# [С](https://circuitdigest.com/microcontroller-projects/arduino-based-self-balancing-robot)амобалансирующийся робот на базе Arduino

После того, как меня вдохновили моторы RYNO и другие самобалансирующиеся скутеры от Segway, я захотел создать нечто подобное. Подумав некоторое время, я решил создать своего самобалансирующегося робота на базе Arduino. Таким образом, я смогу понять основной принцип работы всех этих скутеров, и больше узнать про работу алгоритма PID.

Как только я приступил к сборке, стало понятно, что собрать этого робота не так-то просто. Передо мной открылось большое количество вариантов, из которых нужно было выбрать подходящие именно мне и не запутаться в процессе проектировки, выбора двигателей и настройки значений PID. И мне так же пришлось столкнуться с большим количеством вопросов: какой выбрать аккумулятор, как его расположить, какое сцепление колес мне нужно, какой тип двигателя выбрать, как правильно рассчитать центр тяжести всей конструкции и множеством других.

Но вам точно стоит знать, что, когда вы все это сделаете — вы согласитесь, что это не так сложно, как кажется. Итак, расскажу вам что будет дальше. В этой статье, я буду рассказывать вам о своем опыте постройки самобалансирующегося робота. Вы можете быть полным новичком в радиоэлектронике или наткнуться на эту статью, после бессонных ночей, проведенных в попытках понять, почему же ваш робот не работает. Эта статья будет конечной точной вашего назначения, после которой все заработает. Итак, приступаем к работе!

### Необходимые компоненты для сборки робота

Перед тем как я расскажу вам, как построить робота, прочитайте список того, что я использовал в этом проекте.

* МикроконтроллерArduino UNO x1
* Мотор постоянного тока с редуктором (Желтого цвета) x2
* Драйвер мотора L298N x1
* Гироскоп и акселерометр MPU6050 x1
* Пара колес
* Li-ion батарея на 7.4V x1
* Соединительные провода
* Рамка крепления, напечатанная на 3D принтере x1

**Микроконтроллер:** В этом проекте я использовал микроконтроллер Arduino UNO, потому что с ним просто работать. Вы можете строить на базе Arduino Nano или Arduino mini, но я рекомендую вам раскошелиться на UNO, потому что мы можем запрограммировать его напрямую, без использования какого-либо внешнего оборудования.

**Двигатели:** Лучшим вариантом для этого проекта будет использование 2-х шаговых двигателей. Но для упрощения проекта, я использовал моторчики постоянного тока с редуктором. Да да, вовсе не обязательно иметь шаговые двигатели — мой робот отлично работает на этих дешевых и общедоступных моторчиках с редуктором.

**Драйвер моторов:** Тут все просто: если вы, как и я собираете робота на моторах постоянного тока с редуктором — вам понадобится плата, управляющая напряжением и направлением вращения этих моторов. У меня в запасах был драйвер L298N, но вы так же можете использовать L293D и все будет прекрасно работать. [Здесь](https://circuitdigest.com/microcontroller-projects/dc-motor-control-with-arduino-uno-pwm) вы можете подробнее почитать, про управление мотором постоянного тока при помощи драйвера L293D и Arduino.

**Колеса:** это очень важный параметр. При постройке, мне понадобилась куча времени, чтобы понять, что робот работает не так как, нужно из-за неподходящих колес. Так что обязательно проверьте, чтобы робот имел хорошее сцепление с покрытием, на котором вы будете его использовать. Внимательно следите, чтобы ваш робот не скользил по покрытию, в противном случае вам нужно очистить поверхность или заменить резину на более мягкую.

**Датчик ускорения и гироскоп:** Лучшим вариантом, совмещающим акселерометр и гироскоп в одной плате, является MPU6050. Поэтому не пытайтесь построить такого робота, используя обычный акселерометр по типу ADXL345 – работать не будет. В конце статьи, вы поймете почему. Для тех кому интересна данная плата, можете почитать отдельную статью, посвященную работе с [MPU6050](https://circuitdigest.com/microcontroller-projects/mpu6050-gyro-sensor-interfacing-with-arduino).

**Батарея:** Нам нужна как можно более легкая батарея с напряжением выше 5V, для питания Arduino без повышающего модуля. Поэтому, идеальным выбором является Литий-полимерная (Li-po) батарея на 7.4V. К сожалению, под рукой у меня оказалась только Литий ионная батарея (Li-ion) на 7.4V, которую я и использовал. Но запомните, что Li-po батарея лучше Li-ion.

