**Балансирующий робот**

Выполнил: Бредихин Максим

Проверил: Карпов Валерий Эдуардович

Москва, 2024 г.

Содержание

1. Введение в проблему балансировки робота…………….……..….….2
2. Математические основы балансировки……………………….……...3
3. ПИД-регулятор…………………………………………………………6
4. Список элементов …………………………………………………….7

4.1 Драйвер мотора (Motor Driver)…………………………………….8

4.2 IMU-модуль MPU 6050……………………………………………..9

4.2.1 Фильтрация данных IMU……………………………………………………………........12

4.3 Мотор-редуктор с энкодерами………………………………..13

1. Схема подключения…………………………….……………………...15
2. Механическая часть…………...……………………………………….17
3. Алгоритм управления………………………………………………….19

7.1 Структурная схема………………………………………………….......19

7.2 Настройка коэффициентов ПИД-регуляторов…………………………………………………………….....21

7.3 Необходимость среднего контура управления……………………..22

7.4 Метод масштабирования коэффициентов…………………………..22

7.5 Компенсация нелинейности щеточных моторов…………………...22

1. Задача движения робота вперед-назад…………………………………23

8.1 Добавление небольшой дельты к желаемой скорости моторов…………………………………………………………………….23

8.2 Добавление дельты к сетпоинту угла наклона……………………...23

8.3 Комбинированный подход……………………………………………23

1. Заключение ……………………………………………………………...24
2. Приложение……………………………………………………………...24

10.1 Код проекта…………………………………………………………..24

10.2 Сторонние библиотеки……………………………………………....30

1. Список использованных источников…………………………………...3

1.Введение в проблему балансировки робота

Балансировка двухколесного робота — сложная задача управления, подобная удержанию в устойчивом положении перевернутого маятника. Перевернутый маятник — известная задача теории управления, в которой стержень установлен на платформе, которая может двигаться. Цель состоит в том, чтобы удерживать стержень в верхнем положении, несмотря на естественное желание стержня упасть. Инвертированный маятник , как и двухколесный робот, является системой с неустойчивым равновесием .Когда маятник или робот отклоняется от вертикального положения, сила тяжести начинает действовать, увеличивая отклонение. Чтобы удерживать систему в вертикальном положении и не допускать ее падения, необходимо определить и приложить управляющую силу, что и является основной работа при создании системы управления. Эта система управляема и наблюдаема, это означает, что можно создать алгоритм управления, который сможет поддерживать систему в вертикальном положении даже при наличии возмущений. В следующей части будет рассмотрена математическая модель перевернутого маятника. Эта модель поможет создать алгоритмы управления и подробнее изучить систему.

**2.Математические основы балансировки**

Рассмотрим систему "маятник-тележка". Масса маятника сосредоточена в его центре тяжести, который находится в центре маcc стержня. Масса тележки обозначена как , а масса маятника - какУправляющая сила F действует вдоль оси x тележки. Длина стержня обозначена как l. Угол наклона маятника обозначается как θ.

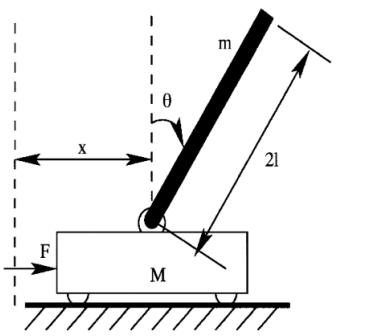
****

Рис.1 Перевернутый маятник.

Перемещение маятника по горизонтальной оси есть сумма перемещения маятника и перемещения тележки:

Запишем уравнения динамики

**Линеаризация модели**

При малых углах отклонения можно принять:

Тогда получаем итоговые уравнения:

