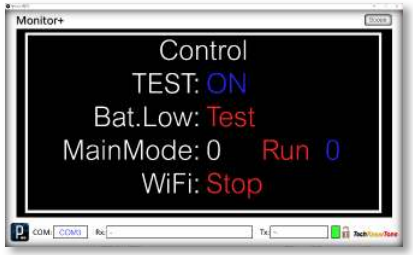
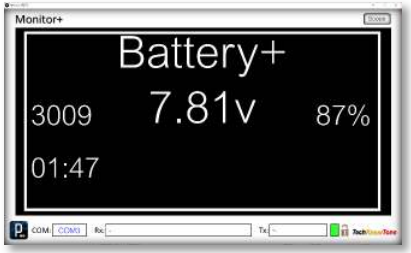
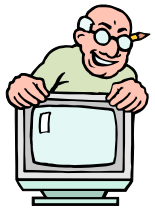
BallBot 4x4 Strider

Calibration





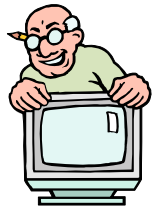
Monitor+



Some useful information on how to tune the BallBot 4x4 Strider robot.

Issue: 1.0 Released: 19/12/2025 TechKnowTone

CAUTION



Lithium batteries can be extremely dangerous, if not handled and cared for properly. This design does not include any form of current limiting circuit, like a fuse. So, care must be taken to ensure that the wiring guidelines are followed accurately, that checks are made for short-circuits, and that battery polarities are marked, and they are inserted the correct way round. Failure to do so, could result in an explosive fire.

Charging Practices: Always remove batteries from your project to charge them. Use a charger, designed for the battery used, and from a trusted supplier. Choose a flat, non-flammable surface to charge on, away from flammable materials. Never leave unattended when charging. Don't charge overnight. Monitor charging to ensure charge characteristics are as expected. Only pair batteries with similar characteristics. Do not overcharge, or leave charging for prolonged periods. This increases the risk of damage and fire.

Battery care & maintenance: Stop using a battery if it is swollen, damaged, dented or leaking. Never charge a damaged battery. Never allow a Lithium battery to discharge below 3.2 volts, as cell damage will occur. Avoid extreme temperatures. Do not charge or store batteries in very hot or cold environments. Don't cover batteries whilst charging, as this can trap heat, causing overheating.

In case of fire: Get out and stay out. If a fire starts, leave immediately, and call the fire brigade. For low voltage Lithium batteries, water is a safe extinguisher.

