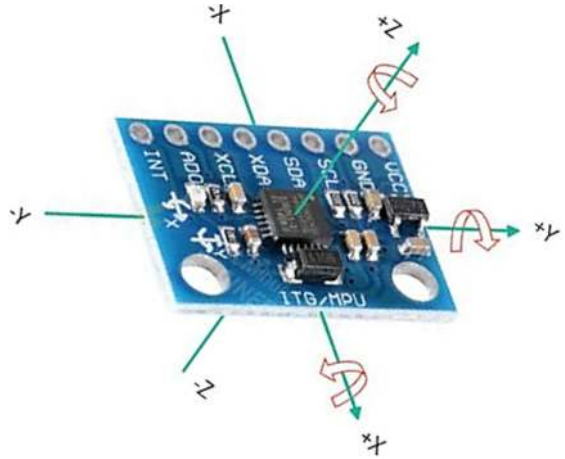
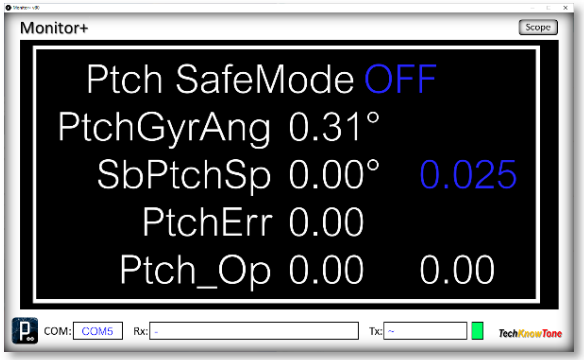
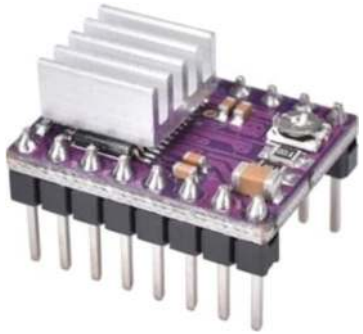
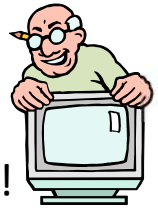
MandalBot BLE

Calibration



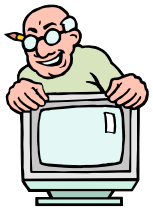






Calibration is an essential process!

Voltage Regulator



Set up the voltage regulator independently using an external power supply and multimeter, before you plug in the micro and drivers. The input voltage source should be between 8 - 12 volts. Trim the regulators output pot to achieve 3.3 volts. This is normally done by turning the small potentiometer on the pcb in a clock-wise direction, to reduce the output voltage. It will normally be too high to start with, and could damage your electronics if not done first.

Once the robot is wired and assembled, check that the output of the voltage regulator is still at 3.3 volts, before plugging in the ESP32.

Now insert the ESP32 micro, with the power off, and download the code into it. When power is re-applied, this should confirm that the RGB LED's are all working.

