Рогозин К.А,

Кондратенко К.С.

«Эту тему проходят на третьем курсе, вы сейчас в третьем классе, но я тороплюсь»

Уральские пельмени

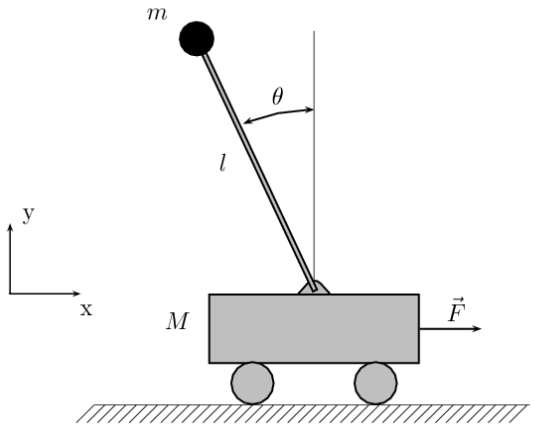
Балансирующий робот на arduino.

1. Теоретическая модель

С началом нового тысячелетия робототехника активно внедряется во все сферы деятельности человека, в том числе и в обыденную жизнь. Традиционный робот, взаимодействующий с человеком, имеет широкое основание и передвигается с малыми ускорениями во избежание потери устойчивости. Центр масс подобных мобильных колесных роботов стараются расположить как можно ближе к поверхности, по которой осуществляется движение, при этом для устойчивости у робота всегда имеется как минимум 3 точки опоры. Двухколесные балансирующие роботы имеют меньшее основание за счет отсутствия требования статической устойчивости. Колесная пара позволяет совершать поворот на месте, что дает им большую мобильность.

С физической точки зрения двухколесный балансирующий робот представляет собой перевернутый маятник, который имеет центр масс выше своей точки опоры, закреплённый на конце жёсткого стержня. Часто точка опоры закрепляется на тележке, которая может перемещаться по горизонтали. В то время как нормальный маятник устойчиво висит вниз, обратный маятник по своей природе неустойчивый и должен постоянно балансировать, чтобы оставаться в вертикальном положении, с помощью применения крутящего момента к опорной точке или при перемещении точки опоры по горизонтали, как части обратной связи системы. Простейшим демонстрационным примером может являться балансировка карандаша на конце пальца.

Перемещая опору маятника можно заставить его колебаться в перевернутом состоянии, поэтому он называется обратным. Стрелка внизу показывает скорость и направление движения тележки, на которой закреплен шарнир маятника



Схематическое изображение перевёрнутого маятника на тележке. Стержень не обладает массой. Массу тележки и массу шара на конце стержня обозначим через M и m. Стержень имеет длину l.

Действительно, если представить, как управлять длинной рейкой, поддерживая ее, стоящую на руке в вертикальном положении, то можно понять, что для поддержания баланса нужно реагировать на наклоны и скорости поворота рейки, задавая не силу, а положение, ускорение и скорость руки.

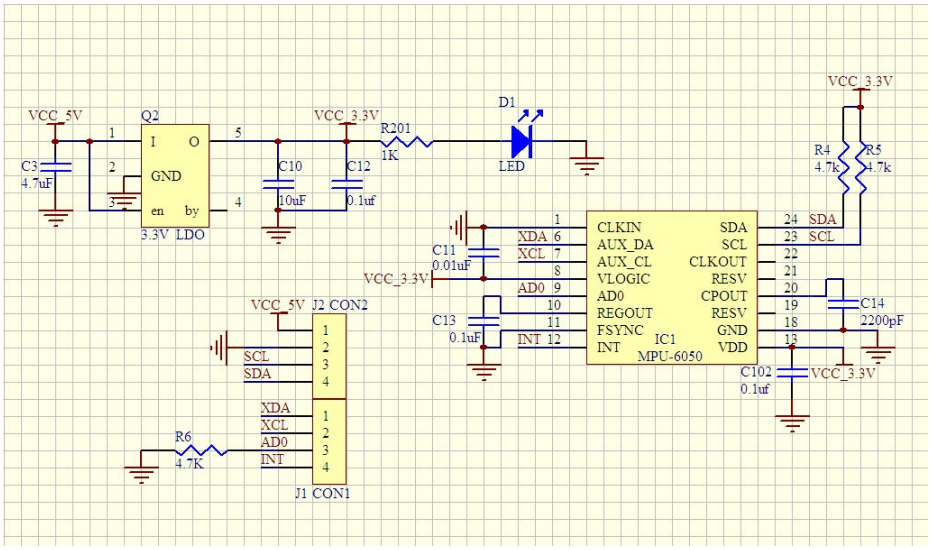
Примеры использования перевернутого маятника - это сегвей и гироскутер которые перевозят человека только на двух колесах не опрокидываясь. Балансирующего робота можно сделать и самостоятельно, владея лишь прямыми руками и некоторыми знаниями программирования.