**Шасси:** Еще одно место, которое не терпит компромиссов — это шасси вашего робота. Вы можете сделать его сами из всего что попадется под руку — дерево, пластик, картон или что-либо еще с чем вы умеете работать. Но вы должны быть уверенны, что шасси получится крепким и не будет шататься, когда робот будет балансировать. Я спроектировал свою собственную модель шасси в бесплатной программе Solidworks, и напечатал его на 3D принтере. Если у вас тоже есть 3D принтер - вы можете распечатать такое же шасси, файлы для печати которого будут прикреплены в следующей главе.

### 3D печать и сборка нашего робота

Если вы решите напечатать такую же шасси, как у меня — то просто скачайте STL файл на [thingverse](https://www.thingiverse.com/thing%3A3052026). Так же я добавил, так называемые дизайн-файлы, если вам захочется отредактировать рамку под ваши предпочтения.

Все детали не имеют выступающих конструкций, и вы легко может напечатать их без использования поддержек и при заполнении в 25% это будет отлично работать. Дизайн самих фигур очень простой и любой принтер справится с их печатью. Для нарезки модели я использую программу Cura software и печатаю все это на принтере Tevo Tarantula, настройки которого вы видите ниже.

Вам нужно будет напечатать часть корпуса и 4 детали для крепления двигателей. Затем, просто собираем всю конструкцию, используя болты и гайки на 3мм и расставляя моторы по местам. В итоге у вас должна получится конструкция, похожая на ту, что изображена ниже.

Изначально я разрабатывал модель под драйвер [L298N](https://circuitdigest.com/microcontroller-projects/arduino-motor-driver-shield-pcb), который расположен на нижней полке нашей рамы, Arduino разместилась на средней, а батарея наверху, как показано на фотографии ниже. Если вы собираете робота, в такой же последовательности, как и я, то советую вам прикрутить Arduino через крепежные отверстия болтами на 3 мм, а также закрепить провода, идущие от батареи нейлоновыми стяжками. Эта компоновка, будет отлично работать при использовании обычных колес, которые я поменял в дальнейшем.

В моем роботе, я поменял расположение батареи и Arduino, для облегчения программирования и подключения проводов. В итоге внешний вид робота отличается от изначальной планировки. После всех подключений, заливки скетча и тестов - мой робот стал выглядеть вот так.

### Принципиальная схема подключения

Процесс подключения всех компонентов нашего робота очень прост. Мы имеем интерфейс управления, плату MPU6050 подключенную к Arduino, а также моторчики, подключенные через драйвер, и питает всю эту радость наша Li-ion батарея на 7.4V. Принципиальная схема подключения показана ниже.

Саму Arduino и драйвер L298N мы питаем, через пин VIN и клемму на 12V соответственно. Регулятор, расположенный на плате Arduino, конвертирует получаемые на входе 7.4V в 5V и процессор ATmega IC вместе с MPU6050, будут запитаны правильным напряжением. Двигатели постоянного тока могут работать в диапазоне от 5V до 12V. Но мы будем подключать их от плюсового провода батареи на 7.4V к входному терминалу на 12V, расположенному на драйвере. Это заставит наши моторы работать от напряжения 7.4V. Ниже вы видите схему подключения датчика MPU6050 и драйвера L298N, к нашей Arduino.

Датчик MPU6050 общается с Arduino через интерфейс I2C, поэтому нам нужно задействовать SPI пина A4 и A5 на Arduino. Двигатели постоянного тока подключены к PWN пинам - D6,D9 D10 и D11. Нам важно подключить их к PWM пинам, потому что мы будем управлять скоростью моторов, через изменение рабочего цикла PWM сигналов. Если вы не совсем понимаете принцип работы этих двух компонентов, советуем вам прочитать статью про взаимодействие с интерфейсом [MPU6050](https://circuitdigest.com/microcontroller-projects/mpu6050-gyro-sensor-interfacing-with-arduino) и мануал по драйверу L298N.