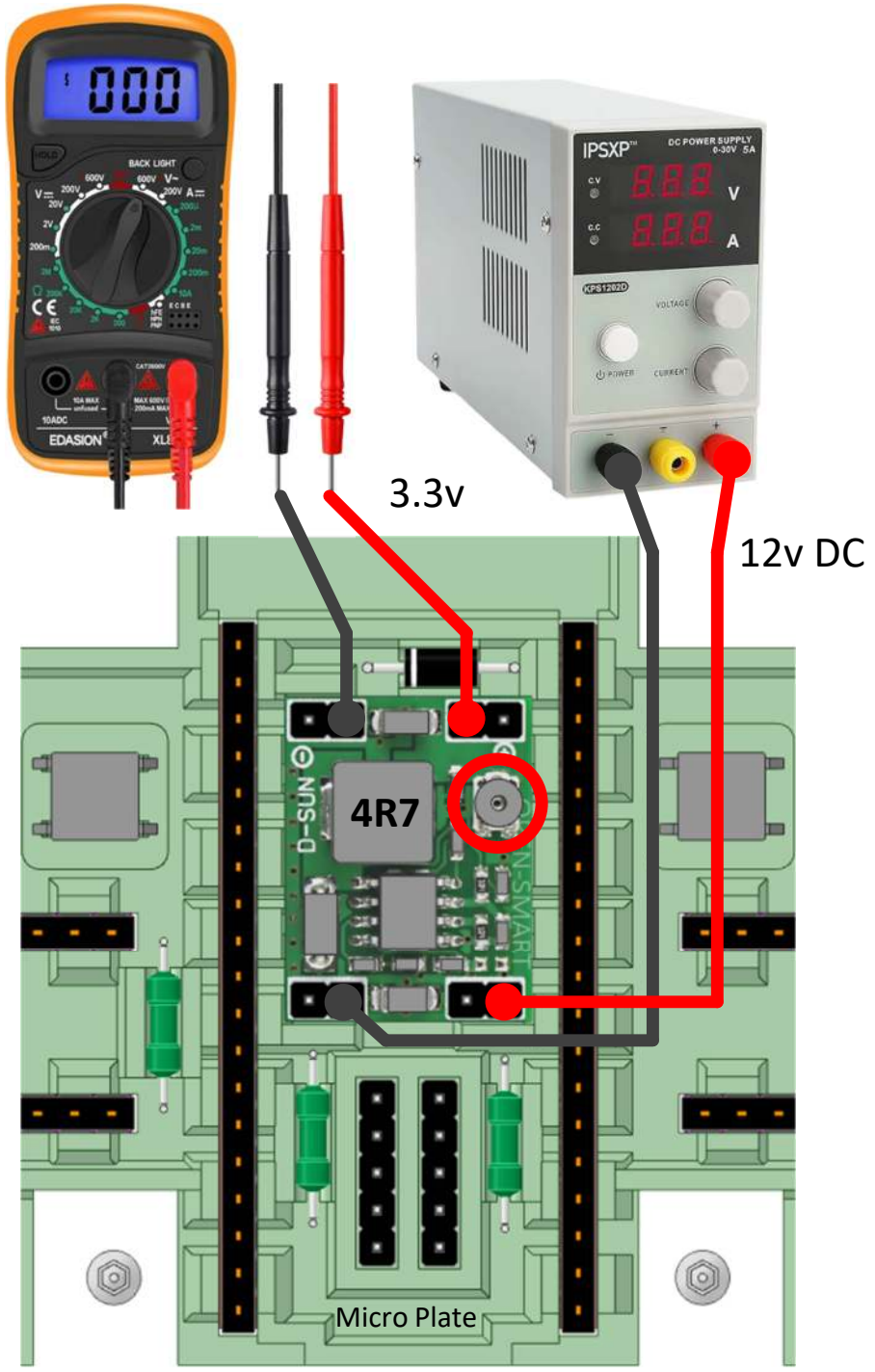
The accuracy of the analogue to digital converter (ADC) in the ESP32 can be improved by calibrating it's measurements against a multimeter, whilst using an adjustable voltage supply. This will give you a much more accurate indication of remaining battery capacity.

```
        #define BatCal    287.7
10.0v == #define BatCritical 2895
11.4v == #define BatHigh    3402
12.3v == #define BatMax     3841
        #define BatPin       36
10.8v == #define BatWarn    3175
```

The value of BatCal is determined, such that the value displayed in the Monito+ app matches that of the voltage measured at the battery terminals.

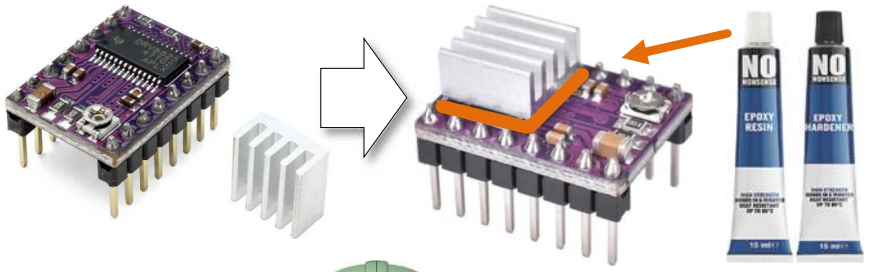
The BatCritical value is used to initiate an auto-shutdown of the robot, should the battery supply fall to or below this level.

