**Link: https://wired.chillibasket.com/2015/01/calibrating-mpu6050/**

**Designing the Robot**

***Weight Distribution:*** Self-balancing robots work on the principle of an inverted pendulum. This means that the system is most stable when all of the mass is positioned as high as possible. This seems to go against common sense; usually systems are more stable when they have a low centre of gravity. In this case keeping the mass on top increases the inertia of the system, meaning that the robot has more time to respond to changes in balance. Therefore my first recommendation is to place the heaviest objects, such as the battery, at the top of the robot.

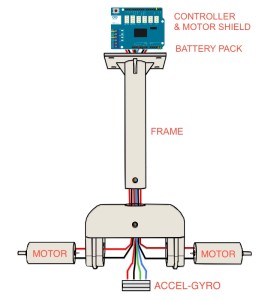
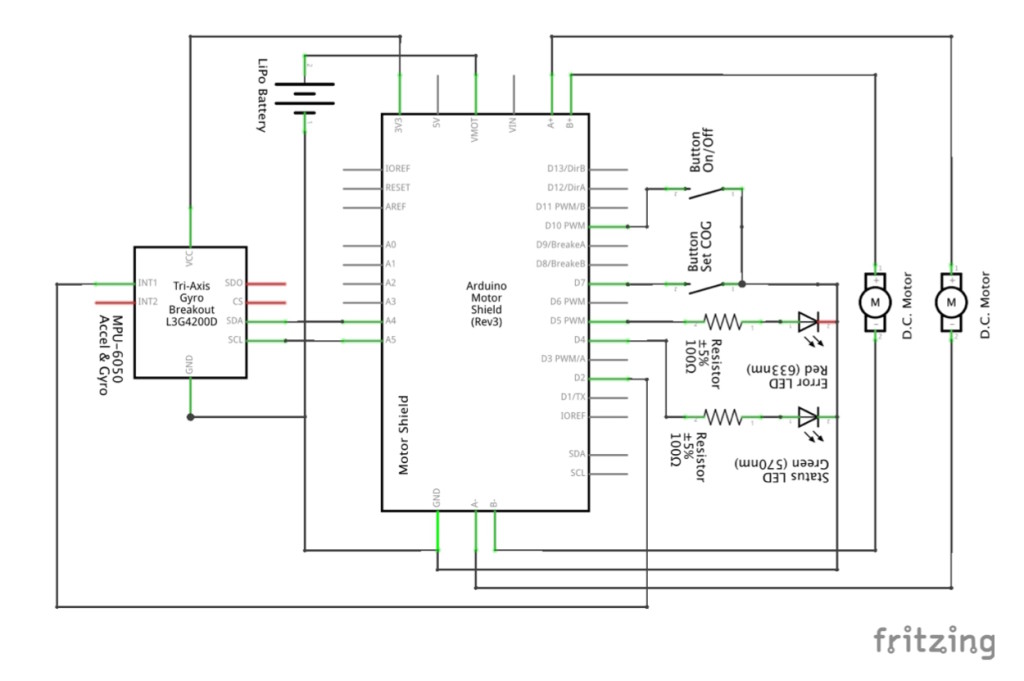
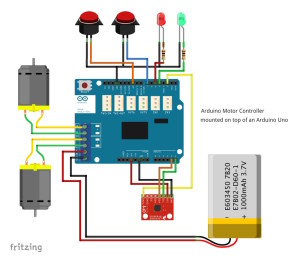
***Sensor Positioning:***The positioning of the accelerometer/gyroscope module is also important. When I was demonstrating my balancing robot at the Dublin Maker Faire this year, I asked a number of people where they think the sensor should be positioned. Most guessed that it should be on top, as this is where it would record the largest amount of movement!

We actually want to avoid as much of this translational movement as possible, as we are only interested in the rotation of the robot. Therefore the sensor should be placed exactly on the axis of rotation, between both wheels. Placing the sensor further up on the frame introduces noise and jitter into the readings, and may cause a feedback loop (similar to the squeaking noise made when a microphone is too close to its own speaker).