2. Сборка и компоненты

Для реализации своего балансирующего робота необходимы: корпус, гироскоп и акселерометр, два двигателя с колесами, аккумуляторы и микроконтроллер, который будет управлять роботом.

Приступая к сборке, необходимо пояснить о расположении компонентов на корпусе. Из теории следует, что чем выше центр масс, тем легче стабилизировать систему, поэтому самые тяжелые компоненты, а это батарейный отсек и плата управления, необходимо поместить в верхнюю часть.

Рассмотрим датчик, используемый в проекте. Микросхема MPU6050 содержит на борту, как акселерометр, так и гироскоп, а помимо этого еще и температурный сенсор. MPU6050 является главным элементом модуля GY-531. Помимо этой микросхемы на плате модуля расположена необходимая обвязка MPU6050, в том числе подтягивающие резисторы интерфейса I2C, а также стабилизатор напряжения на 3,3 вольта с малым падением напряжения (при питании уже в 3,3 вольта на выходе стабилизатора будет 3 ровно вольта) с фильтрующими конденсаторами. Ну и бонусом на плате распаян SMD светодиод с ограничивающим резистором как индикатор питающего напряжения. Размер платы модуля GY-521 10 x 20 мм.



На плате имеется 8 контактов:

- VCC — положительный контакт питания;
- GND — земля;
- SDA — линия данных I2C;
- SCL — линия синхроимпульсов I2C;
- INT — настраиваемое прерывание;
- AD0 — I2C адрес; по-умолчанию AD0 подтянут к земле, поэтому адрес устройства — 0x68; если соединить AD0 к контактом питания, то адрес изменится на 0x69;
- XCL, XDA — дополнительный I2C интерфейс для подключения внешнего магнитометра.

Характеристики модуля MPU6050:

- напряжение питания: от 3,5 до 6 В;
- потребляемый ток: 500 мкА;
- ток в режиме пониженного потребления: 10 мкА при 1,25 Гц, 20 мкА при 5 Гц, 60 мкА при 20 Гц, 110 мкА при 40 Гц;
- диапазон: ± 2, 4, 8, 16g;
- разрядность АЦП: 16;
- интерфейс: I2C (до 400 кГц).

Разберемся, как можно использовать датчики акселерометра и гироскопа. Гироскоп выдает значения мгновенной угловой скорости с разрешением, заданным в настройках, например 2000 градусов в секунду. Если прошить микроконтроллер и смотреть на получаемые данные, то увидим только нули. Если начать крутить датчик, то получим мгновенные значения угловой скорости. Заметьте, что скорость мы получаем в градусах в секунду, а это значит, что линейные скорости не влияют на эти показания - показания будут изменяться только при повороте датчика в пространстве. Далее с помощью этих данных можно получить ориентацию объекта в пространстве. Для этого нужно получить мгновенное значение угловой скорости и умножить его на промежуток времени между

опросами датчика гироскопа. Пример разрешение 2000 градусов в секунду, промежуток между опросами датчика 0,1 секунда, значение мгновенной скорости 300, значит 300*0,1=30 - за это время ось гироскопа была повернута на 30 градусов. Далее каждое полученное значение нужно сложить с предыдущим. Если ось двигалась в одном направлении - значение 30 градусов, если в другом, то -30, таким образом, при возвращении датчика в исходное положение всегда (в идеале) будет 0, при отклонении от исходного положения, при выполнении вышеописанных действий, получим угол отклонения. Обрабатывая углы трех осей гироскопа можно получить ориентацию объекта в пространстве.

