  }
}
// Функция для управления вторым мотором
void MOTOR2(int PWM_2)
{
  if (PWM_2 > 0)
  {
    digitalWrite(Umotor_2[0], LOW);
    digitalWrite(Umotor_2[1], HIGH);
    analogWrite(PWM1[1], PWM_2);
  }
```

```
    else
    {
      digitalWrite(Umotor_2[0], HIGH);
      digitalWrite(Umotor_2[1], LOW);
      analogWrite(PWM1[1], abs(PWM_2));
    }
  }

  // Основной код
  void setup()
  {
    for (char pin = 0; pin <= 2; pin++)
      pinMode(PWM1[pin], OUTPUT);
    for (char pin1 = 0; pin1 <= 2; pin1++)
      pinMode(Umotor_1[pin1], OUTPUT);
    for (char pin2 = 0; pin2 <= 2; pin2++)
      pinMode(Umotor_2[pin2], OUTPUT);
    Serial.begin(115200);   // Общение с компьютером на частоте 115200
    Wire.begin();           // Работа с I2C шиной
    MPU6050_ResetWake();    // Сбрасывает настройки по умолчанию
    MPU6050_SetGains(0, 1); // Настраивает максимальные значения шкалы
измерений гироскопа и акселерометра
    MPU6050_SetDLPF(0);     // Настройка фильтра низких частот
    MPU6050_OffsetCal();    // Калибровка гироскопа и акселерометра
    MPU6050_SetDLPF(6);     // Настройка фильтра низких частот
    Serial.print("\tangle_y_accel");
    Serial.print("\tangle_y");
    Serial.println("\tLoad");
    t = millis();
    pid.SetMode(AUTOMATIC);
    pid.SetSampleTime(10);
    pid.SetOutputLimits(-255, 255); // Ограничение крайних значений PID
    // Настройка порта для управления пищалкой
    pinMode(SND_PIN, OUTPUT);
    digitalWrite(SND_PIN, HIGH);
    delay(30);
    digitalWrite(SND_PIN, LOW);
    delay(60);
    digitalWrite(SND_PIN, HIGH);
    delay(120);
    digitalWrite(SND_PIN, LOW);
  }
  void loop()
  {
    t = millis();
    MPU6050_ReadData();
    angle_y_gyro = (gyro_y_scalled * ((float)dt / 1000) + angle_y);
    angle_y_accel = -atan(accel_x_scalled / (sqrt(accel_y_scalled *
accel_y_scalled + accel_z_scalled * accel_z_scalled))) * (float)rad2degree;
    angle_y = Filter_gain * angle_y_gyro + (1 - Filter_gain) *
angle_y_accel;
    Serial.print(angle_y);
    Serial.println("\t");
    {
    }
    input = angle_y;
    pid.Compute();
    int PWM = output; // Передача выходного ШИМ на двигатели
    Serial.print(PWM);
    Serial.print(" ");
    MOTOR1(PWM);
    MOTOR2(PWM);
  }
```

ПРИЛОЖЕНИЕ В
Листинг файла «PID_v1.cpp»

```
#if ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif
#include "PID_v1.h"
PID::PID(double *Input, double *Output, double *Setpoint,
         double Kp, double Ki, double Kd, int ControllerDirection)
{
   PID::SetOutputLimits(0, 255);
   SampleTime = 100;
   PID::SetControllerDirection(ControllerDirection);
   PID::SetTunings(Kp, Ki, Kd);
   lastTime = millis() - SampleTime;
   inAuto = false;
   myOutput = Output;
   myInput = Input;
   mySetpoint = Setpoint;
}
void PID::Compute()
{
   if (!inAuto)
      return;
   unsigned long now = millis();
   unsigned long timeChange = (now - lastTime);
   if (timeChange >= SampleTime)
   {
      double input = *myInput;
      double error = *mySetpoint - input;
      ITerm += (ki * error);
      if (ITerm > outMax)
         ITerm = outMax;
      else if (ITerm < outMin)
         ITerm = outMin;
      double dInput = (input - lastInput);
      double output = kp * error + ITerm - kd * dInput;
      if (output > outMax)
         output = outMax;
      else if (output < outMin)
         output = outMin;
      *myOutput = output;
      lastInput = input;
      lastTime = now;
   }
}
void PID::SetTunings(double Kp, double Ki, double Kd)
{
   if (Kp < 0 || Ki < 0 || Kd < 0)
      return;
   dispKp = Kp;
   dispKi = Ki;
   dispKd = Kd;
   double SampleTimeInSec = ((double)SampleTime) / 1000;
   kp = Kp;
   ki = Ki * SampleTimeInSec;
   kd = Kd / SampleTimeInSec;
   if (controllerDirection == REVERSE)
   {
      kp = (0 - kp);
      ki = (0 - ki);
```