### Программируем самобалансирующегося робота

Следующим шагом в этом проекте, будет программирование Arduino UNO, для того чтобы научить робота держать баланс. Здесь и происходит вся магия: основная идея очень проста. Нам нужно знать, направление наклона робота, вперед или назад – об этом нам скажет датчик **MPU6050**, затем если он наклонен вперед, то нам так же нужно прокрутить колеса вперед, и если он наклоняется назад, то прокрутить колеса в противоположном направлении.

В то же время нам нужно управлять скоростью с которой крутятся колеса: если робот слегка отклонился от центра, то колеса крутятся медленно и чем дальше от центра наклонен робот, тем быстрее крутятся колеса. Мы используем алгоритм для PID (Пропорционально-интегрально-дифференцирующего регулятора) и чтобы этот алгоритм правильно работал, мы выставляем центральное положение робота, как заданное значение (setpoint), а уровень наклона используем в качестве выходного значения (output).

Для того, чтобы узнать текущее положение робота, мы используем **MPU6050**, который представляет из себя 6-ти осевой акселерометр с гироскопом в одной плате. Чтобы получить достоверные данные от датчика, нам нужно получить данные как от акселерометра, так и от гироскопа, потому что данные, полученные с акселерометра, имеют помехи, а данные, полученные от гироскопа, имеют тенденцию отклоняться со временем. Поэтому мы объединяем эти два показателя и получаем следующие значения – положение робота по продольной, поперечной и вертикальной осям, хотя мы будем использовать только значение вертикальной оси.

Это звучит немного головокружительно, правда? Но не беспокойтесь! Скажем спасибо огромному сообществу фанатов Arduino, которые уже написали библиотеки, которые могут рассчитывать значения для PID-алгоритма и так же получать данные положения по вертикальной оси с датчика MPU6050. Скажем спасибо этим двум парням: [br3ttb](https://github.com/br3ttb) and [jrowberg](https://github.com/jrowberg) за создание таких прекрасных библиотек. Прежде чем мы продолжим, скачайте их библиотеки и закиньте их в папку /lib вашей Arduino IDE.



Теперь, после того как мы добавили библиотеки в Arduino IDE, давайте уже начнем программировать нашего самобалансирующегося робота. Как и всегда, полный скетч данного проекта расположен внизу данной статьи, а здесь я лишь объясняю некоторые важные куски кода. Как я говорил ранее, мой скетч основан, на [примере кода для MPU6050](https://github.com/jrowberg/i2cdevlib/blob/master/Arduino/MPU6050/examples/MPU6050_DMP6/MPU6050_DMP6.ino), который и оптимизировал под свою задачу и добавил метод для работы PID-алгоритма с методом управления для нашего самобалансирующегося робота.

Первым делом подключаем библиотеки, необходимые для работы программы. Они включают в себя встроенную библиотеку для последовательного протокола обмена данными (I2C), и библиотек для вычисления значений PID-алгоритма и получения данных с MPU6050, о которых мы говорили ранее.



Затем мы объявляем переменные, которые нужны нам для получения данных с датчика MPU6050. Далее мы считываем направление гравитации и значения кватернионов, а затем вычисляем положение робота по вертикальной, продольной и поперечной осям. В итоге, результат вычислений хранится в массиве с плавающей точкой, под названием *ypr[3]*.



Следующая часть содержит важнейший кусок кода и это тот момент, где вы потратите больше всего времени, так как – вам нужно подобрать правильные значения, для того чтобы робот стоял строго по центру. Если вы очень хорошо собрали своего робота и симметрично расположили все компоненты для достижения идеального центра тяжести (что в большинстве случаев не так), то заданное значение (setpoint) для вас будет равно 180. Если нет, то подключите вашего робота к компьютеру, откройте монитор порта и наклоните робота так, чтобы найти баланс, затем считайте значение, показанное в мониторе порта - оно и будет вашим заданным значением. Значения Kp, Kd и Ki так же настраиваются индивидуально для вашего робота. Тут важно понять, что не существует двух одинаковых роботов с идентичными значениями Kp, Kd и Ki, так как имеют место быть погрешности в сборке, поэтому в любом случае вам придется подгонять эти параметры под себя. Посмотрите видео в конце этой статьи, чтобы лучше понять, как настроить эти значения.