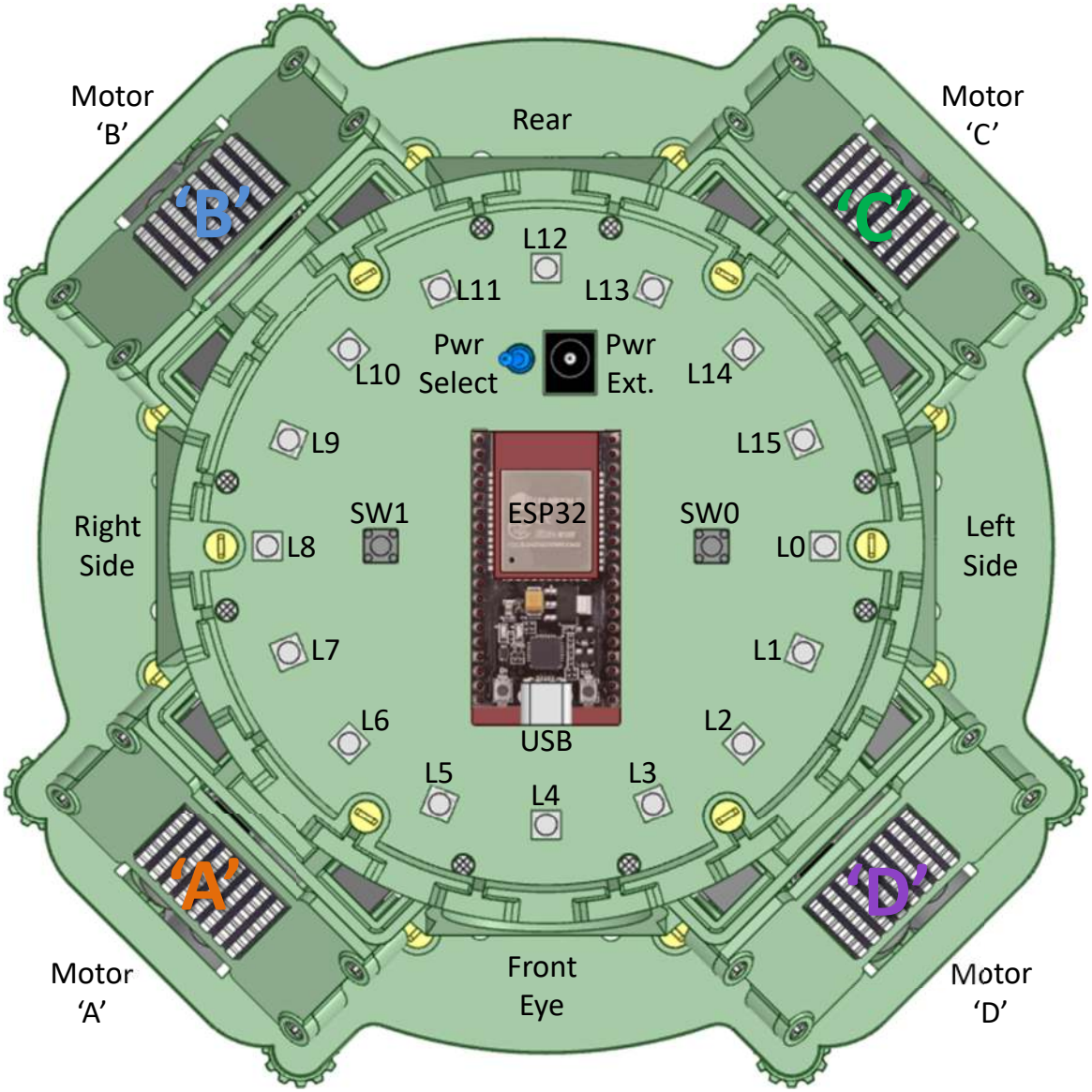
Built-in Monitoring: Most of my project designs include code, and circuitry, to monitor battery voltage, whilst in use. This code then seeks to alert the operator, when the battery has reached a critical low voltage, before shutting down power consuming circuitry; including the micro. Time should therefore be spent on calibrating this feature, as a precaution, for good battery management and maintenance.

Carefully dispose of batteries that damaged, or discharged below their critical voltage.