{"mathml":"<math style=\"font-family:Times New Roman;font-size:14px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"14px\"><mtext>&#x41F;&#x443;&#x441;&#x442;&#x44C;&#xA0;&#x432;&#x435;&#x43A;&#x442;&#x43E;&#x440;&#xA0;&#x43F;&#x440;&#x43E;&#x441;&#x442;&#x440;&#x430;&#x43D;&#x441;&#x442;&#x432;&#x430;&#xA0;&#x441;&#x43E;&#x441;&#x442;&#x43E;&#x44F;&#x43D;&#x438;&#x439;&#xA0;&#xA0;</mtext><mi>&#x3B7;</mi><mo>=</mo><mo>[</mo><mi>x</mi><mo>,</mo><mover><mi>x</mi><mo>&#x2D9;</mo></mover><mo>,</mo><mi>&#x3B8;</mi><mo>,</mo><mover><mi>&#x3B8;</mi><mo>&#x2D9;</mo></mover><msup><mo>]</mo><mi>T</mi></msup><mo>.</mo><mspace linebreak=\"newline\"/><mtext>&#xA0;&#x417;&#x430;&#x43C;&#x435;&#x43D;&#x430;&#xA0;(1)&#xA0;&#x43D;&#x430;&#xA0;(2)&#xA0;&#x43F;&#x440;&#x438;&#x432;&#x43E;&#x434;&#x438;&#x442;&#xA0;:&#xA0;</mtext><mspace linebreak=\"newline\"/><mover><mi>&#x3B8;</mi><mo>&#xA8;</mo></mover><mo>=</mo><mfrac><mrow><msub><mi>m</mi><mi>p</mi></msub><mi>l</mi></mrow><mrow><msub><mi>I</mi><mi>p</mi></msub><mo>+</mo><msub><mi>m</mi><mi>p</mi></msub><msup><mi>l</mi><mn>2</mn></msup></mrow></mfrac><mo>(</mo><mi>g</mi><mi>&#x3B8;</mi><mo>-</mo><mover><mi>x</mi><mo>&#xA8;</mo></mover><mo>)</mo><mo>=</mo><mfrac><mrow><msub><mi>m</mi><mi>p</mi></msub><mi>l</mi></mrow><mrow><msub><mi>I</mi><mi>p</mi></msub><mo>+</mo><msub><mi>m</mi><mi>p</mi></msub><msup><mi>l</mi><mn>2</mn></msup></mrow></mfrac><mfenced><mrow><mi>g</mi><mi>&#x3B8;</mi><mo>-</mo><mfrac><mn>1</mn><mrow><msub><mi>m</mi><mi>c</mi></msub><mo>+</mo><msub><mi>m</mi><mi>p</mi></msub></mrow></mfrac><mo>(</mo><mo>-</mo><msub><mi>m</mi><mi>p</mi></msub><mi>l</mi><mover><mi>&#x3B8;</mi><mo>&#xA8;</mo></mover><mo>+</mo><mi>F</mi><mo>)</mo></mrow></mfenced><mo>&#x21D2;</mo><mover><mi>&#x3B8;</mi><mo>&#xA8;</mo></mover><mo>=</mo><mfrac><mrow><msub><mi>m</mi><mi>p</mi></msub><mi>l</mi></mrow><mrow><msub><mi>I</mi><mi>p</mi></msub><mo>+</mo><mfrac><mrow><msub><mi>m</mi><mi>c</mi></msub><msub><mi>m</mi><mi>p</mi></msub><msup><mi>l</mi><mn>2</mn></msup></mrow><mrow><msub><mi>m</mi><mi>c</mi></msub><mo>+</mo><msub><mi>m</mi><mi>p</mi></msub></mrow></mfrac></mrow></mfrac><mfenced><mrow><mi>g</mi><mi>&#x3B8;</mi><mo>-</mo><mfrac><mn>1</mn><mrow><msub><mi>m</mi><mi>c</mi></msub><mo>+</mo><msub><mi>m</mi><mi>p</mi></msub></mrow></mfrac><mi>F</mi></mrow></mfenced><mspace linebreak=\"newline\"/><mi>&#x41F;&#x443;&#x442;&#x435;&#x43C;</mi><mo>&#xA0;</mo><mi>&#x43F;&#x440;&#x435;&#x43E;&#x431;&#x440;&#x430;&#x437;&#x43E;&#x432;&#x430;&#x43D;&#x438;&#x439;</mi><mo>&#xA0;</mo><mi>&#x43F;&#x43E;&#x43B;&#x443;&#x447;&#x438;&#x43C;</mi><mo>&#xA0;</mo><mo>:</mo><mo>&#xA0;</mo><mover><mi>x</mi><mo>&#xA8;</mo></mover><mo>=</mo><mfrac><mrow><mo>-</mo><msubsup><mi>m</mi><mi>p</mi><mn>2</mn></msubsup><msup><mi>l</mi><mn>2</mn></msup><mi>g</mi></mrow><mrow><msub><mi>I</mi><mi>p</mi></msub><mo>(</mo><msub><mi>m</mi><mi>c</mi></msub><mo>+</mo><msub><mi>m</mi><mi>p</mi></msub><mo>)</mo><mo>+</mo><msub><mi>m</mi><mi>c</mi></msub><msub><mi>m</mi><mi>p</mi></msub><msup><mi>l</mi><mn>2</mn></msup></mrow></mfrac><mi>&#x3B8;</mi><mo>+</mo><mfrac><mrow><msub><mi>m</mi><mi>p</mi></msub><msup><mi>l</mi><mn>2</mn></msup><mo>+</mo><msub><mi>I</mi><mi>p</mi></msub></mrow><mrow><msub><mi>I</mi><mi>p</mi></msub><mo>(</mo><msub><mi>m</mi><mi>c</mi></msub><mo>+</mo><msub><mi>m</mi><mi>p</mi></msub><mo>)</mo><mo>+</mo><msub><mi>m</mi><mi>c</mi></msub><msub><mi>m</mi><mi>p</mi></msub><msup><mi>l</mi><mn>2</mn></msup></mrow></mfrac><mi>F</mi><mspace linebreak=\"newline\"/><mfrac><mi>d</mi><mrow><mi>d</mi><mi>t</mi></mrow></mfrac><mfenced open=\"[\" close=\"]\"><mtable><mtr><mtd><mi>x</mi></mtd></mtr><mtr><mtd><mover><mi>x</mi><mo>&#x2D9;</mo></mover></mtd></mtr><mtr><mtd><mi>&#x3B8;</mi></mtd></mtr><mtr><mtd><mover><mi>&#x3B8;</mi><mo>&#x2D9;</mo></mover></mtd></mtr></mtable></mfenced><mo>=</mo><mfenced open=\"[\" close=\"]\"><mtable><mtr><mtd><mn>0</mn></mtd><mtd><mn>1</mn></mtd><mtd><mn>0</mn></mtd><mtd><mn>0</mn></mtd></mtr><mtr><mtd><mn>0</mn></mtd><mtd><mn>0</mn></mtd><mtd><mfrac><mrow><mo>-</mo><msubsup><mi>m</mi><mi>p</mi><mn>2</mn></msubsup><msup><mi>l</mi><mn>2</mn></msup><mi>g</mi></mrow><mrow><msub><mi>I</mi><mi>p</mi></msub><mo>(</mo><msub><mi>m</mi><mi>c</mi></msub><mo>+</mo><msub><mi>m</mi><mi>p</mi></msub><mo>)</mo><mo>+</mo><msub><mi>m</mi><mi>c</mi></msub><msub><mi>m</mi><mi>p</mi></msub><msup><mi>l</mi><mn>2</mn></msup></mrow></mfrac></mtd><mtd><mn>0</mn></mtd></mtr><mtr><mtd><mn>0</mn></mtd><mtd><mn>0</mn></mtd><mtd><mn>0</mn></mtd><mtd><mn>1</mn></mtd></mtr><mtr><mtd><mn>0</mn></mtd><mtd><mn>0</mn></mtd><mtd><mfrac><mrow><msub><mi>m</mi><mi>p</mi></msub><mi>l</mi><mi>g</mi><mo>(</mo><msub><mi>m</mi><mi>c</mi></msub><mo>+</mo><msub><mi>m</mi><mi>p</mi></msub><mo>)</mo></mrow><mrow><msub><mi>I</mi><mi>p</mi></msub><mo>(</mo><msub><mi>m</mi><mi>c</mi></msub><mo>+</mo><msub><mi>m</mi><mi>p</mi></msub><mo>)</mo><mo>+</mo><msub><mi>m</mi><mi>c</mi></msub><msub><mi>m</mi><mi>p</mi></msub><msup><mi>l</mi><mn>2</mn></msup></mrow></mfrac></mtd><mtd><mn>0</mn></mtd></mtr></mtable></mfenced><mfenced open=\"[\" close=\"]\"><mtable><mtr><mtd><mi>x</mi></mtd></mtr><mtr><mtd><mover><mi>x</mi><mo>&#x2D9;</mo></mover></mtd></mtr><mtr><mtd><mi>&#x3B8;</mi></mtd></mtr><mtr><mtd><mover><mi>&#x3B8;</mi><mo>&#x2D9;</mo></mover></mtd></mtr></mtable></mfenced><mo>+</mo><mfenced open=\"[\" close=\"]\"><mtable><mtr><mtd><mn>0</mn></mtd></mtr><mtr><mtd><mfrac><mrow><msub><mi>m</mi><mi>p</mi></msub><msup><mi>l</mi><mn>2</mn></msup><mo>+</mo><msub><mi>I</mi><mi>p</mi></msub></mrow><mrow><msub><mi>I</mi><mi>p</mi></msub><mo>(</mo><msub><mi>m</mi><mi>c</mi></msub><mo>+</mo><msub><mi>m</mi><mi>p</mi></msub><mo>)</mo><mo>+</mo><msub><mi>m</mi><mi>c</mi></msub><msub><mi>m</mi><mi>p</mi></msub><msup><mi>l</mi><mn>2</mn></msup></mrow></mfrac></mtd></mtr><mtr><mtd><mn>0</mn></mtd></mtr><mtr><mtd><mfrac><mrow><mo>-</mo><msub><mi>m</mi><mi>p</mi></msub><mi>l</mi></mrow><mrow><msub><mi>m</mi><mi>c</mi></msub><msub><mi>I</mi><mi>p</mi></msub><mo>+</mo><msub><mi>m</mi><mi>p</mi></msub><msub><mi>I</mi><mi>p</mi></msub><mo>+</mo><msub><mi>m</mi><mi>c</mi></msub><msub><mi>m</mi><mi>p</mi></msub><msup><mi>l</mi><mn>2</mn></msup></mrow></mfrac></mtd></mtr></mtable></mfenced><mi>F</mi></mstyle></math>","origin":"MathType for Microsoft Add-in"}

[9]

Можно представить полученную матрицу в виде пространства состояний:

Если мы сможем манипулировать матрицей A, то сможем и манипулировать решением нашей системы. Мы можем задать его как u = -Kx: то есть подать состояния системы на вход, создать обратную связь. Здесь K – вектор коэффициентов.

Подставив это выражение для u в исходное уравнение системы, получим:

(21)[2]

Видим, что через подбор коэффициентов матрицы K, мы можем получить любое нужное нам решение системы при условии, что система полностью управляема. По сути, матрица K это математическое представление того, как мы хотим, чтобы система реагировала на внешние воздействия. Существуют различные инструменты, которые позволяют нам реализовать это представление в реальной системе.

3.ПИД регулятор

Регулятор — это алгоритм или программный модуль, работающий на микроконтроллере, который управляет объектом, например, двигателем. Управляющее устройство, получая сигнал от регулятора, воздействует на объект. Датчик на объекте передает текущее значение параметра, что формирует обратную связь. Основная задача регулятора — сравнивать текущее значение с заданным (уставкой) и формировать управляющий сигнал, чтобы достигнуть цели. Регулятор обновляет сигнал через равные промежутки времени, называемые частотой дискретизации, что важно для точного управления..

Выходной сигнал ПИД регулятора определяется тремя слагаемыми:

В дискретной реализации же:

* Пропорциональная составляющая (P): вычисляется как разность между целевым и текущим значением (ошибка регулирования). Управляющее воздействие пропорционально этой ошибке. При приближении к заданному значению, управляющее воздействие будет уменьшаться и не позволит полностью компенсировать ошибку
* Интегральная составляющая (I): интегрирует ошибку регулирования по времени, что позволяет системе со временем устранять остаточную ошибку.

Дифференциальная составляющая (D): реагирует на скорость изменения ошибки, помогая системе предвосхищать будущие отклонения. D составляющая особенно полезна для быстродействующих систем, однако при наличии шумов на входе может вызвать нестабильность.