```
      kd = (0 - kd);
   }
}
void PID::SetSampleTime(int NewSampleTime)
{
   if (NewSampleTime > 0)
   {
      double ratio = (double)NewSampleTime / (double)SampleTime;
      ki *= ratio;
      kd /= ratio;
      SampleTime = (unsigned long)NewSampleTime;
   }
}
void PID::SetOutputLimits(double Min, double Max)
{
   if (Min >= Max)
      return;
   outMin = Min;
   outMax = Max;
   if (inAuto)
   {
      if (*myOutput > outMax)
         *myOutput = outMax;
      else if (*myOutput < outMin)
         *myOutput = outMin;

      if (ITerm > outMax)
         ITerm = outMax;
      else if (ITerm < outMin)
         ITerm = outMin;
   }
}
void PID::SetMode(int Mode)
{
   bool newAuto = (Mode == AUTOMATIC);
   if (newAuto == !inAuto)
   {
      PID::Initialize();
   }
   inAuto = newAuto;
}
void PID::Initialize()
{
   ITerm = *myOutput;
   lastInput = *myInput;
   if (ITerm > outMax)
      ITerm = outMax;
   else if (ITerm < outMin)
      ITerm = outMin;
}
void PID::SetControllerDirection(int Direction)
{
   if (inAuto && Direction != controllerDirection)
   {
      kp = (0 - kp);
      ki = (0 - ki);
      kd = (0 - kd);
   }
   controllerDirection = Direction;
}
double PID::GetKp() { return dispKp; }
double PID::GetKi() { return dispKi; }
double PID::GetKd() { return dispKd; }
```

ПРИЛОЖЕНИЕ Г
Листинг файла «PID_v1.h»

```
class PID
{
public:
#define AUTOMATIC 1
#define MANUAL 0
#define DIRECT 0
#define REVERSE 1
  PID(double *, double *, double *,
      double, double, double, int);
  void SetMode(int Mode);
  void Compute();
  void SetOutputLimits(double, double);
  void SetTunings(double, double, double);
  void SetControllerDirection(int);
  void SetSampleTime(int);
  double GetKp();
  double GetKi();
  double GetKd();
  int GetMode();
  int GetDirection();
private:
  void Initialize();
  double dispKp;
  double dispKi;
  double dispKd;
  double kp;
  double ki;
  double kd;
  int controllerDirection;
  double *myInput;
  double *myOutput;
  double *mySetpoint;
  unsigned long lastTime;
  double ITerm, lastInput;
  int SampleTime;
  double outMin, outMax;
  bool inAuto;
};
```

ПРИЛОЖЕНИЕ Д
Листинг файла «gyro_accel.cpp»

```
#include <Arduino.h>
#include <Wire.h>
#include "gyro_accel.h"
#define MPU6050_address 0x68
#define MPU6050_self_test_x 13
#define MPU6050_self_test_y 14
#define MPU6050_self_test_z 15
#define MPU6050_self_test_A 16
#define MPU6050_sample_div 25
#define MPU6050_config 26
#define MPU6050_gyro_config 27
#define MPU6050_accel_config 28
#define MPU6050_data_start 59
#define MPU6050_PWR1 107
#define MPU6050_PWR2 108
#define g 9.81
int temp = 0;
int accel_x = 0;
int accel_y = 90;
int accel_z = 0;
int gyro_x = 0;
int gyro_y = 90;
int gyro_z = 0;
int accel_x_OC = 0;
int accel_y_OC = 90;
int accel_z_OC = 0;
int gyro_x_OC = 0;
int gyro_y_OC = 90;
int gyro_z_OC = 0;
float temp_scalled;
float accel_x_scalled;
float accel_y_scalled;
float accel_z_scalled;
float gyro_x_scalled;
float gyro_y_scalled;
float gyro_z_scalled;
float accel_scale_fact = 1;
float gyro_scale_fact = 1;
void MPU6050_ReadData()
{
  Wire.beginTransmission(MPU6050_address);
  Wire.write(MPU6050_data_start);
  Wire.endTransmission();
  int read_bytes = 14;
  Wire.requestFrom(MPU6050_address, read_bytes);
  if (Wire.available() == read_bytes)
  {
    accel_x = Wire.read() << 8 | Wire.read();
    accel_y = Wire.read() << 8 | Wire.read();
    accel_z = Wire.read() << 8 | Wire.read();
    temp = Wire.read() << 8 | Wire.read();
    gyro_x = Wire.read() << 8 | Wire.read();
    gyro_y = Wire.read() << 8 | Wire.read();
    gyro_z = Wire.read() << 8 | Wire.read();
  }
  accel_x_scalled = (float)(accel_x - accel_x_OC) * accel_scale_fact / 1000;
  accel_y_scalled = (float)(accel_y - accel_y_OC) * accel_scale_fact / 1000;
  accel_z_scalled = (float)(accel_z - accel_z_OC) * accel_scale_fact / 1000;
  gyro_x_scalled = (float)(gyro_x - gyro_x_OC) * gyro_scale_fact / 1000;
  gyro_y_scalled = (float)(gyro_y - gyro_y_OC) * gyro_scale_fact / 1000;
```