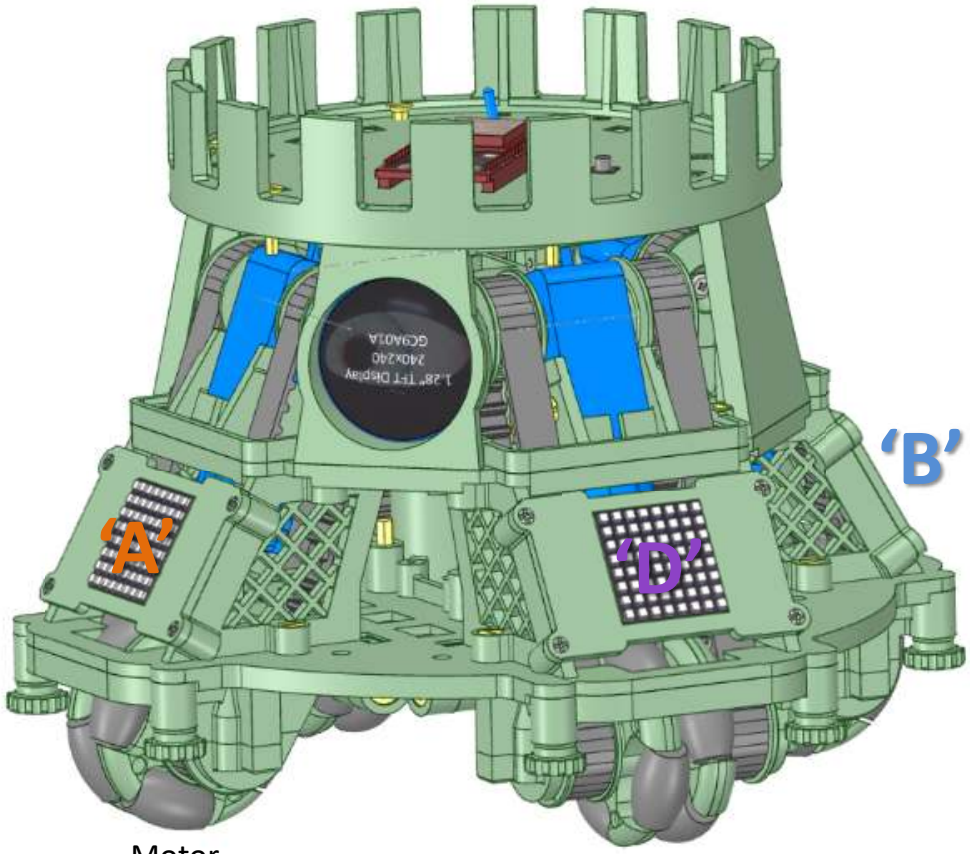
Component Reference

Not really a calibration process, but it will be useful to refer to this diagram when coding LED patterns, modifying motor drive routines, or reading sensors.

The diagram shows component assignments and conventions used in this design, and their relationship to coding conventions.



Top View

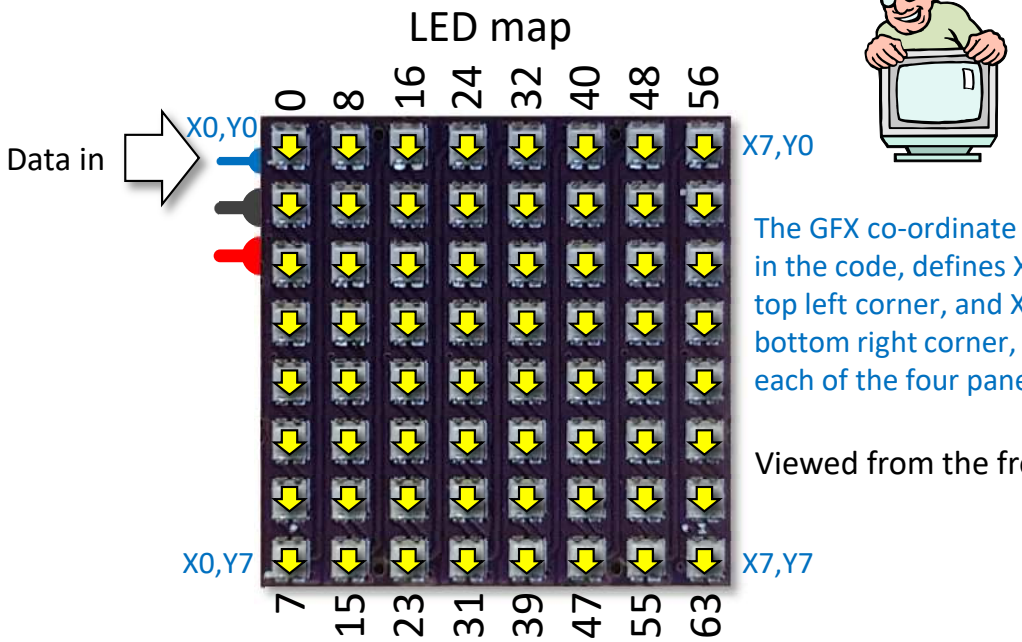


Front View

Issue: 1.0 Released: 19/12/2025 TechKnowTone

GFX and LED Maps

The code contains a simple graphics library (DS_GFX) which contains functions that help you to draw and manipulate pixels on the DotStar panels. This code assumes the mapping of the LED pixels in panels, as shown here; so, the orientation and connection of panels must comply with this., to get the correct results



The GFX co-ordinate system, in the code, defines X0,Y0 as top left corner, and X7,Y7 as bottom right corner, for each of the four panels.