Общий выходной сигнал регулятора вычисляется как сумма всех трёх составляющих:

4.Список необходимых элементов

Для создания балансирующего робота потребуются следующие компоненты:

1. Микроконтроллер: Arduino Mega
2. Датчик угла наклона: MPU6050 (акселерометр и гироскоп), который используется для измерения угла наклона робота.
3. Двигатели: Два мотор-редуктора с энкодерами обеспечения достаточного крутящего момента и измерения скорости вращения колес.
4. Драйвер L298N: Для управления двумя двигателями при помощи ШИМ-сигнала.
5. Источник питания: Два аккумулятора 7.4 V, обеспечивающие стабильное напряжение и ток для всех компонентов
6. Два понижающих преобразователя напряжения LM2596S для питания микроконтроллера и драйвера двигателя.
7. Корпус: Легкий и прочный, чтобы удерживать все компоненты.
8. Соединительные элементы: гайки M3, шайбы M3,стойки М4
9. Фланцы для крепления двигателей: 2 шт.

Микроконтроллер: в качестве управляющего микроконтроллера в данном проекте мы выбрали плату Arduino Mega, так как она имеет больше пинов, настроенных на аппаратные прерывания, чем UNO и NANO, это позволит обрабатывать лучше обрабатывать одновременно показания обоих энкодеров в максимальном разрешении .

**Двигатели**: для построения самобалансирующегося робота могут использоваться шаговые двигатели и моторы постоянного тока с энкодерами, в данной реализации использованы двигатели с энкодерами.

**4.1 Драйвер мотора** (Motor Driver):

H-мост используется для изменения направления вращения двигателя постоянного тока, меняя полярность питающего напряжения. Он состоит из четырех выключателей. Когда замкнуты A1 и A2, ток течет справа налево, и двигатель вращается по часовой стрелке. Если замкнуты B1 и B2, ток течет слева направо, и двигатель вращается против часовой стрелки.

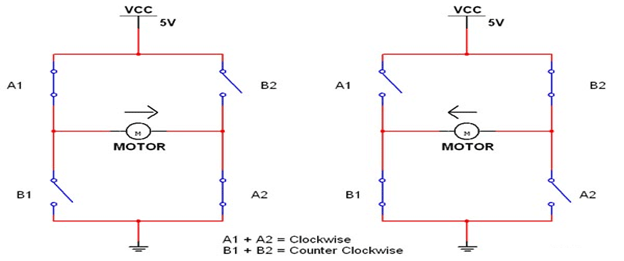


Рис.2 H-мост

Enable A и Enable B пины используются в качестве ШИМ пинов для управления скоростью моторов. Для подключения моторов постоянного тока (DC-моторов) служат порты:

* *OUT1 и OUT2 - мотор A*;
* *OUT3 и OUT4 – мотор B*

Управление электромоторами полностью независимо друг от друга.

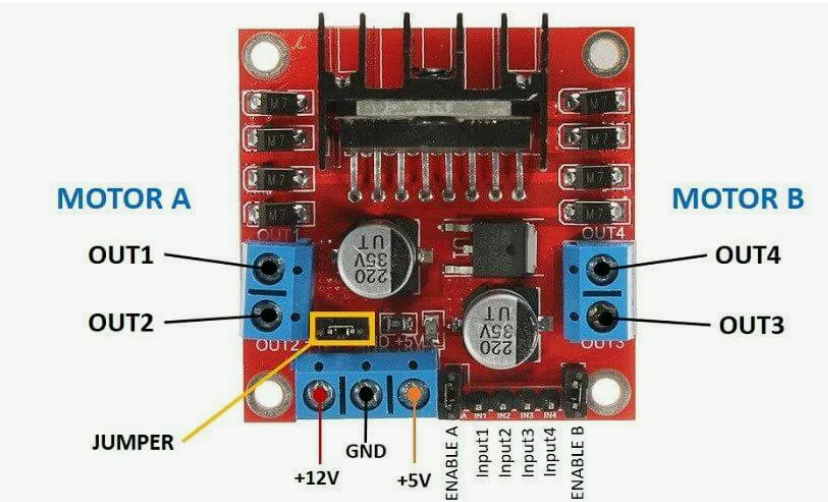


Рис.3 Драйвер L298M

**Колеса:** для построения самобалансирующегося робота должны иметь очень хорошее сцепление с полом и ни в коем случае они не должны проскальзывать при движении по полу. Также больший диаметр колес мог бы увеличить линейную скорость вращения колес, что могло бы добавить запас по управлению роботом.



Рис.4 Колесо для робота

**Батарея**: для питания драйвера двигателей необходимо напряжение 12V. В качестве аккумуляторов, были использованы две литий-ионных батареи 7.4 В соединенных последовательно



Рис.5 Аккумулятор Li-on

**Корпус робота**: для изготовления корпуса робота можно использовать листы картона, дерево, пластик и т.п. Шасси должно быть достаточно прочным и не изгибаться при балансировке. В качестве корпуса и для закрепления силовых элементов было использовано оргстекло.

**4.2 IMU-модуль MPU 6050(3х осевой гироскоп + акселерометр).**

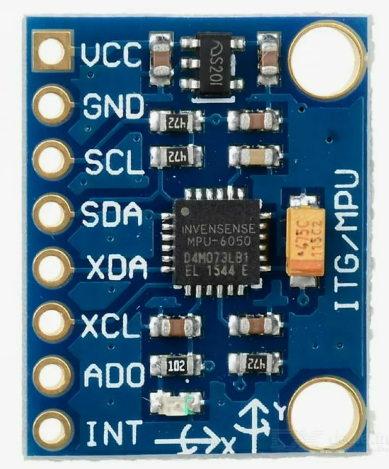


Рис.6 MPU-6050

MPU-6050, как показано на рис. 7, представляет собой 3-осевой акселерометр и 3-осевой гироскоп на одном кристалле. Для взаимодействия используется I2C-шина с помощью Arduino.

Гироскоп измеряет угловую скорость в градусах/секунду, а акселерометр — ускорение вдоль осей в метрах/секунду. Однако, оба датчика имеют свои ограничения: гироскоп накапливает ошибки при интегрировании, а акселерометр не может точно определить угол при движении с ускорением. Для повышения точности требуется калибровка обоих датчиков: калибровка акселерометра позволяет выставить «ноль» для вектора силы тяжести, а калибровка гироскопа уменьшает его «дрифт», то есть статическое отклонение в режиме покоя.

По отдельности акселерометр и гироскоп не могут выдавать точные показания угла отклонения, поэтому будем использовать специальные фильтры для объединения показаний и повышения точности.

4.2.1 Фильтрация данных IMU

Для того чтобы улучшить качество входных данных, можно воспользоваться фильтром низких частот. Он позволяет оставить только полезную низкочастотную часть сигнала и убрать высокочастотную часть , такую как шумы, возникающие от различного рода вибраций и приборов вокруг.

* **Фильтрация при помощи комплементарного фильтра.**

При объединении показаний акселерометра и гироскопа на основе комплементарного фильтра значение угла наклона рассчитывается по формуле:

где— расчетное значение угла наклона— значение угла наклона из предыдущего расчета, — коэффициент комплементарного фильтра (подбирается экспериментально от 0,001 до 0,05),— угловая скорость, Δt — время итерации,— угол склонения, рассчитанный по текущим показаниям акселерометра.