If the ESP32 micro boots up on a low voltage reading, it will assume that it is being fed from a USB supply. This will limit code functionality.

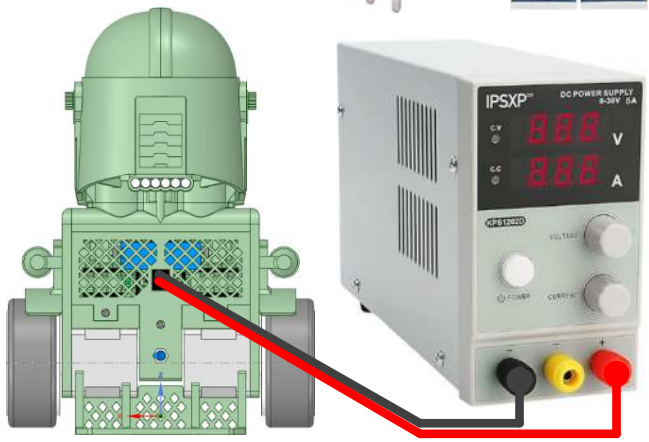


DRV8825 Stepper Drivers

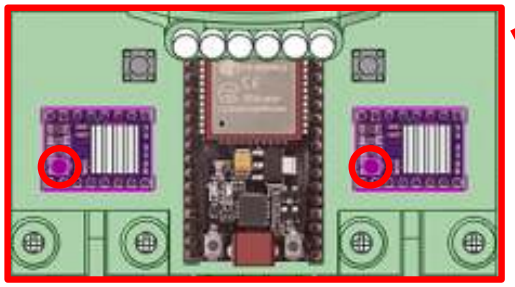
When you buy DRV8825 stepper motor drivers, they are often supplied with heatsinks, which need to be attached to the drivers using the conductive sticky tape on them. Do this carefully, to avoid the heatsink shorting out some of the pins on the pcb. Once happy with the location, I recommend that you secure them into position by surrounding their bases in 2-part epoxy glue; as they are quite exposed on the front of the MandalBot, and could easily be move with handling.



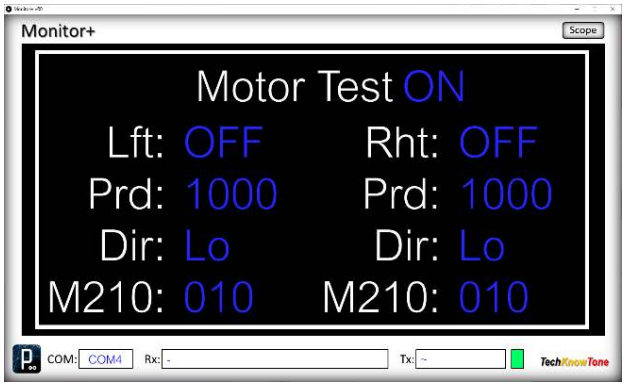
Next connect the robot, in the switched OFF condition, to an external DC supply, set to 12 volts. With the switch like this, the external supply will then be fed through to the internal circuitry, and you will be able to monitor the current being drawn by the robot.

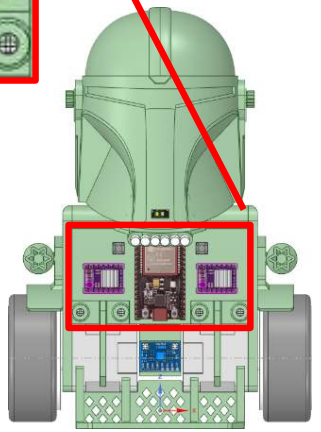


Using the Monitor+ app, navigate to the Motor Test screen, in TEST mode. Note that when the display has a border, this indicates that it is in TEST mode, as opposed to DEMO mode, with no border. On this screen you can modify the variables shown in blue text. For example, clicking on Motor Test **OFF** will enter a state in which each motor can be turned ON and run; and you can set the stepping period **Prd** and direction **Dir** too.