Viewed from the front

The GFX library functions are used by different LED modes, to create interesting patterns, and simple animations. Here are some examples of the functions available:

```
DotStar GFX functions:

GFX_Bar(P,X,Y,W,H)          - draw a solid bar on panel P at X,Y, width W, height H in 32-bit DotColr
GFX_Box(P,X,Y,W,H)          - draw a box on panel P at X,Y, width W, height H in 32-bit DotColr
GFX_ClrXY(X,Y)              - clears a pixel on LedFrame to the current background colour BakColr
GFX_ClrXYP(F,X,Y)           - clears a pixel on frame F to the current background colour BakColr
GFX_ClrXY(P,X,Y)            - clears a pixel on panel P to the 32-bit DotColr
GFX_CopyPanel(PF,PT)        - copy the contents of panel PF to panel PT
GFX_CopyPanelFlipV(PF,PT)   - copy the contents of panel PF to panel PT, flipped vertically
Gfx_DrawChar(F,X,Y,C,M)     - draws a single character C on frame F, at X,Y using mode M
Gfx_DrawGFX(F,X,Y,C,M)      - draws a single GFX character C on frame F, at X,Y using mode M
GFX_GetXY(X,Y)              - returns the pixel pointer for X,Y in range 0 - 63
GFX_GetColor(P,X,Y)         - puts the colour of pixel at X,Y on panel P in 32-bit DotColr
GFX_FillColor(P,C)          - fills panel P with 32-bit colour C
GFX_FillRGB(P,R,G,B)        - fills panel P with RGB colours
GFX_FillRow(P,R,Col32)      - fills row R on panel P with 32-bit colour Col32
GFX_Line(P,X0,Y0,X1,Y1)     - draw a line on panel P from X0,Y0 to X1,Y1 in 32-bit DotColr
GFX_RollDwn(P)              - roll the contents of panel P down vertically
GFX_RollUp(P)               - roll the contents of panel P up vertically
Gfx_SetBakXY(X,Y)           - sets a pixel on the current frame to the BakColr colours
Gfx_SetXY(X,Y)              - sets a pixel on the current frame to the DotColr colours
Gfx_SetXYRGB(X,Y)           - sets a pixel on the current frame to the current RGB colours, ColR, ColG, ColB
```

Battery Voltage Calibration

See Lithium discharge curve obtained from the internet. In this analysis the lipo battery consists of two identical batteries connected in series.
Assume fully charged 8.2v battery max voltage is V_{BM} >= 8.4v max (charging)
Set battery warning point at V_{BW} = 7.2v (2 x 3.6v)
Set battery critical point at V_{BC} = 6.6v (2 x3.3v), don't go below this!

The ESP32 is powered via a 5v voltage regulator, connected to the V_{in} pin, but the 6k8 supply sampling resistor is connected to source V_{Batt} or Ext. supply.
For ESP32 V_{ADC} == 4095 on 12-bit converter (4095 max).
If we use a 6k8 resistor feeding A0 and a 3k3 resistor to GND, we get a conversion factor of 10.1v == 4095, or 2.47mV/bit, or 405.4 bit/v
Using a Multimeter and a variable DC supply, I determined the following V_{ADC} values for corresponding threshold voltages:

MAX. O.C V_{OC} = 8.4v, gave A0 = 3363 On V_{ADC} (2 x 4.2v)

MAX: (100%) V_M = 8.2v, gave A0 = 3620 on V_{ADC} (2 x 4.1v)

HIGH: (80%) V_H = 7.8v, gave A0 = 3083 on V_{ADC} (2 x 3.9v)