Итоговая величина является долевой суммой интегрированного значения показаний гироскопа и мгновенного значения показаний акселерометра. Доля показаний акселерометра мала, поскольку они умножаются на малую величину , равную 0,001–0,05, и основной вклад вносят показания гироскопа — они умножаются на величину (1 - ), равную 0,999–0,95. Однако благодаря акселерометру, компенсируются ошибки измерений, которые обязательно возникают и накапливаются, если пользоваться только гироскопом.{Мобильные датчики на базе ардуино}

* **Определение углов при помощи встроенного в датчик DMP (Digital Motion Processor).**

Из всех фильтров самый хороший результат даёт встроенный в датчик фильтр DMP , будем использовать в его в качестве основного. Ниже представлен код инициализации с использованием библиотеки “Simple\_MPU6050”[5]:

|  |
| --- |
| #include "Simple\_MPU6050.h"  #define MPU6050\_DEFAULT\_ADDRESS     0x68 // address pin low (GND), default for InvenSense evaluation board  Simple\_MPU6050 mpu;  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*                Setup and Loop                 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void setup() {    // initialize serial communication    Serial.begin(115200);    while (!Serial); // wait for Leonardo enumeration, others continue immediately    Serial.println(F("Start:"));    // Setup the MPU    mpu.begin();    mpu.SetAddress(MPU6050\_DEFAULT\_ADDRESS); //Sets the address of the MPU.    Serial.println(F("\n\n The MPU6050 comes with no calibration values set.\n"                     " This is what the default calibratin values are:\n"                     " "));    mpu.PrintActiveOffsets();      Serial.println(F("\n\n We are going to calibrate this specific MPU6050,\n"                     " Start by having the MPU6050 placed stationary on a flat surface to get a proper accellerometer calibration\n"                     " \t\t\t[Press Any Key]"));    while (Serial.available() && Serial.read()); // empty buffer    while (!Serial.available());                 // wait for data    while (Serial.available() && Serial.read()); // empty buffer again    mpu.CalibrateMPU();  // Calibrates the Gyro.    delay(1000);    Serial.println();    for (int i = 5; i<=30;i+=5){      Serial.println("Stabalization Delay");      delay (100);      mpu.CalibrateAccel(i); // Calibrates the Accellerometer.      mpu.CalibrateGyro(i);  // Calibrates the Gyro.    }    mpu.PrintActiveOffsets();    // Setup is complete!  }  void loop() {    delay(100);    Serial.println(F("\n\nThe above values are your calibration offsets.\n"                     " You can recheck the calibration using the last calibration as a starting poing for this specific MPU6050,\n"                     " This can further inprove the accuracy and may refine the calibration values\n"                     " Placed the MPU6050 stationary on a flat surface to get a proper accellerometer calibration\n"                     " \t\t\t[Press Any Key]"));    while (Serial.available() && Serial.read()); // empty buffer    while (!Serial.available());                 // wait for data    while (Serial.available() && Serial.read()); // empty buffer again    mpu.CalibrateMPU();  // Calibrates the MPU.    mpu.PrintActiveOffsets();  } |

**4.3 Мотор-редукторы с энкодерами**

Для того чтобы у робот имел запас по управлению необходимо, чтобы двигатели имели скорость вращения более 200 оборотов в минуту. Немаловажно убедиться, что кроме подходящего числа оборотов в минуту двигатель имеет крутящий момент, необходимый для приведения робота в движение. Исходя из данных требований был выбран двигатель JGB37-520B c номинальной скоростью 500 оборотов в минуту и максимальным крутящим моментом на валу, равным 2.2 кг/ см.

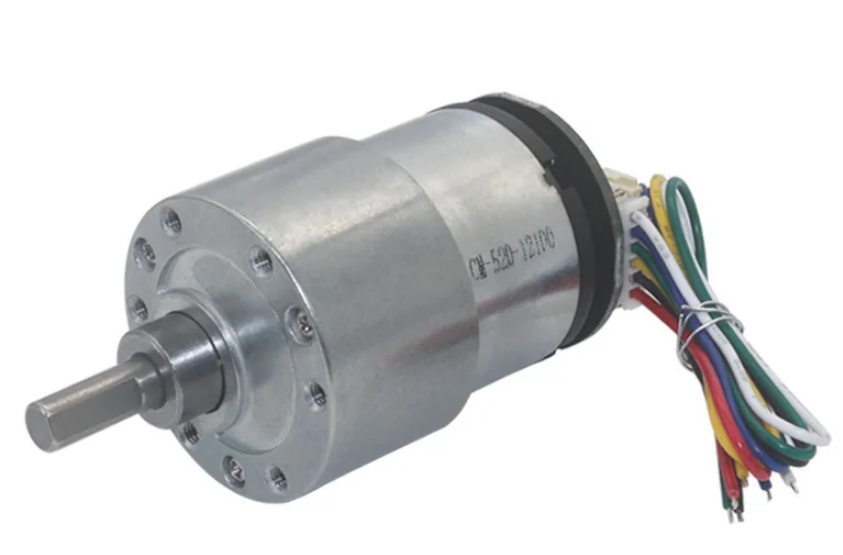


Рис.7 Мотор редуктор с энкодерами

Для получения обратной связи с по скорости используются квадратурные энкодеры.

Основные понятия:

* Квадратурный энкодер: это тип энкодера, который генерирует два сигнала (A и B) с фазовым сдвигом 90 градусов. Эти сигналы позволяют определить направление и скорость вращения.

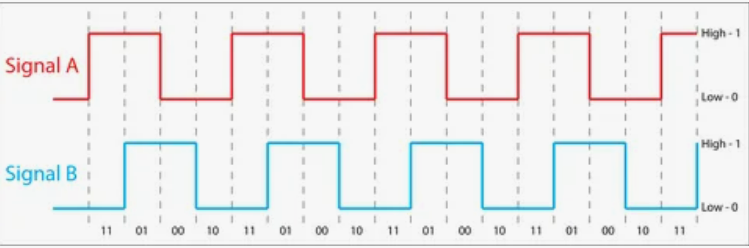


Рис.8 Квадратурные обмотки энкодеров.

Прерывания используются для немедленного реагирования на события (например, изменение состояния сигнала на пине) без необходимости постоянного опроса состояния пинов в основном цикле программы. Это позволяет значительно улучшить точность и эффективность обработки сигналов . Два сигнала (A и B): для каждого энкодера используются два пина (18 и 19 для первого энкодера, 2 и 3 для второго), к которым подключены сигналы A и B.Прерывания устанавливаются на оба сигнала каждого энкодера. Таким образом, любое изменение сигнала на пинах 18, 19, 2 или 3 вызывает вызов соответствующей функции обработчика прерывания (encoder1ISR или encoder2ISR).

Обработчики прерываний будут анализировать состояние сигналов A и B для определения направления вращения и увеличения или уменьшения показаний счетчика положения. В обычных условиях без использования прерываний или если прерывания используются только для одного из сигналов, разрешение ограничивается одним изменением положения на каждый цикл сигнала A. Однако, если устанавливать прерывания на оба сигнала (A и B) и реагировать на любое изменение состояния (и на фронте, и на спаде), можно обрабатывать до четырех изменений положения за каждый цикл сигнала A (два изменения на фронте и спаде сигнала A и два на фронте и спаде сигнала B),что позволит в четыре раза увеличить разрешение обработки.[8]