With both motors OFF the robot will be consuming a base level current, associated with running the micro and related electronics. Make a note of this base current. Then for each motor, turn it ON and adjust its associated DRV8825 driver potentiometer, to achieve a measured current of base + motor current. The aim is to get the motor current to be 150 - 200 mA, by adjusting the potentiometer. Turning the pot clockwise will reduce the current, and anti-clockwise will increase it. By default it can be quite high initially, and therefore a current limited supply is recommended.

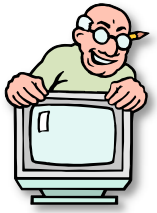




Once you have adjusted one driver, then turn OFF that motor and repeat the exercise for the other one. When you later turn both ON, the current drawn should be equivalent to the base current + both motors.

The aim is to give the stepper motors sufficient drive current, when static or stepping, without causing them to over-heat. Excess current will quickly reduce battery life, and run the motors very hot; and could even lead to them failing.

MPU6050A Orientation & Offsets



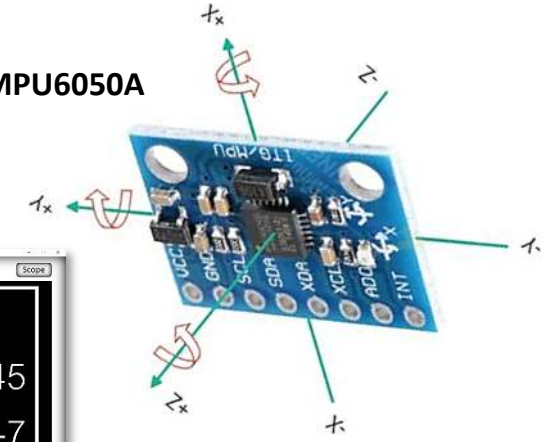
With the MPU6050 mounted centrally between the stepper motors as shown, the following applies:

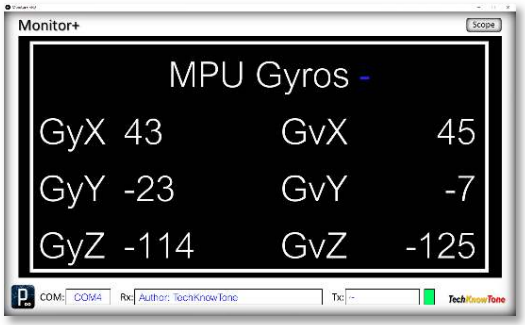
Pitch - Y gyro, -ve tilt forwards, +ve tilt backwards
- Z accelerometer, -ve lean forwards, +ve lean backwards

Roll - Z gyro, +ve tilt right, -ve tilt left
- Y accelerometer, -ve lean tilt, +ve tilt left

Yaw - X gyro, +ve turning left ,-ve turning right
- X accelerometer, , -ve upright, +ve upside down