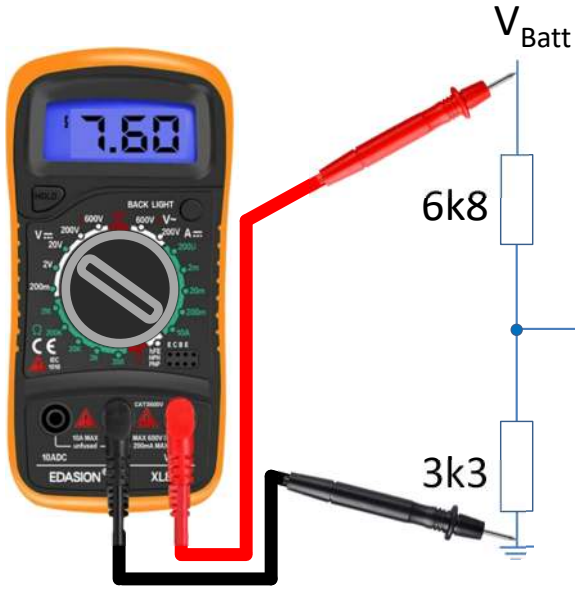
WARNING: (20%) V_{BW} = 7.2v, gives A0 = 2820 on V_{ADC} (2 x 3.6v)

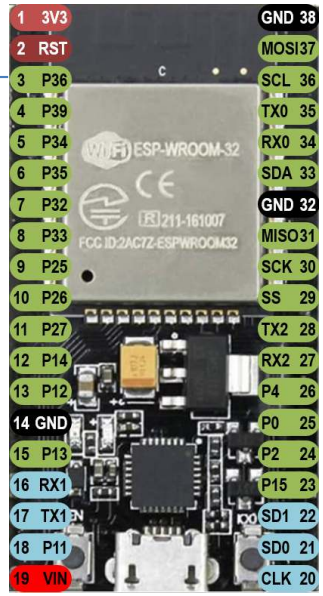
CRITICAL: (0%) V_{BC} = 6.6v, gives A0 = 25590 on V_{ADC} (2 x 3.3v)

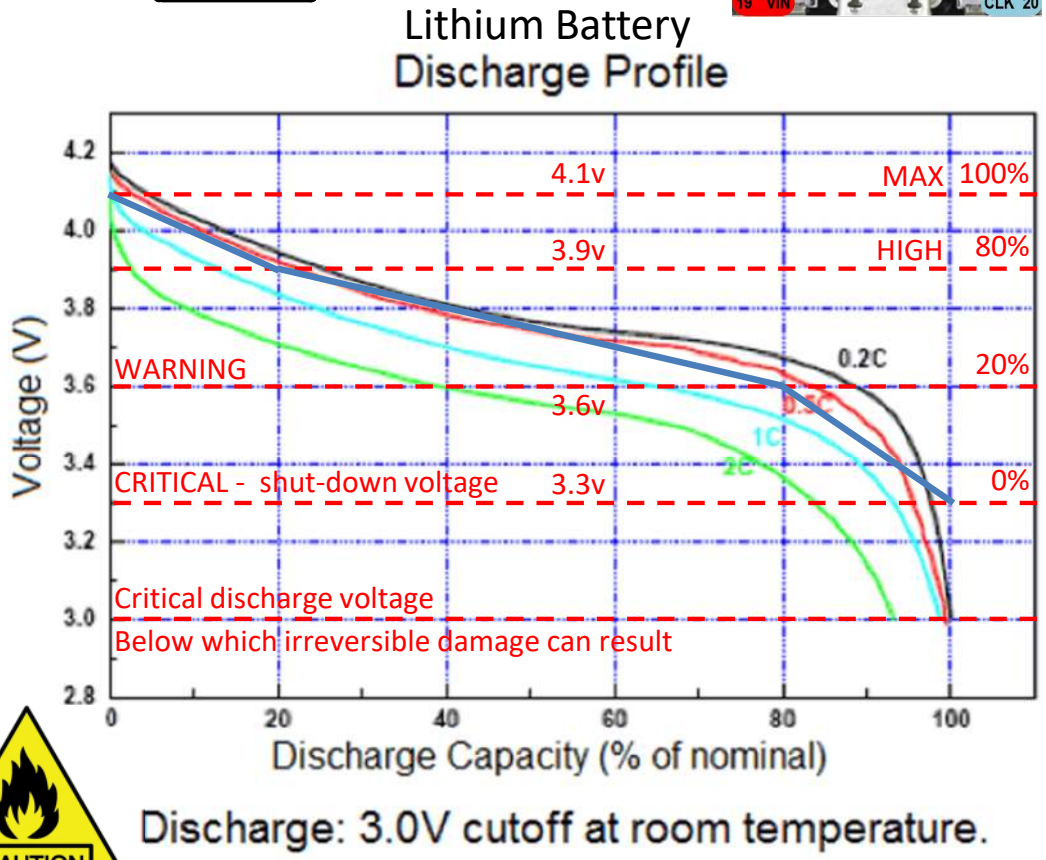
The code will sample the battery voltage on power-up to ensure it is sufficient, then at every 40ms interval, calculating an average (1/50) to remove noise. Then converts ADC values to voltage in the getBatV() function.