***Frame Design:***For the rest of the frame, it is up to your own imagination what you want to do with it. I’ve included a couple of pictures and sketches below to help you come up with your own designs. Although I 3D printed two of my frames, I made my first prototype out of lollipop sticks (and it worked really well)!

**Assembling the Frame**

Putting together the frame and electronics is actually the easiest part of the project! Once you have your frame designed and ready to go, all you have to do is stick/screw all of the components together. Here is a schematic I made to help you with the wiring of the robot:

[](https://wired.chillibasket.com/wp-content/uploads/2015/09/overal-assembly-diagram.jpg)[](https://wired.chillibasket.com/wp-content/uploads/2015/10/balancing-schematic-1.jpg)[](https://wired.chillibasket.com/wp-content/uploads/2015/10/balancing-schematic-2.jpg)

**Combined Program**

In the previous parts of the tutorial I included snippets of code to show you how each part of the self-balancing robot should work. Here I have compiled all of the parts together into one code that you can use and modify for your own robot. I included a horrendous amount of comments, so that the program is as easy to follow as possible!

*Note: This code is programmed for the specific components I was using, such as an Arduino motor shield, and the MPU6050 Accel-Gyro module.*

Code in the same folder

Here is a breakdown of how to use this code with your robot:

1. Before starting the program, connect the board via USB to your computer, and open a terminal window in the Arduino software (baud rate: 115200).
2. **Wait until robot is ready:** At the start the robot automatically initialises the MPU6050 module. Once this is done, the following message should appear:  
   DMP Ready! Let's Proceed.
3. **Set Centre of Gravity:** You should set the centre of gravity of the robot, so that the robot knows which way is up! Do this by steadily holding the robot upright, with the wheels off the floor, and pressing the button connected to GPIO-4. The LED on GPIO-2 will flash, and the following message should appear:  
   > Setting new centre of gravity <
4. **Automatic On/Off:** The motors of the robot automatically turn off if the robot has fallen over, or is lying on its side. To turn them back on, hold the robot steadily in an upright position. The motors should start and the following message appear:  
   >>>> Balancing System Active <<<<  
   If the robot has fallen over and motors are off, this message appears:  
   >>>> Balancing System Stopped <<<<
5. **Manual On/Off:** To manually turn the balancing system on/off, press the button connected to GPIO-7. The LED on GPIO-10 will be bright, if the balancing system is turned on. One of the following messages will appear to let you know which state the robot is in:  
   > Turning off balancing system <  
   > Turning on balancing system <
6. **Sending new PID values:** Please read my guide [The PID Controller](https://wired.chillibasket.com/2015/03/pid-controller/), to see how to calibrate your robot. You can send new PID values via the console window, by typing the letter of the constant (P, I or D) you want to set, followed by the number you want to set it at. Then press the enter/return key to send. The code accepts any numbers between 0.01 – 99.99. For example:  
   P8.2 I1.51 D15  
   This sets the [P]roportional Constant to 8.2, the [I]ntegral Constant to 1.51, and the [D]erivative Constant to 15.

**Dealing with Common Errors**

I have found that most of the common errors can be dealt with by checking the following:

1. **Check the Pin Mapping:** Make sure that the GPIO number on (lines 54-65) match up with the ones you are using on your robot.
2. **Update the MPU-6050 Offsets:** Each sensor has unique offset values, which have to be inputted on (lines 183-188). I explain how to find these offsets in my “[Calibrating & Optimising the MPU6050](https://wired.chillibasket.com/2015/01/calibrating-mpu6050/)” part of this tutorial.
3. **Ensure sensor is working properly:** Check that the MPU-6050 is working, by uncommenting (line 704) of my code. While running, the robot should display the current angle on the console. When held upright, the angle should be 0. When pitching forwards/backwards, the number should be positive/negative in degrees.
4. **Both motors should spin in same direction:** Set the turning constants on (lines 102-103) to zero. Now both motors should spin in same direction. If not, then one of the motors is wired backwards.
5. **Motors balance in wrong direction:** Instead of stopping the robot from falling, the motors speed up the fall. This means that both motors are wired in backwards!

This finally concludes my tutorial about self-balancing robots! If you have any questions or suggestions, please leave a comment below.

*Updated: 23rd May 2019 – Reformatted post*