С акселерометром все проще. Измеряя ускорения трех осей датчика можно получить данные, преобразуя их с помощью геометрии, по которым можно также получить ориентацию объекта в пространстве. Помимо этого акселерометр измеряет линейные ускорения, то есть ориентация объекта может искажаться при движении датчика в линейных направлениях. Также с помощью акселерометра можно определять движение объекта или его столкновение. Например, детектировать падение объекта или толчок о преграду, чтобы обходить это.

Таким образом, при интегрировании состояния угла положения, также интегрируется и погрешность - при длительном использовании можно получить уже абсолютно неправильные значения. Поэтому часто гироскоп используют в паре с акселерометром, образуя, в простом варианте, альфа-бета фильтр или комплементарный фильтр.

3. Алгоритм работы

Теперь необходимо разобраться с алгоритмом работы балансирующего робота. Микроконтроллер получает данные с гироскопа и сравнивает их с заданным углом вертикали робота, т.е. того который возвращает гироскоп, при котором стоит робот, не падая. Если данные различаются, то микроконтроллер подает питание на двигатели, чтобы скорректировать угол наклона. Двигатели начинают вращаться в направлении падения робота, таким образом, что основание оказывается под центром тяжести робота, и он перестает падать. Но двигатели инерционны и каждый раз угол наклона разный, поэтому он может попросту перескочить нужное положение, что снова приводит к падению. Поэтому приходиться повторять такую операцию снова, но в другом направлении. Чтобы этого не происходило, необходимо каждый раз считывать угол и корректировать мощность двигателя. Для этого используют алгоритм пропорционально-интегрально-дифференцирующего (ПИД) регулирования.

4. ПИД-регулятор

С помощью настройки ПИД-регулятора (PID-controller) мы можем скорректировать переходный процесс так, как нам нужно для решения своей задачи.

Пример:

ПИД-регулятор открывает и закрывает регулирующий вентиль на горячей трубе так, чтобы из крана текла вода с температурой +40°C с погрешностью плюс-минус 2 градуса.

Регулятор вычисляет рассогласование (ошибку) - отклонение реальной температуры (например, +20°C) от заданного значения (+40°C) и решает – когда и насколько необходимо приоткрыть горячий вентиль, чтобы температура повысилась на 20°C. Реальную (фактическую) температуру регулятор узнаёт с помощью датчика температуры (обратная связь), а заданную температуру (уставку) ему сообщает оператор, например, набирая число «40» на своём ПК.

Чтобы настроить ПИД-регулятор, необходимо подобрать правильную комбинацию трёх коэффициентов:

- Пропорционального – Kp
- Интегрального – Ki
- Дифференциального – Kd

Могут использоваться и более простые - П и ПИ-регуляторы.

Формула ПИД-регулятора:

$$o(t) = P + I + D = Kp\ e(t) + Ki \int e(t)dt + Kd\ de(t)/dt \ [1]$$

- o(t) – выходной сигнал;
- P – пропорциональная составляющая;
- I – интегрирующая составляющая;
- D – дифференцирующая составляющая;
- Kp, Ki, Kd – коэффициенты пропорционального, интегрирующего, дифференцирующего звеньев;
- e(t) – ошибка рассогласования.

Чем больше Пропорциональный коэффициент, тем выше быстродействие, но меньше запас устойчивости. Но! простой П-регулятор не может полностью отработать рассогласование, т.е. всегда работает с ошибкой.