MPU6050A





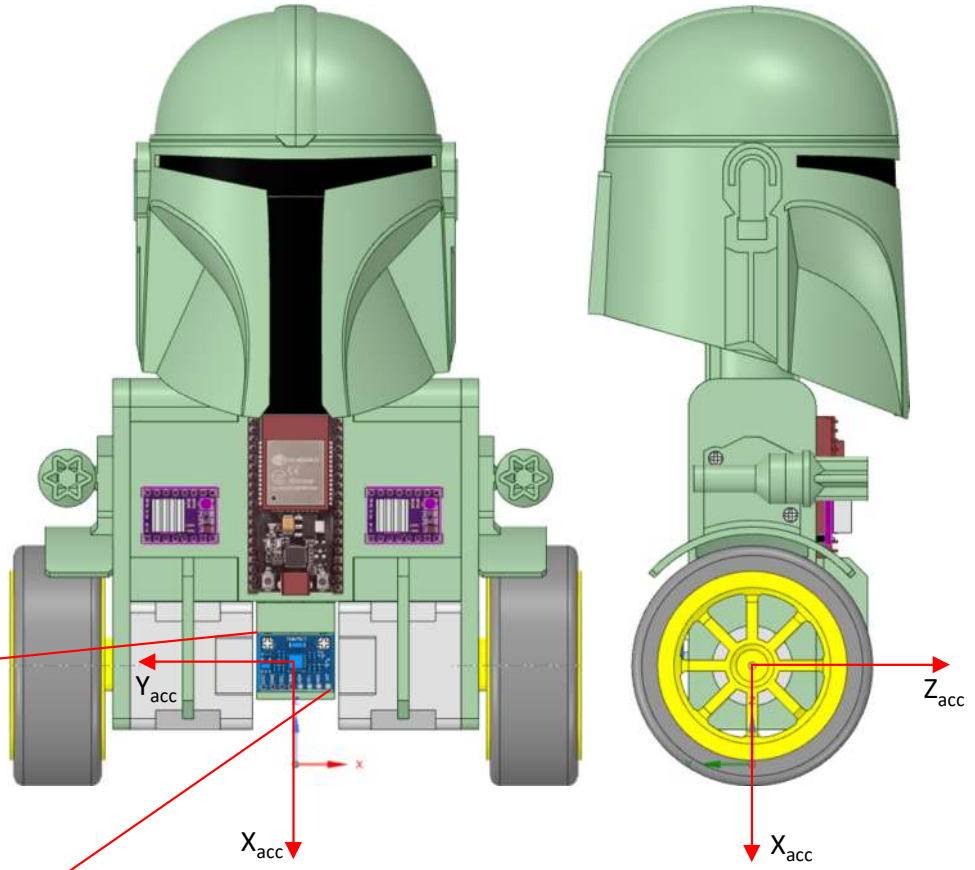
Note that the X-axis accelerometer is pointing downwards, when the robot is upright and balancing. Hence it is greatly influenced by gravity and gives a max negative reading, equivalent to 1 g_o (one unit of gravity).

The X-axis value is negative because the gravitational force is acting on the sensor as if it were accelerating upwards, and therefore in the opposite direction to the polarity indicated by the diagram. The MPU6050 thinks it is accelerating upwards.

The MPU6050 sensor values will contain offsets, due to manufacturing tolerances, and may not give maximum values. We therefore need to determine these offsets, and remove them from readings in our code; there by improving the overall accuracy of the system.

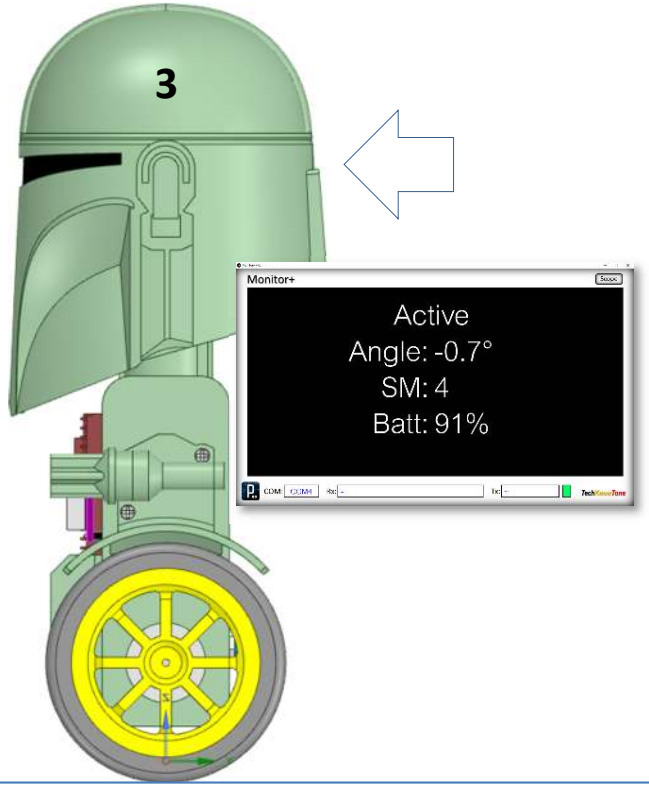
Set the code offsets to zero initially, as my values are not what you want. Then place the robot in its stand and use the Monitor+ app to view the gyro and accelerometer readings, and averages. We use the average values as the offsets.