В следующей строчке кода мы подключаем алгоритм для работы с PID, путем передачи вводных переменных - input, output, set point, Kp, Ki и Kd. Из них мы уже установили значения для переменных Kp, Ki и Kd в приведенном выше фрагменте кода. Значением переменной input, будет являться текущее значение положения вертикальной оси, полученное с помощью датчика MPU6050, а значением переменной output, будет результат вычислений PID алгоритма. Таким образом, алгоритм дает нам значение переменной output, которое мы используем для корректировки значения Input, чтобы оно было близко к заданному значению (setpoint).



Функция *void setup*, отвечает за инициализацию датчика MPU6050 путем настройки цифрового обработчика движений или DMP (Digital Motion Processor). Это поможет нам объединить данные акселерометра с данными гироскопа и обеспечить правильные показания расположения робота по вертикальной, продольной и поперечной осям. Мы не будем сильно углубляться в теорию этого процесса, так как эти знания далеки от нашей темы. Но есть один кусок кода, на который вы должны обратить свое внимание, и это значения смещений для вашего акселерометра, которые находятся в функции setup (GyroOffset\AccelOffset). Каждый экземпляр датчика MPU6050 имеет свои собственные значения для смещений, и вы можете найти их, используя [этот скетч для Arduino](https://forum.arduino.cc/index.php?action=dlattach;topic=446713.0;attach=193816), а затем просто подставьте их в следующие строки кода.



Мы так же должны инициализировать цифровые пины PWM, которые мы используем для подключения двигателей. В нашем случае это пины D6, D9, D10 и D11. Итак мы определяем данные пины, как пины вывода и задаем им изначальное значение LOW.



Внутри главного цикла функции, мы проверяем, доступны ли данные с MPU6050 для чтения. Если доступны, то мы используем их для вычисления значений PID алгоритма, а затем выводим значения input и output в монитор порта. Мы делаем это чтобы перестраховаться и точно знать, что наш алгоритм правильно работает. Затем на основе значения переменной output мы решаем, куда должен двигаться робот – вперед, назад или стоять на месте.
Мы предполагаем, что MPU6050 вернет нам значение 180, когда робот будет находится в вертикальном положении. Мы получим положительное значение для корректировки, если робот наклоняется вперед, и отрицательные значения, когда он наклоняется назад. Поэтому мы обязательно проверяем это условие и вызываем соответствующие функции, необходимые для перемещения робота вперед или назад, в зависимости от полученных данных.



Так же стоит помнить, что от значения переменной output нашего PID-регулятора зависит скорость вращения моторчиков.Если робот слегка наклоняется, то мы делаем небольшую корректировку и колеса крутятся медленно. Если небольшая корректировка не помогла, и робот продолжает наклоняться, мы просто увеличиваем скорость вращения моторчиков. Значение того, насколько быстро будут вращаться моторчики, будет определяться нашим алгоритмом. Для вращения моторчиков в противоположную сторону, мы умножаем значение output на -1 и таким образом, превращаем отрицательное значение в положительное.



### Завершающий этап постройки самобалансирующегося робота на базе Arduino

Когда вы закончили этап сборки, вам следует загрузить скетч в Arduino. Тщательно проверьте правильность подключения всех компонентов, и помните, что мы используем Li-ion батарею, которая может быть опасна при неправильном обращении с ней. Так что дважды проверьте все контакты и разъемы, чтобы ни один из проводов не отвалился и не вызвал короткое замыкание, так как наш робот может получать небольшие удары во время работы. Подключите питание и откройте монитор порта: если ваша Arduino успешно взаимодействует с датчиком MPU6050 и все работает так как вы и ожидали, то вы увидите следующую картину.

Здесь мы видим значения input и output для PID алгоритма, в формате input => output**.** Если робот идеально сбалансирован, то значение output будет равно 0.Значение input, показывает текущее значение, полученное с датчика MPU6050. Буква “F” обозначает движение робота вперед, а буква “R” соответственно назад.

После настройки алгоритма, я рекомендую вам пока что не отключать usb-кабель от Arduino. В первое время, очень важно понаблюдать за изменениями значений input и output в мониторе порта и так же вам возможно понадобиться подкорректировать значения Kp, Ki и Kd и заново загрузить скетч. На видео ниже, вы можете посмотреть на итоговую работу робота, а также в нем показано как изменять значения для PID алгоритма.

Надеюсь, эта статья помогла вам в постройке вашего собственного самобалансирующегося робота, и не смотря на обилие сложных терминов и непонятных с первого раза вещей, вы все-таки довели дело до конца, и ваш робот прекрасно работает.