ПИ-регулятор позволяет избавиться от статической (установившейся) ошибки, но, чем больше Интегральный коэффициент, тем больше перерегулирование (динамическая ошибка).

ПИД-регулятор позволяет нам уменьшить перерегулирование, но, чем больше Дифференциальный коэффициент, тем больше погрешность из-за влияния шумов.

Если шумы идут по каналу обратной связи, то мы можем их отфильтровать с помощью фильтра низкой частоты, но чем больше постоянная этого фильтра, тем медленнее регулятор будет отрабатывать возмущения.

5. Комплементарный фильтр

Как говорилось выше, чтобы получить верные данные положения тела, одного гироскопа или акселерометра недостаточно, нужно объединять их. Сделать это можно с помощью комплементарного или альфа-бета фильтра.

Итак, имеются два прибора, каждый из которых позволяет рассчитать угол наклона машины относительно поверхности земли. Но в случае гироскопа точность таких расчетов снижается из-за дрейфа нуля и ошибок интегрирования. В случае же акселерометра слишком велика чувствительность к внешним воздействиям.

Возникает естественное желание объединить показания этих двух устройств, для устранения их недостатков. Сделать такое объединение позволяет комплементарный фильтр, который немного меняет формулу для интегрирования показаний гироскопа:

$$a(t) = (1-K) * (a(t-1) + gx*dt) + K * acc \quad [2]$$

Здесь

- a(t) — искомый угол наклона, учитывающий показания акселерометра;
- a(t-1) — угол тела в предыдущий момент времени;
- gx — скорость вращения тела вокруг оси X;
- dt — время, которое прошло с момента предыдущего вычисления угла a;
- acc — значение угла наклона, полученное при помощи акселерометра;
- K — коэффициент комплементарного фильтра.

Как видно из формулы, итоговая величина угла наклона представляет собой сумму интегрированного значения гироскопа и мгновенного значения акселерометра. По сути, главная задача комплементарного фильтра здесь в том, чтобы с помощью показаний акселерометра нивелировать дрейф нуля гироскопа и ошибки дискретного интегрирования. Указанное выражение именно это и делает. На каждом шаге интегрирования (по сути, на шаге цикла управления машиной) мы корректируем интеграл угла наклона с помощью показаний акселерометра. Сила этой коррекции определяется коэффициентом фильтра K.

Выбор коэффициента K зависит от величины дрейфа нуля гироскопа, от скорости накопления ошибок вычисления и от условий использования машины. Так, слишком большое значение K приведет к тому, что на результат работы фильтра будет сильно влиять вибрация корпуса. Слишком малое значение K может оказаться недостаточным, чтобы ликвидировать дрейф нуля гироскопа. Как правило, коэффициент комплементарного фильтра подбирается вручную для каждого инклинометра исходя из вышеуказанных условий. Например, для любительских мультикоптеров K может принимать значение в диапазоне от 0,05 до 0,01.

6. Прерывания, регистры и таймеры

После сборки можно приступать к программированию. Алгоритм вроде бы понятен, но и тут есть свои подводные камни, даже целые булыжники. Процессор микроконтроллера в обычном режиме может выполнять только одну последовательность действий. Т.е. если цикл программы занимает, например, 100 мс, то он будет выполнять только его, а для корректной балансировки необходимо опрашивать датчик, например, каждые 5 мс. Вот тут и возникает проблема. Но, к счастью, разработчики микроконтроллеров предусмотрели такие случаи и сделали возможности выполнения

(псевдо-)параллельных процессов. Они выполняются с помощью прерываний. Прерывания бывают двух типов: внешние – аппаратные и внутренние – по таймеру.

Прерывание представляет собой событие, при наступлении которого выполнение основной программы приостанавливается и вызывается функция, обрабатывающая прерывание определённого типа.