Note that the MPU6050 is quite sensitive. Gyro offsets once determined, wont change much. The Z_{acc} value is particularly important. Rotate the robot about its vertical axis and average the min/max average readings.

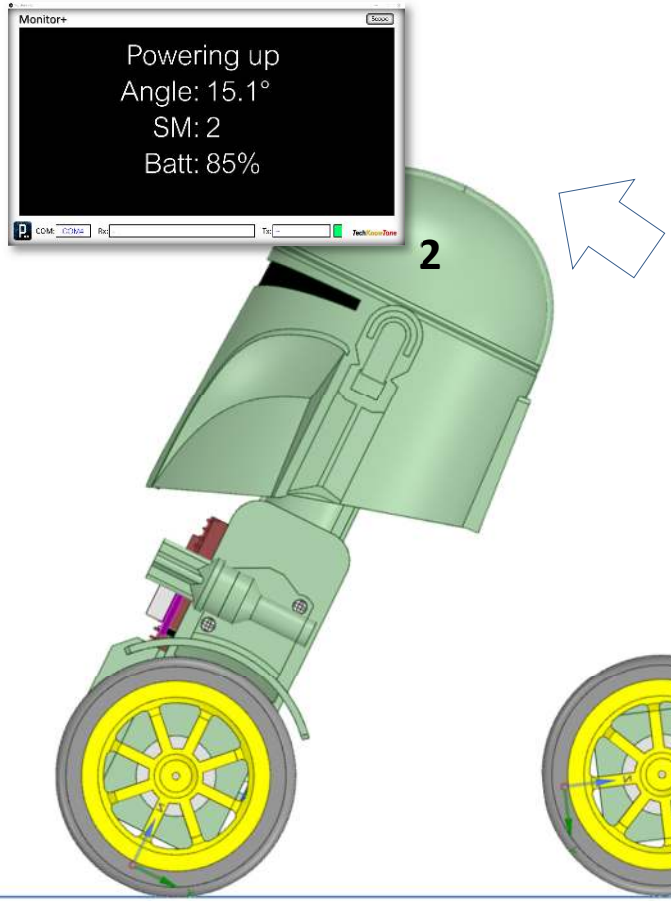


Issue: 1.0 Released: 30/10/2023 TechKnowTone

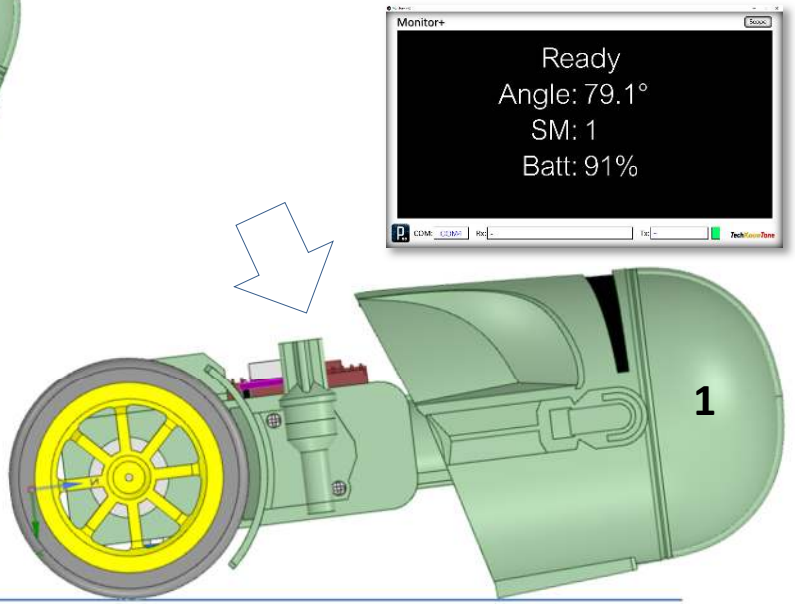
Safe Modes Orientation



Vertical, within setpoint range, Safe Mode transitions from 3 to 4.
Safe Mode 3 – PID output is ramped up from 50% to 100%.
Safe Mode 4 – PID is in full self-balance mode. Here we look for toppling.