5.Схема подключения

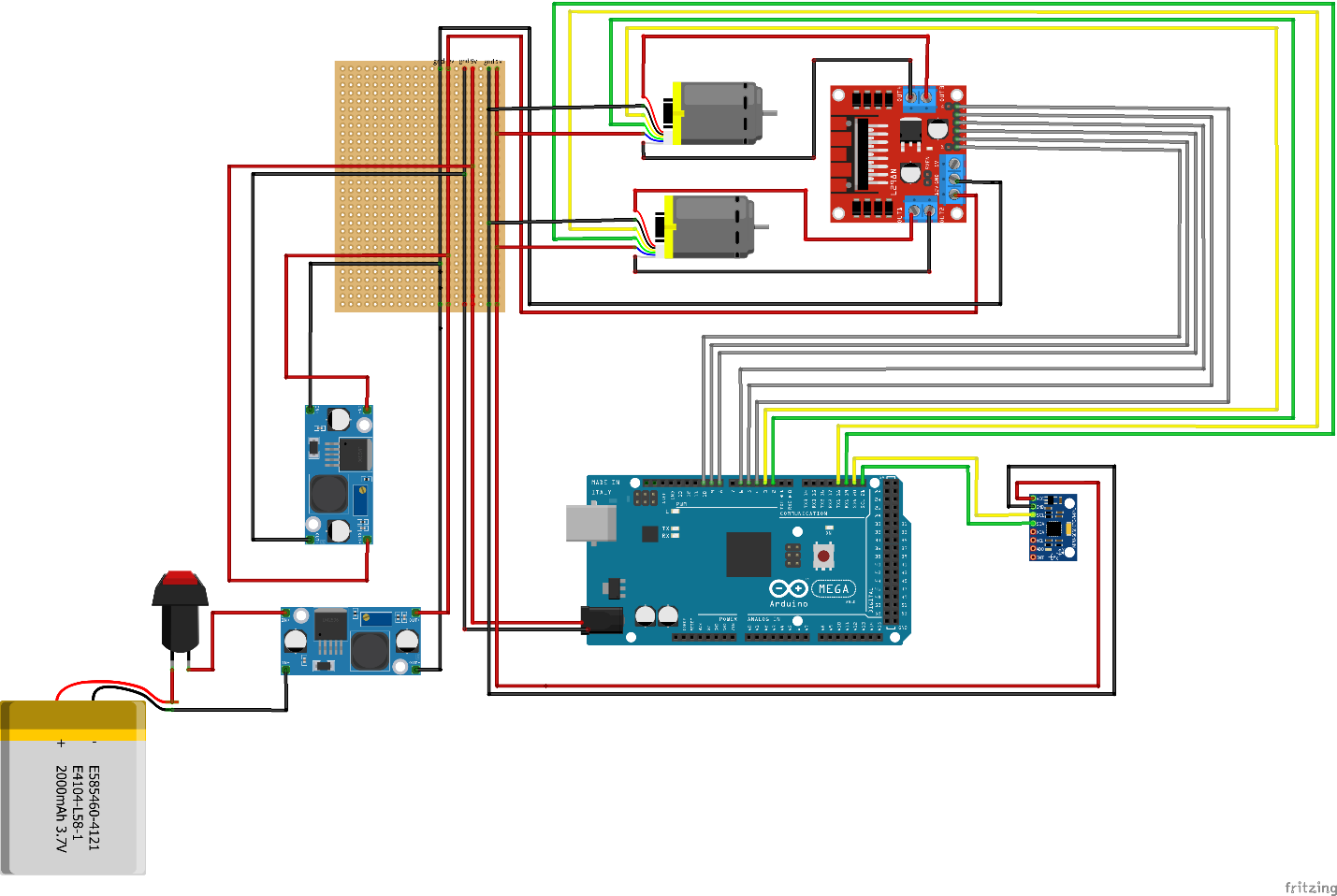


Рис.9 Электрическая схема робота

1. Мотор- редукторы с энкодерами:

Подключен к выходу мотора на драйвере L298N.

**Правый мотор**:

Метод подключения:

* Красный провод: мотор +
* Черный провод: земля энкодера -
* Желтый провод: Канал А на 2 пине Arduino
* Зеленый провод: Канал B на 3 пине Arduino
* Синий провод: Шина ( 5V)
* Белый провод: мотор-

**Левый мотор**: подключение питания аналогично правому двигателю. Каналы A и B подключены к пинам 18 и 19 на Arduino.

**Драйвер моторов (L298N):**

* + IN1, IN2, IN3, IN4: Подключены к цифровым пинам Arduino: 5,6,8,9.
  + ENA (Enable Motor A): Подключен к пину 4Arduino.
  + ENB (Enable Motor B): Подключен к пину 10 Arduino.
  + VCC: Подключен к положительному пину 12V шины .
  + GND: Подключен к общей земле (минус).

1. **MPU6050** (гироскоп и акселерометр) подключается к Arduino при помощи I2C шины:
   * VCC: Подключен к пину 5V на Arduino.
   * GND: Подключен к общему GND.
   * SDA: Подключен к пину 20 на Arduino (SDA).
   * SCL: Подключен к пину 21 на Arduino (SCL).
2. **Arduino Mega**:

Arduino питается через провод, подключенный к шине 9V. Для загрузки кода и отладки используется Serial порт через USB.

1. **Питание**:

В качестве питания используются две шины питания на 12V для питания драйвера двигателя и на 9V для питания Arduino. Также используется общая шина на 5 V, встроенная в плату расширения.

* + Литий- ионные аккумуляторы на 7.4 V подключены к входу первого DC-DC преобразователя(модуль снижения напряжения).
  + Выход первого DC-DC преобразователя подключен к 12 V шине питания.
  + Вход второго DC-DC преобразователя подключен 12V шине
  + Выход второго DC-DC подключен к 9V шине.

**6.Механическая часть.**

Размещение тяжелых аккумуляторов в верхней части робота способствует повышению центра тяжести, что делает движение робота более устойчивым, так как повышается его момент инерции. При увеличенном моменте инерции робот падает медленнее, что облегчает отслеживание начала падения и позволяет быстрее принять меры для его компенсации.

В качестве основного материала для корпуса робота было выбрано оргстекло. Это легкий и одновременно прочный материал, позволяющий сохранить жесткость конструкции во время испытаний. В качестве стоек были выбраны металлический прутки диаметром М4 и длиной 30 см, чтобы повысить центр тяжести. Для крепления элементов схемы к корпусу мною были использованы гайки и винты M3,а для закрепления различных частей корпуса между собой и стойками, мною были использованы винты и шайбы М4(Рис.11).Двигатели крепятся к корпусу при помощи специальных фланцев, жестко фиксирующих положение моторов(Рис.10).

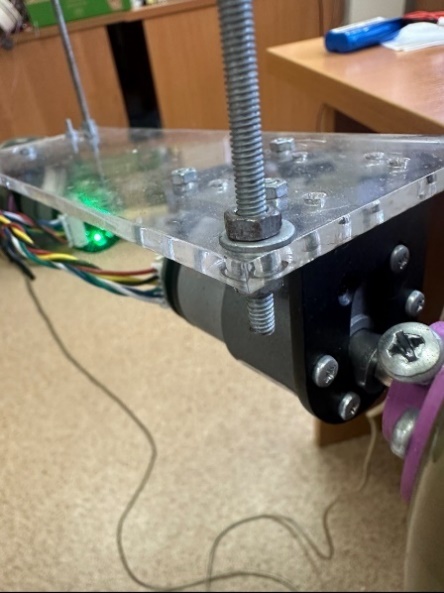
 

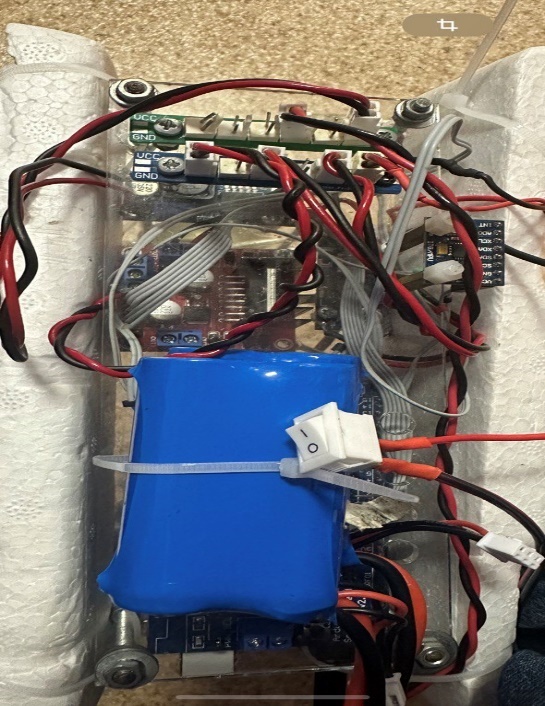
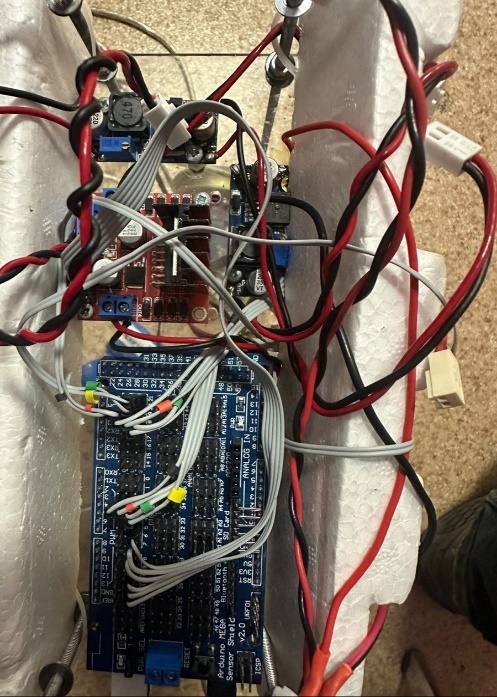
Рис.10 Крпепление двигателей к корпусу Рис.11Крпление стоек к корпусу 