Прерывания делятся на внутренние и внешние. К источникам внутренних прерываний относятся встроенные модули микроконтроллера (таймеры, приёмопередатчик USART и т.д). Внешние прерывания возникают при поступлении внешних сигналов на выводы микроконтроллера (например, сигналы на выводы RESET и INT). Характер сигналов, приводящих к возникновению прерывания, задаётся в регистре управления MCUCR, в частности в разрядах - ISC00 (бит 0) и ISC01 (бит 1) для входа INT 0; ISC10 (бит 2) и ISC11 (бит 3) для входа INT1.

В микроконтроллере ATmega8 каждому прерыванию соответствует свой вектор прерывания (адрес в начале области памяти программ, в которой хранится команда для перехода к заданной подпрограмме обработки прерывания).

Table 18. Reset and Interrupt Vectors

Vector No.	Program Address(2)	Source	Interrupt Definition
1	0x000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, and Watchdog Reset
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	TIMER2 COMP	Timer/Counter2 Compare Match
5	0x004	TIMER2 OVF	Timer/Counter2 Overflow
6	0x005	TIMER1 CAPT	Timer/Counter1 Capture Event
7	0x006	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	0x007	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	0x008	TIMER1 OVF	Timer/Counter1 Overflow
10	0x009	TIMER0 OVF	Timer/Counter0 Overflow
11	0x00A	SPI, STC	Serial Transfer Complete
12	0x00B	USART, RXC	USART, Rx Complete
13	0x00C	USART, UDRE	USART Data Register Empty
14	0x00D	USART, TXC	USART, Tx Complete
15	0x00E	ADC	ADC Conversion Complete
16	0x00F	EE_RDY	EEPROM Ready
17	0x010	ANA_COMP	Analog Comparator
18	0x011	TWI	Two-wire Serial Interface
19	0x012	SPM_RDY	Store Program Memory Ready

Notes: 1. When the BOOTRST Fuse is programmed, the device will jump to the Boot Loader address at reset, see "Boot Loader Support – Read-While-Write Self-Programming" on page 202
2. When the IVSEL bit in GICR is set, Interrupt Vectors will be moved to the start of the boot Flash section. The address of each Interrupt Vector will then be the address in this table added to the start address of the boot Flash section

Table 19 on page 47 shows reset and Interrupt Vectors placement for the various combinations of BOOTRST and IVSEL settings. If the program never enables an Interrupt source, the Interrupt Vectors are not used, and regular program code can be placed at these locations. This is also the case if the Reset Vector is in the Application section while the Interrupt Vectors are in the boot section or vice versa.

В mega8 все прерывания имеют одинаковый приоритет. В случае одновременного возникновения нескольких прерываний первым будет обрабатываться прерывание с меньшим номером вектора.

За управление прерываниями в ATmega8 отвечают 4 регистра:

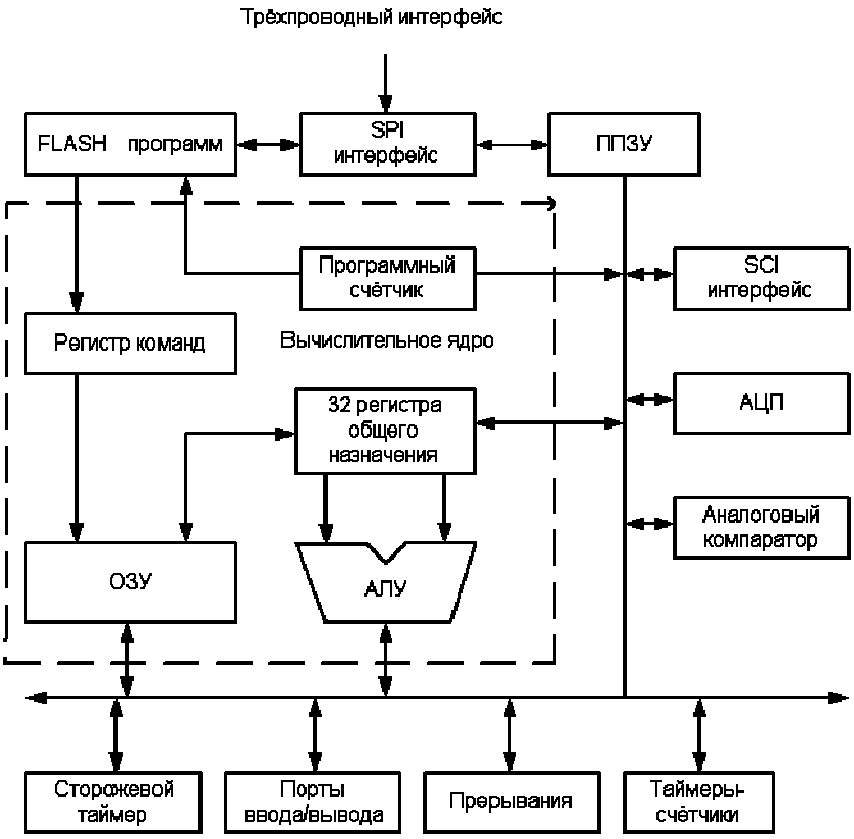
- GIMSK (он же GICR) - запрет/разрешение прерываний по сигналам на входах INT0, INT1
- GIFR - управление всеми внешними прерываниями
- TIMSK, TIFR - управление прерываниями от таймеров/счётчиков