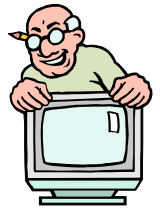


Safe Mode 1 transitions to Safe Mode 2 once the vertical angle is reduced below 20°. Safe Mode 2 then applies low speed drive to take the robot to its vertical position. This aids the self-balance initiation process.



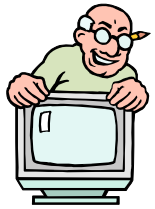
Lay the robot down on its back.
Safe Mode 0 – robot is looking for lying down condition, with a vertical angle of >= 70°. Once laid down for a 2 second period, the robot then enters Safe Mode 1.

Working from right to left, this explains the process of initialising the robots self-balancing mode.

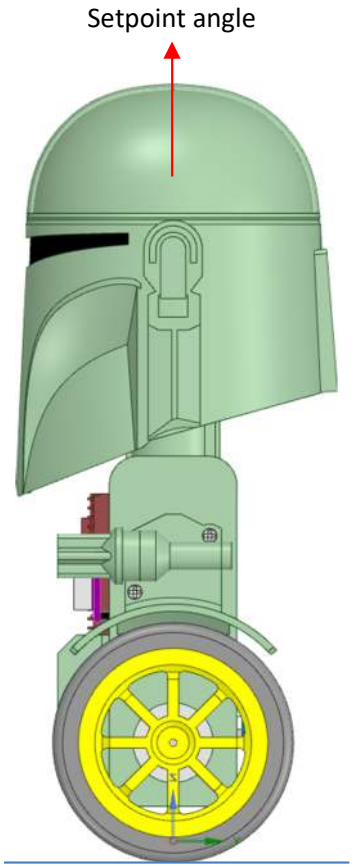


Note: if the left-hand button is pressed at any time the robot is forced into Safe Mode 0. This enables you to make the robot safe, for picking up, when it has been laid down and entered Safe Mode 1. The LED colours, and sequences, change to give the user feedback as to which Safe Mode they are in. The Monitor+ app, when connected over Wi-Fi, provides very useful information on robot angles and Safe Mode values, as shown in the diagrams above.

MandalBot Pitch PID Adjustment



The aim is to balance the MandalBot using its internal PID controller code, when the robot is vertical, at 0° facing forwards. The angle at which a robot will balance is based on its shape, and the distribution of weight with it. In other words, its centre of gravity must be directly above the wheel axle axis for it to be balance.



The angle at which this balance condition occurs is known as the setpoint for the system, and the code is designed to find this value automatically, based on the movement of the robot as the PID controller drives the stepper motors and tries to balance it. Once this is determined experimentally, we set this as the start angle, such that the robot goes easily into the self-balancing condition from rest.

Start by switching off your I-gain and D-gain values, and set P-gain to be 10.00 using the Monitor+ app. Move the robot through its start-up procedure, to get into Safe Mode 4 and have the PID controller driving the motors. When you lean the robot, it should want to drive in that direction. As you increase the P-gain it should become more responsive, and get to a point where it is almost self-balancing. But beyond this point, increasing P-gain further will lead to instability. Determine this point, then back-off the P-gain to around 60%. Then start to increase the I-gain and D-gain values, which should take you into self-balancing. The I-gain integrates the angular error, there by accumulating small errors around the balance point and causing the PID controller to respond to them, when the P-gain alone would not. The D-gain acts to prevent overshoot, cause by the accumulating I-gain, ands there by improve stability.

Once your robot is self-balancing, you can look at the Safe Mode screen to see what angle the self-balance setpoint SbSp has been driven towards, and use this as the start angle going forward.

Here you see the values I arrived at in tuning my robot, and these are stored as defaults in the code. You should arrive at similar figures.

Your robot will balance with a range of numbers, so here I am trying to get it to be quite responsive to external pushes, or obstacles when driving.