Рис.12 Шина питания и аккумуляторы Рис.13 Силовая часть робота

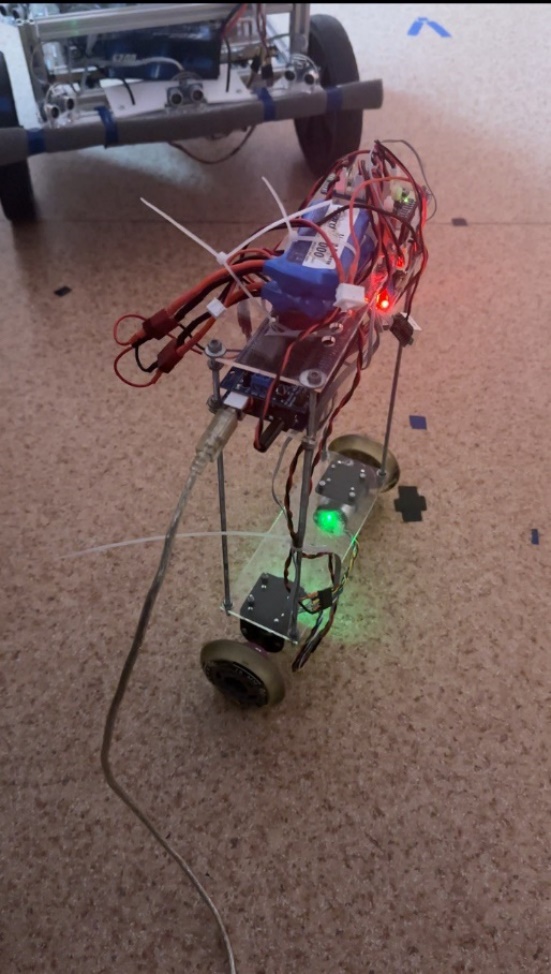


Рис.14 Балансирующий робот в сборе

**7.Алгоритм управления**

Робот получает значения ускорения и гироскопа, то есть углы наклона. Микроконтроллер обрабатывает эти данные, применяя ПИД-регулирование для получения правильного движения моторов. На основе данных, полученных контроллером от MPU-6050, контроллер управляет моторами, заставляя их двигаться вперед или назад, чтобы поддерживать положение робота. Робот должен находиться под углом 0° к оси y, как показано на рисунке 11(a). Когда корпус робота наклоняется в каком-либо направлении — по часовой стрелке или против часовой стрелки, — изменяется угол между осью y и корпусом, как показано на рисунках 11(b) и 11(c). Гироскопический датчик MPU-6050 определяет этот угол, и микроконтроллер начинает действия для поддержания его в заданном положении.[1]

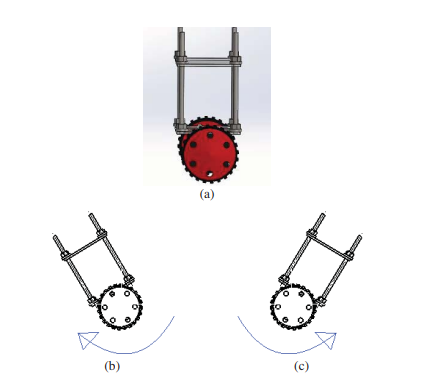


Рис.15 Балансирующий робот

7.1 Структурная схема

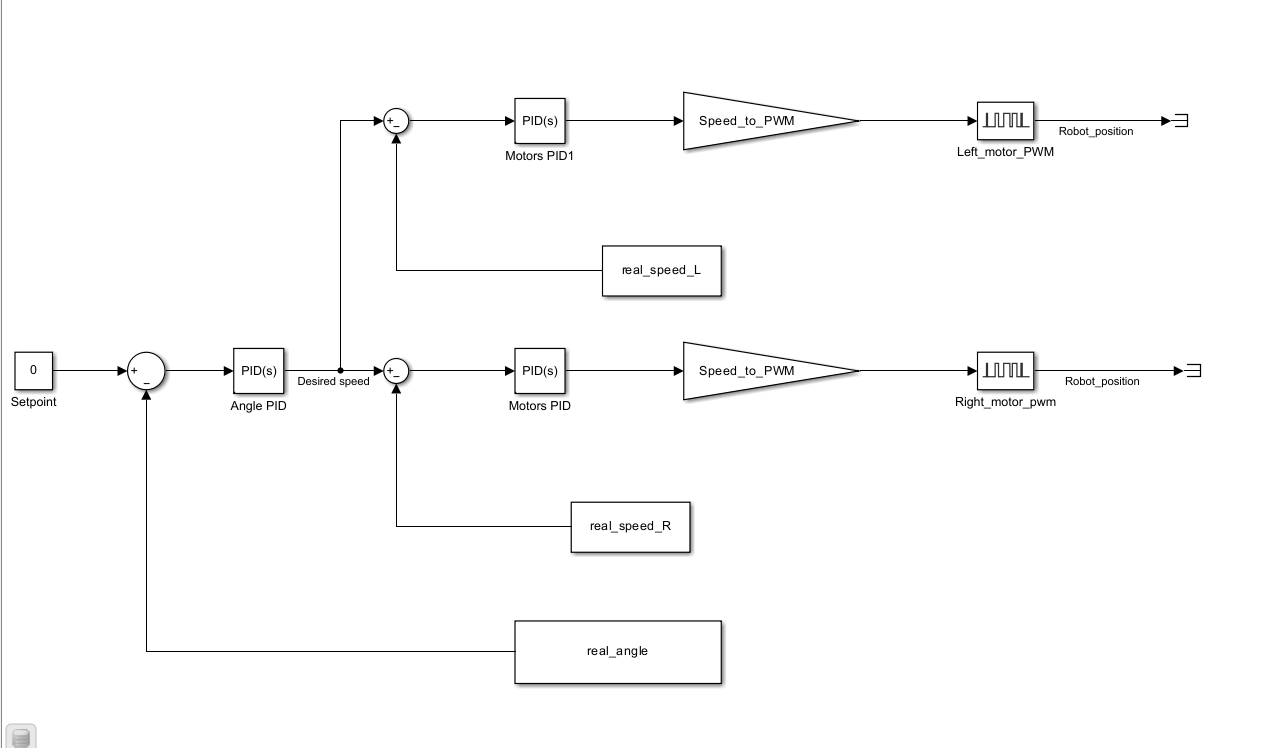


Рис.16 Структурная схема алгоритма балансировки.

Данная схема представляет собой систему управления балансирующим роботом с использованием каскадного ПИД-регулирования.

1.Контур управления углом наклона:

* + Целевое значение угла наклона задается как 0 (вертикальное положение).
  + Измеренный угол наклона (real\_angle) сравнивается с целевым значением.
  + Разница подается на ПИД-регулятор угла (Angle PID), который вычисляет требуемую скорость для балансировки.

1. Контур управления скоростью:
   * Выход Angle PID становится заданием скорости для обоих колес.
   * Измеренные скорости колес (real\_speed\_L и real\_speed\_R) сравниваются с заданием.
   * Разница подается на ПИД-регуляторы скорости (Motors PID и Motors PID1).
2. Управление двигателями:
   * Выходы ПИД-регуляторов скорости преобразуются в значения ШИМ (Speed\_to\_PWM).
   * Эти значения ШИМ подаются на двигатели (Left\_motor\_PWM и Right\_motor\_pwm).
3. Обратная связь:
   * Датчики измеряют реальный угол наклона и скорости вращения колес.
   * Эти значения подаются обратно в систему для замыкания контуров управления.

**7.2 Настройка коэффициентов ПИД- регуляторов**

**Контур управления скоростью моторов**.

Процесс настройки ПИД регулятора является итерационным и основывается на методе проб и ошибок. Зададим начальное значение Kp в диапазоне от 0.1 до 0.2 и будем наблюдать за поведением моторов. Характер движения двигателей должен быть плавным, поддерживая постоянную заданную скорость. Постепенно можно увеличивать пропорциональную компоненту и смотреть за откликом системы, не выводя ее за границу устойчивости. На следующем этапе необходимо отработать динамическую ошибку(ошибку по времени),для этого будем увеличивать интегральную компоненту и смотреть за поведением системы. Чем выше интегральная компонента, тем меньше будет накопленная ошибка. Дифференциальную компоненту в данном случае рекомендуется оставить равным 0.

**Внутренний контур**

Для настройки контура ПИД-регулятора угла (Angle\_PID) необходимо следовать следующим шагам:

1. Настройка пропорционального коэффициента (Kp):

Начните с постепенного увеличения коэффициента Kp до тех пор, пока робот не начнет балансировать. На этом этапе робот еще не сможет стоять на месте, но он не должен падать. Важно, чтобы движение было относительно стабильным, без резких отклонений.