Начинающие радиолюбители и программисты микроконтроллеров не всегда знают, что такое регистры и таймеры-счетчики, а в работе с прерываниями эти знания необходимы. Для этого нужно разобраться во внутренней структуре микроконтроллера.

как правило, любой микроконтроллер содержит следующие основные узлы:

- Арифметико-логическое устройство (АЛУ или ALU);
- Оперативная память (ОЗУ);
- Постоянная память (ПЗУ);
- Генератор тактовой частоты;
- Порты ввода/вывода;
- Таймеры;

Вот так выглядит упрощенная структурная схема микроконтроллера:



Сердцем микроконтроллера является арифметико-логическое устройство (АЛУ). АЛУ производит все арифметические и логические операции с двоичными данными. Бывают АЛУ различной разрядности: 8-, 16- или 32-разрядные. Например, если АЛУ 8-

разрядное, то оно может провести операцию над двумя восьмиразрядными числами и выдать восьмиразрядный результат операции. АЛУ производит операции над числами и возвращает результат операции в виде числа. Данные числа помещаются в регистры общего назначения – своеобразную временную память. У каждого микроконтроллера количество регистров может быть разным. На структурной схеме приведен пример контроллера, у которого 32 регистра общего назначения.

Таймеры-счётчики — это такие устройства или модули в микроконтроллере, которые, как видно из названия, постоянно что-то считают. Считают они либо до определённой величины, либо до такой величины, сколько они битности. Считают они постоянно с одной скоростью, со скоростью тактовой частоты микроконтроллера, поправленной на делители частоты, которые мы будем конфигурировать в определённых регистрах.

И вот эти таймеры-счётчики постоянно считают, если мы их инициализируем.

Таймеров в МК Atmega8 три: два 8-битных и один 16-битный.

7. Программирование

Теперь рассмотрим структуру и основные части программы для балансирующего робота.

В программе используются несколько библиотек:

- "Wire.h" , "I2Cdev.h" – библиотеки протокола I2C
- "MPU6050.h" – библиотека для работы с датчиком
- "math.h" – библиотека для математических вычислений

Библиотека ардуино – это некий программный код, хранящийся не в скетче, а во внешних файлах, которые можно подключить к вашему проекту. В библиотеке хранятся различные методы и структуры данных, которые нужны для упрощения работы с датчиками, индикаторами, модулями и другими компонентами. Использование готовых программ существенно упрощает работу над проектами, потому что можно сосредоточиться на основной логике, не тратя время на множество мелочей.