In the code I have assumed a discharge curve ranging from 8.2v (100%) to 6.6v (0%) capacity, using the blue overlay line shown. The voltage is monitored and used to predict the remaining capacity of the battery in use.

Note: If connected to USB port with internal battery switched OFF the ADC will read a value 5 volts (A0 = 1919) or less. So, if the micro starts with such a low reading it knows that it is on USB power, which limits functions available.



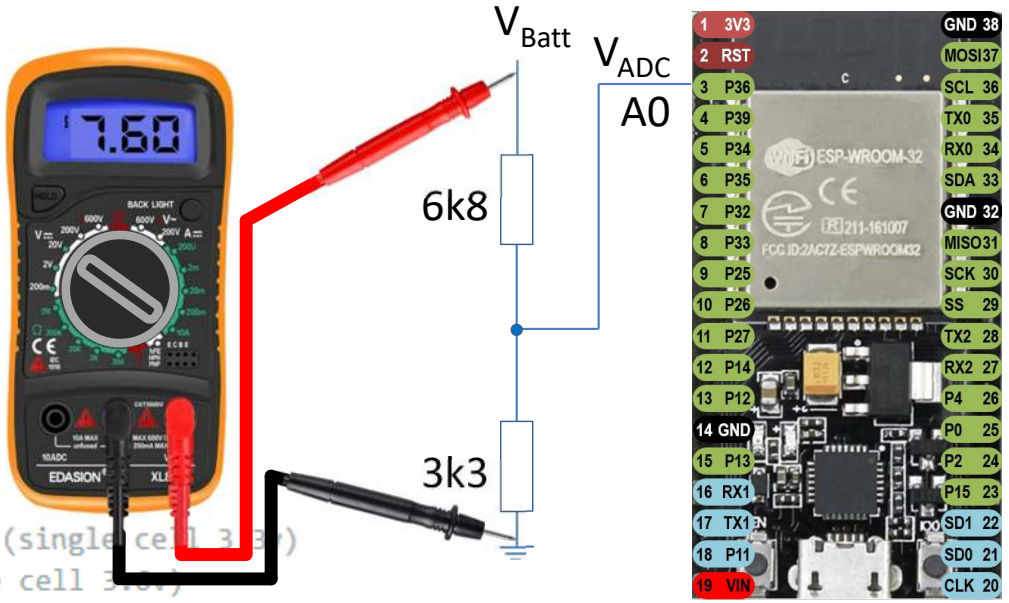




Issue: 1.0 Released: 19/12/2025 TechKnowTone

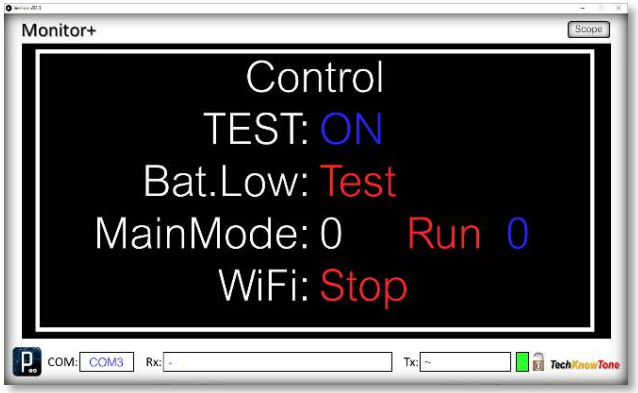
Battery Voltage Calibration Continued

To take these measurements, use the Monitor+ app, to navigate to the Control screen, and select TEST and ON. This forces the code into TEST mode, and a box is then drawn around all of the Monitor+ screens. In TEST mode the robot shut-down function will not be disabled, making it possible to reduce the supply voltage, and take those readings. Power the robot from both an external adjustable power supply, and from a USB cable. Start with the supply set to 8.4v, confirmed by the multimeter, and record the ADC value shown on the Battery+ screen.