1. Настройка интегрального коэффициента (Ki):

После того как робот начнет балансировать, начинайте увеличивать значение коэффициента Ki. Увеличивайте его до тех пор, пока колебания робота не станут устойчивыми или вовсе не исчезнут. Помните, что слишком большое значение Ki может привести к тому, что робот будет менее устойчивым к внешним возмущениям. В таких случаях даже небольшие воздействия могут вызвать незатухающие колебания, которые нарушат его равновесие.

1. Настройка дифференциального коэффициента (Kd):

После настройки Kp и Ki переходите к увеличению коэффициента Kd. Это поможет снизить скорость изменения ошибок и сделать систему более стабильной. Однако следите за тем, чтобы робот не начал дрожать или трястись, что может указывать на избыточное значение Kd.

**7.3 Необходимость среднего контура управления:**

В реальных системах практически невозможно найти два абсолютно идентичных двигателя. Даже двигатели одной модели могут иметь небольшие различия в характеристиках из-за производственных допусков, износа или условий эксплуатации. Разные характеристики двигателей могут привести к тому, что при одинаковом входном сигнале они будут вращаться с разной скоростью. Индивидуальные ПИД-регуляторы для каждого двигателя (Motors PID и Motors PID1) позволяют компенсировать эти различия. Они обе спечивают, чтобы реальная скорость каждого колеса (real\_speed\_L и real\_speed\_R) максимально точно соответствовала заданию, полученному от контура управления углом.

В теоретической ситуации, когда оба двигателя абсолютно идентичны, можно было бы обойтись без отдельных контуров управления скоростью. В этом случае выход Angle\_PID мог бы напрямую преобразовываться в сигналы ШИМ для обоих двигателей.

**7.4 Метод масштабирования коэффициентов**

Для упрощения сопоставления данных был предложен метод масштабирования коэффициентов. Поскольку максимальная скорость системы составляет 5000 тиков в секунду при максимальном значении ШИМ 255, значения угла отклонения, варьирующиеся от -90 до 90 градусов, были масштабированы в диапазон от -5000 до 5000. В дальнейшем, эти масштабированные значения сопоставляются с реальной скоростью и переводятся обратно в диапазон ШИМ от -255 до 255. Этот метод позволяет учесть различие в масштабах и повысить эффективность работы регуляторов, обеспечивая более точное и адекватное управление системой.

**7.5 Компенсация нелинейности щеточных моторов**

Щеточные моторы имеют нелинейную зависимость между подаваемым напряжением (или ШИМ-сигналом) и скоростью вращения. Эта нелинейность особенно заметна при низких скоростях вращения.

Существует минимальное значение ШИМ, ниже которого двигатель не начинает вращение. Это явление называют "мертвой зоной». Причины возникновения "мертвой зоны": статическое трение в подшипниках и щетках мотора, инерция ротора и механической системы, противо-ЭДС при низких скоростях. Минимальное значение ШИМ, при котором колеса начинают вращаться, необходимо определять экспериментально для каждого конкретного мотора. После определения "мертвой зоны" необходимо внести коррекцию в алгоритм управления ,например, добавить смещения к выходному сигналу ПИД-регулятора.

* 1. Задача движения робота вперед-назад

После того как робот достиг устойчивого равновесия в вертикальном положении, необходимо решить задачу его движения вперед и назад. Существуют три основных подхода для реализации этой задачи:

* 1. **Добавление или вычитание небольшой дельты к желаемой скорости моторов:**
  + Этот подход заключается в том, чтобы корректировать скорость моторов, добавляя или вычитая небольшое значение (дельту) к желаемой скорости. Выход регулятора будет постоянно немного больше необходимого, что будет заставлять робота двигаться в одну из сторон. Однако этот метод имеет значительные недостатки, поскольку робот постоянно стремится восстановить равновесие, его движения могут стать рывковыми и нестабильными.

**8.2 Добавление дельты к сетпоинту угла наклона:**

* + В этом методе корректируется целевой угол наклона робота (сетпоинт). Добавление небольшой дельты к углу наклона заставляет робота двигаться в определенном направлении, однако данный способ ограничен допустимым углом отклонения. При небольшом отклонении робот не будет двигаться ни в одну из сторон, а будет стараться поддерживать равновесие. Большой же угол будет заваливать робота в одну из сторон, что сделает его управление невозможным.
  1. **Комбинированный подход:**
  + Оптимальным является третий метод, который представляет собой комбинацию двух предыдущих. В этом случае дельта добавляется как к скорости моторов, так и к сетпоинту угла наклона. Такой подход позволяет уменьшить рывки, характерные для первого метода, и смягчить чрезмерный крен, возникающий во втором методе. Эта комбинация обеспечивает более плавное и контролируемое движение робота вперед и назад, позволяя лучше управлять его устойчивостью и маневренностью.

Для торможения робот постепенно уменьшает дельту как для сетпоинта угла наклона, так и для скорости моторов. Этот подход помогает избежать резкого торможения в условиях ограниченного запаса устойчивости.

9.Заключение.

После ряда тестов и калибровок, двухколесный робот смог самостоятельно удерживать равновесие. Для этого потребовались правильно настроенные значения PID, передаваемые микроконтроллеру, и точные измерения с IMU-датчика. Также робот умеет перемещаться вперед и назад с определенной скоростью.

Несмотря на простоту и надежность ПИД-регулятора, существуют более сложные и адаптивные методы управления, которые могут значительно улучшить производительность робота в будущем. Возможной модернизацией могла бы быть замена ПИД регулятора на контроллер нечеткой логики или линейно-квадратичный регулятор (LQR). Кроме того, в будущем возможно внедрение нейронной сети, использующей подход обучения с подкреплением (Reinforcement Learning).

В дополнение к совершенствованию системы управления, планируется также реализация новых функциональных возможностей робота. В частности, в будущем будет добавлена функция выполнения поворотов а также адаптация робота для движения по неровной поверхности. Это потребует дополнительных доработок в системе управления и, возможно, внесения изменений в конструкцию робота.