```
  gyro_z_scalled = ((float)(gyro_z - gyro_z_OC) * gyro_scale_fact / 1000);
  temp_scalled = (float)temp / 340 + 36.53;
}
void MPU6050_ResetWake()
{
  Serial.println("Resetting MPU6050 and waking it up.....");
  Wire.beginTransmission(MPU6050_address);
  Wire.write(MPU6050_PWR1);
  Wire.write(0b10000000);
  Wire.endTransmission();
  delay(100);
  Wire.beginTransmission(MPU6050_address);
  Wire.write(MPU6050_PWR1);
  Wire.write(0b00000000);
  Wire.endTransmission();
}
void MPU6050_SetDLPF(int BW)
{
  if (BW < 0 || BW > 6)
  {
    BW = 0;
  }
  Wire.beginTransmission(MPU6050_address);
  Wire.write(MPU6050_config);

  Wire.write(BW);
  Wire.endTransmission();
}
void MPU6050_SetGains(int gyro, int accel)
{
  byte gyro_byte, accel_byte;
  Wire.beginTransmission(MPU6050_address);
  Wire.write(MPU6050_gyro_config);
  if (gyro == 0)
  {
    gyro_scale_fact = (float)250 * 0.0305;
    gyro_byte = 0b00000000;
  }
  else if (gyro == 1)
  {
    gyro_scale_fact = 500 * 0.0305;
    gyro_byte = 0b00001000;
  }
  else if (gyro == 2)
  {
    gyro_scale_fact = 1000 * 0.0305;
    gyro_byte = 0b00010000;
  }
  else if (gyro == 3)
  {
    gyro_scale_fact = 2000 * 0.0305;
    gyro_byte = 0b00011000;
  }
  else
  {
    gyro_scale_fact = 1;
  }
  Wire.write(gyro_byte);
  Wire.endTransmission();
  Serial.print("The gyro scale is set to ");
  Serial.print(gyro_scale_fact);
  Serial.println(" milli Degree/s");
  Wire.beginTransmission(MPU6050_address);
```

```
  Wire.write(MPU6050_accel_config);
  if (accel == 0)
  {
    accel_scale_fact = (float)2 * g * 0.0305;
    accel_byte = 0b00000000;
  }
  else if (accel == 1)
  {
    accel_scale_fact = 4 * g * 0.0305;
    accel_byte = 0b00001000;
  }
  else if (accel == 2)
  {
    accel_scale_fact = 8 * g * 0.0305;
    accel_byte = 0b00010000;
  }
  else if (accel == 3)
  {
    accel_scale_fact = 16 * g * 0.0305;
    accel_byte = 0b00011000;
  }
  else
  {
    accel_scale_fact = 1;
  }
  Wire.write(accel_byte);
  Wire.endTransmission();
  Serial.print("The accel scale is set to ");
  Serial.print(accel_scale_fact);
  Serial.println(" milli m/s^2");
}
void MPU6050_OffsetCal()
{
  Serial.println("Calibrating gyroscope .... dont move the hardware
.........");
  int x = 0, y = 0, z = 0, i;
  MPU6050_ReadData();
  MPU6050_ReadData();
  x = gyro_x;
  y = gyro_y;
  z = gyro_z;
  for (i = 1; i <= 1000; i++)
  {
    MPU6050_ReadData();
    x = (x + gyro_x) / 2;
    y = (y + gyro_y) / 2;
    z = (z + gyro_z) / 2;
    Serial.print(".");
  }
  Serial.println(".");
  gyro_x_OC = x;
  gyro_y_OC = y;
  gyro_z_OC = z;
  Serial.print("gyro_x register offset = ");
  Serial.println(x);
  Serial.print("gyro_y register offect = ");
  Serial.println(y);
  Serial.print("gyro_z register offset = ");
  Serial.println(z);
  Serial.println("Calibrating accelrometer .... dont move the hardware
.........");
  x = accel_x;
  y = accel_y;
```

```
  z = accel_z;
  for (i = 1; i <= 1000; i++)
  {
    MPU6050_ReadData();
    x = (x + accel_x) / 2;
    y = (y + accel_y) / 2;
    z = (z + accel_z) / 2;
    Serial.print(".");
  }
  Serial.println(".");
  accel_x_OC = x;
  accel_y_OC = y;
  accel_z_OC = z - (float)g * 1000 / accel_scale_fact;
  Serial.print("Accel_x register offset = ");
Serial.println(x);
  Serial.print("Accel_y register offect = ");
  Serial.println(y);
  Serial.print("Accel_z register offset = ");
  Serial.println(z);
}
```

ПРИЛОЖЕНИЕ Е
Листинг файла «gyro_accel.h»

```
extern int accel_x_OC;
extern int accel_y_OC;
extern int accel_z_OC;
extern int gyro_x_OC;
extern int gyro_y_OC;
extern int gyro_z_OC;

extern float temp_scalled;
extern float accel_x_scalled;
extern float accel_y_scalled;
extern float accel_z_scalled;
extern float gyro_x_scalled;
extern float gyro_y_scalled;
extern float gyro_z_scalled;

void MPU6050_ReadData();
void MPU6050_ResetWake();
void MPU6050_SetDLPF(int BW);
void MPU6050_SetGains(int gyro, int accel);
void MPU6050_OffsetCal();
```