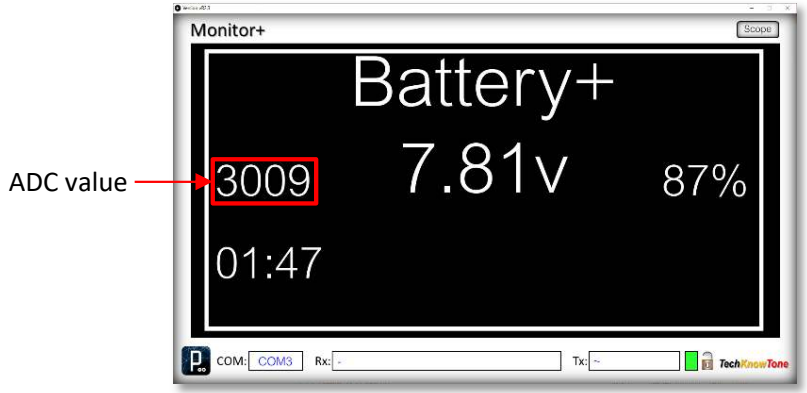
Then repeat this process, by reducing the supplied voltage, for each of the key values, and record them in the code:

```
#define Bat6v6 2559  // 6.6v critical battery threshold, shutdown (single cell 3.3v)
#define Bat7v2 2820  // 7.2v battery threshold 20% warning (single cell 3.6v)
#define Bat7v8 3083  // 7.8v battery threshold 80% (single cell 3.9v)
#define Bat8v2 3260  // 8.2v fully charged battery voltage, light load
#define Bat8v4 3363  // 8.4v fully charged battery voltage, O/C
#define BatUSB 1500  // minimum triggers US supply assumption
```



Normally, to prevent damage to batteries, if the voltage falls below the CRITICAL value, the system will shut-down many features. But, in TEST mode this feature is disabled, so that the ADC value at that critical voltage can be determined.

In normal mode, if the power returns, the code will detect this, and perform a RESTART.



Issue: 1.0 Released: 19/12/2025 TechKnowTone

Motor PWM Drive Mapping

Depending on the tightness of the drive belts, there can be a significant amount of start-up stiction in the drive system. This non-linear effect can have a significant impact on the PID controllers' ability to balance the robot on a ball. To reduce this effect, in code, we re-map the motor demand onto a profile, defined by two variables, MtrMin and MtrMinStart.

If they are set equal, the mapping process has no effect. But if MtrMin is larger, then this has the effect of producing larger PWM output values, below the MtrMinStart value. By using the Monitor+ app, in TEST mode, you can adjust these values whilst the robot is actually balancing, to see the benefit of this feature. Always start with them being equal, with a value in the region of 25 (ie. 10% of total PWM output).