Вот и то, для чего рассматривали прерывания, регистры и таймеры – объявление и настройка прерывания в программе

```
void init_PID() {  //объявление прерывания
  cli();          // отключение глобальных прерываний
  TCCR1A = 0;     // установка регистра TCCR1A на 0
  TCCR1B = 0;     // то же для TCCR1B
  // установка таймера на 5 мс
  OCR1A = 9999;
  // включить режим CTC
  TCCR1B |= (1 << WGM12);
  // Установка бита CS11 для предписания 8
  TCCR1B |= (1 << CS11);
  // включение сравнения прерывания таймера
  TIMSK1 |= (1 << OCIE1A);
  sei();          // включение глобальных прерываний
}
```

Далее идет подключение и установка данных датчика

```
mpu.initialize();  //включение датчика
    mpu.setXGyroOffset(200); //установка начальных данных гироскопа и акселерометра
    mpu.setYGyroOffset(100);
    mpu.setZAccelOffset(1788);
```

Теперь, займемся вычислением угла наклона в данный момент времени

- Получаем данные с акселерометра и переводим их

```
accAngle = atan2(accY, accZ)*RAD_TO_DEG;
```

- Получаем данные с гироскопа и переводим их в угол

```
gyroRate = map(gyroX, -32768, 32767, -250, 250);
gyroAngle = (float)gyroRate*sampleTime;
```

- Теперь берем оба значения и с помощью комплементарного фильтра вычисляем истинный угол наклона

```
currentAngle = 0.9934*(prevAngle + gyroAngle) + 0.0066*(accAngle);
```

После того, как угол найден, можно приступить к расчету мощности двигателя с помощью ПИД-алгоритма

- В самом начале программы задаем коэффициенты ПИД-регулятора

```
#define Kp  60
#define Kd  0.15
#define Ki  80
```

- Вычисляем ошибку, вычитая из текущего угла желаемый

```
error = currentAngle - targetAngle;
```

- Находим накопленную ошибку (интеграл)

```
errorSum = errorSum + error;
errorSum = constrain(errorSum, -300, 300);
```

- Наконец, имея все данные, вычисляем мощность двигателя

```
motorPower = Kp*(error) + Ki*(errorSum)*sampleTime - Kd*(currentAngle-prevAngle)/sampleTime;
```

8. Настройка ПИД-регулятора по методу Циглера-Николса

Циглер и Николс предложили свой вариант быстрой настройки ПИД-регулятора для периодического переходного процесса, в котором затухание примерно равно 4.

- Обнуляем K_i и K_d
- Постепенно увеличиваем Kp до критического значения K_c, при котором возникают автоколебания
- Измеряем период автоколебаний T
- Вычисляем значения K_p, K_i и K_d по разным формулам для разных регуляторов:
  - для П-регулятора: $K_p=0,50*K_c$
  - для ПИ-регулятора: $K_p=0,45*K_c$, $K_i=1,2*K_p/T$
  - для ПИД-регулятора: $K_p=0,60*K_c$, $K_i=2,0*K_p/T$, $K_d=K_p*T/8$

1. что такое балансирующий робот. краткая математическая модель.
2. алгоритм программы. Желательно в виде графической структуры.
3. описание библиотек. Что это такое. как их применять в программе.
   что конкретно выполняют библиотеки в данной программе.
4. Описание ПИД-регулятора. что это такое. как он применяется в данной программе.
5. акселерометр-гироскоп MPU6050. что собой представляет этот модуль.
6. Что такое прерывания. для чего они нужны. каое преимущество дает прерывание.
7. Регистры. Мало кто знает внутренюю структуру процессора. Надо бы как то популярно написать о регистрах, таймерах.
   почему в данном случае применяется непосредственная работа с регистрами, а не с портами, как привыкли практически все.

Источники:

- https://intellect.ml/balansiruyushhij-robot-81
- http://atmega8.ru/wiki/view/doc.9.html
- http://narodstream.ru/avr-urok-10-tajmery-schetchiki-preryvaniya/
- http://hamlab.net/mcu/training/introduction.html
- http://cxem.net/mc/mc324.php
- http://robotclass.ru/articles/complementary-filter/