10. Приложение

10.1.Код проекта

|  |
| --- |
| #include <PID\_v1.h>  #include "Wire.h"  #include "Simple\_MPU6050.h"  #include "AccelMotor.h"  #define MPU6050\_DEFAULT\_ADDRESS     0x68  #define MIN\_ABS\_SPEED 120  #define MPU\_UPDATE\_INTERVAL 20 // ms  Simple\_MPU6050 mpu;  // orientation/motion vars  Quaternion q;  VectorFloat gravity;  float ypr[3] = { 0, 0, 0 };  float xyz[3] = { 0, 0, 0 };  float speedA = 0;  float speedB = 0;  double setpoint = 0; // Желаемый угол наклона  double input = 0; // Измеренный угол наклона  double output = 0; // Выход PID-регулятора  double Kp\_angle = 32.2;  double Ki\_angle = 0.4;  double Kd\_angle = 0.1;  double speedDelta = 0;  const double maxSpeedDelta = 400; // Максимальное значение дельты  const double speedIncrement = 25; // Шаг изменения дельты скорости при каждом нажатии клавиши  const float MAX\_OUTPUT = 5000;  double Kp\_balance = 16.2, Ki\_balance = 200, Kd\_balance = 0.1;  double Kp\_drive = 15.0, Ki\_drive = 130, Kd\_drive = 0.05;  bool isDrivingMode = false; // false - балансировка, true - езда  double driveSetpoint = 0;  double balanceSetpoint = 0;  const float delta\_angle = 0.05;  unsigned long lastMPUUpdate = 0;  String inputString = "";  boolean stringComplete = false;  // Определение моторов  AccelMotor motor1(DRIVER3WIRE, 4, 5, 6, HIGH);  AccelMotor motor2(DRIVER3WIRE, 8, 9, 10, HIGH);  PID anglePID(&input, &output, &setpoint, Kp\_angle, Ki\_angle, Kd\_angle, DIRECT);  // Переменные для хранения счетчиков энкодеров  volatile long encoder1Count = 0;  volatile long encoder2Count = 0;  // Функции обработчики прерываний  void encoder1ISR() {    encoder1Count += motor1.getState();  }  void encoder2ISR() {    encoder2Count += motor2.getState();  }  // Callback функция для MPU  void Print\_Values (int16\_t \*gyro, int16\_t \*accel, int32\_t \*quat) {    mpu.GetQuaternion(&q, quat);    mpu.GetGravity(&gravity, &q);    mpu.GetYawPitchRoll(ypr, &q, &gravity);    mpu.ConvertToDegrees(ypr, xyz);      input = xyz[1]\*5000/90; // Используем pitch angle как входной сигнал для PID  }  void setup() {      Wire.begin();      Serial.begin(115200);      while (!Serial);      Serial.println(F("Initializing MPU..."));      mpu.begin();      mpu.Set\_DMP\_Output\_Rate\_Hz(100);  // Устанавливаем частоту обновления DMP на 100 Гц      mpu.CalibrateMPU();      mpu.load\_DMP\_Image();      mpu.on\_FIFO(Print\_Values);      Serial.println(F("Setup starting..."));      anglePID.SetMode(AUTOMATIC);      anglePID.SetSampleTime(10);      anglePID.SetOutputLimits(-5000, 5000);        // Инициализация PID для режима балансировки      setBalanceMode();      // Настройка мотора 1      motor1.setRatio(19 \* 11);      motor1.setDt(10);      motor1.setMaxSpeed(6000);      motor1.setAcceleration(500);      motor1.setMinDuty(50);      motor1.setRunMode(PID\_SPEED);      motor1.kp = 0.1;      motor1.ki = 0.1;      motor1.kd = 0.0;      // Настройка мотора 2      motor2.setRatio(19 \* 11);      motor2.setDt(10);      motor2.setMinDuty(50);      motor2.setStopZone(10);      motor2.setRunMode(PID\_SPEED);      motor2.kp = 0.1;      motor2.ki = 0.1;      motor2.kd = 0.0;      // Настройка прерываний для энкодеров      attachInterrupt(digitalPinToInterrupt(18), encoder1ISR, CHANGE);      attachInterrupt(digitalPinToInterrupt(19), encoder1ISR, CHANGE);      attachInterrupt(digitalPinToInterrupt(2), encoder2ISR, CHANGE);      attachInterrupt(digitalPinToInterrupt(3), encoder2ISR, CHANGE);      inputString.reserve(200);  // Резервируем память для входящей строки      Serial.println("Angle,PID\_Output,Motor1\_Speed,Motor1\_PWM,Motor2\_Speed,Motor2\_PWM");      Serial.println("Enter P, I, D followed by value (e.g. P50) or S followed by setpoint value between -5 and 5 (e.g. S1.57)");  }    void loop() {      motor1.tick(encoder1Count);      motor2.tick(encoder2Count);      unsigned long currentMillis = millis();      if (currentMillis - lastMPUUpdate >= MPU\_UPDATE\_INTERVAL) {          lastMPUUpdate = currentMillis;          if (mpu.dmp\_read\_fifo(false)) {              anglePID.Compute();              output = constrain(output, -MAX\_OUTPUT, MAX\_OUTPUT);              motor1.setTargetSpeedDeg(output);              motor2.setTargetSpeedDeg(output);              if (stringComplete) {                  processCommand(inputString);                  inputString = "";                  stringComplete = false;              }              // Вывод отладочных данных              Serial.print(xyz[1]);  // Pitch angle              Serial.print(",");              Serial.print(output);              Serial.print(",");              Serial.print(motor1.getSpeedDeg());              Serial.print(",");              Serial.print(motor1.getDuty());              Serial.print(",");              Serial.print(motor2.getSpeedDeg());              Serial.print(",");              Serial.println(motor2.getDuty());          }      }  }  void serialEvent() {      while (Serial.available()) {          char inChar = (char)Serial.read();          inputString += inChar;          if (inChar == '\n') {              stringComplete = true;          }      }  }  void processCommand(String command) {      command.trim();  // Удаляем пробелы в начале и конце строки        if (command.length() < 2) return;  // Игнорируем слишком короткие команды        char type = command.charAt(0);      float value = command.substring(1).toFloat();        switch(type) {          case 'P':              Kp\_angle = value;              anglePID.SetTunings(Kp\_angle, Ki\_angle, Kd\_angle);              Serial.print("Kp set to: ");              Serial.println(Kp\_angle);              break;          case 'I':              Ki\_angle = value;              anglePID.SetTunings(Kp\_angle, Ki\_angle, Kd\_angle);              Serial.print("Ki set to: ");              Serial.println(Ki\_angle);              break;          case 'D':              Kd\_angle = value;              anglePID.SetTunings(Kp\_angle, Ki\_angle, Kd\_angle);              Serial.print("Kd set to: ");              Serial.println(Kd\_angle);              break;          case 'S':              if (value >= -5 && value <= 5) {                  balanceSetpoint = value \* 5000 / 90;  // Преобразуем в то же масштабирование, что и input                  if (!isDrivingMode) {                      setpoint = balanceSetpoint;                  }                  Serial.print("Setpoint set to: ");                  Serial.println(value);              } else {                  Serial.println("Setpoint must be between -5 and 5");              }              break;          case 'F':          case 'f':              if (!isDrivingMode) {                  setDriveMode();              }              speedDelta += speedIncrement;              if (speedDelta > maxSpeedDelta) speedDelta = maxSpeedDelta;              driveSetpoint = balanceSetpoint + delta\_angle; // Добавляем дельту к сетпоинту              setpoint = driveSetpoint;              Serial.println("Moving forward");              break;          case 'B':          case 'b':              if (!isDrivingMode) {                  setDriveMode();              }              speedDelta -= speedIncrement;              if (speedDelta < -maxSpeedDelta) speedDelta = -maxSpeedDelta;              driveSetpoint = balanceSetpoint + delta\_angle; // Уменьшаем дельту от сетпоинта              setpoint = driveSetpoint;              Serial.println("Moving backward");              break;          case 'M': // Команда смены режима на балансировку              if (isDrivingMode) {                  setBalanceMode();              }              break;          default:              Serial.println("Unknown command");      }  }  // Функции для плавного переключения режимов  void setBalanceMode() {      isDrivingMode = false;      balanceSetpoint = xyz[1] \* 5000 / 90;      setpoint = balanceSetpoint;      speedDelta = 0;      Kp\_angle = Kp\_balance;      Ki\_angle = Ki\_balance;      Kd\_angle = Kd\_balance;      anglePID.SetTunings(Kp\_angle, Ki\_angle, Kd\_angle);      Serial.println("Switched to Balance Mode");  }  void setDriveMode() {      isDrivingMode = true;      Kp\_angle = Kp\_drive;      Ki\_angle = Ki\_drive;      Kd\_angle = Kd\_drive;      anglePID.SetTunings(Kp\_angle, Ki\_angle, Kd\_angle);      driveSetpoint = balanceSetpoint; // Синхронизируем setpoint с текущим углом      Serial.println("Switched to Drive Mode");  } |

**10.2 Сторонние библиотеки**

<PID\_v1.h>- библиотека для работы с ПИД регулятором;

"Wire.h"-необходимая библиотека для обмена данными по I2C шине;

"Simple\_MPU6050.h"-библиотека для работы с MPU-6050

"AccelMotor.h"- библиотека для управления двигателем при помощи энкодера[6]

11. Список использованных источников

1. Rohilla, Yogesh & Shekhawat, Anmol. (2020). Design and Control of Two-wheeled Self-Balancing Robot using Arduino. 10.1109/ICOSEC49089.2020.9215421.
2. IOP Conf. Series: Materials Science and Engineering 263 (2017) 052007 doi:10.1088/1757-899X/263/5/052007.
3. Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control. Steven L. Brunton, J. Nathan Kutz
4. <https://github.com/mahowik/BalancingWii>
5. <https://github.com/ZHomeSlice/Simple_MPU6050>
6. <https://github.com/GyverLibs/AccelMotor>
7. 2020). Arduino - Software. Available: <https://www.arduino.cc/en/Main/Software>
8. <https://tsibrov.blogspot.com/2017/10/blog-post.html>
9. https://faculty.washington.edu/chx/teaching/me547/pendulum\_on\_a\_cart\_prob.pdf
10. J. Qiu et al., "Two-wheeled self-balancing robot modeling and
11. nonlinear control," in 2017 14th International Conference on
12. Ubiquitous Robots and Ambient Intelligence (URAI), 2017, pp. 734-
13. 739: IEEE.