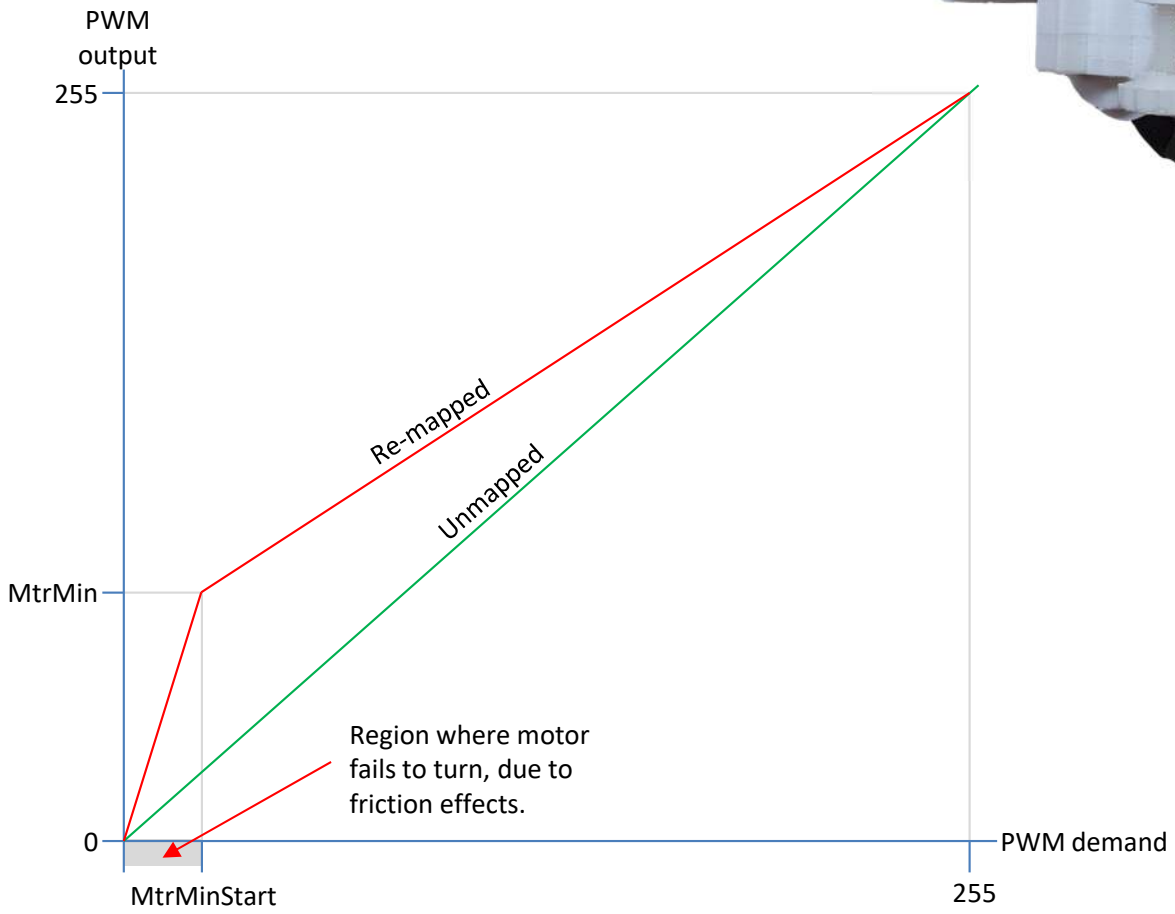
You may end up with values like this:

MtrMin = 60

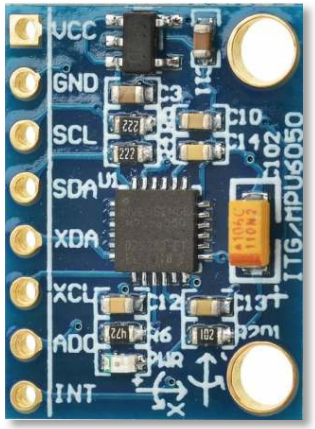
MtrMinStart = 20

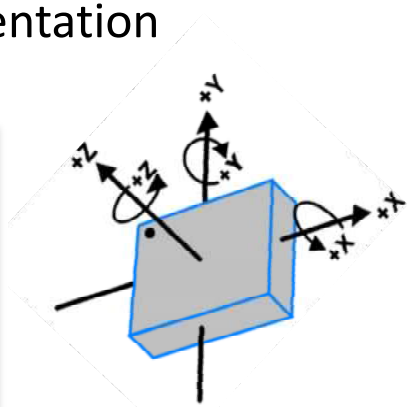
This gives you a threefold increase in PWM power, at demands below PWM = 20.

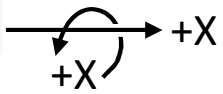
If overdone however, the downside could mean that the system becomes more unstable around the balance point, and has less gain at higher demands. So, adjust these values carefully!

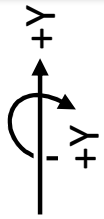


MPU-6050 Orientation

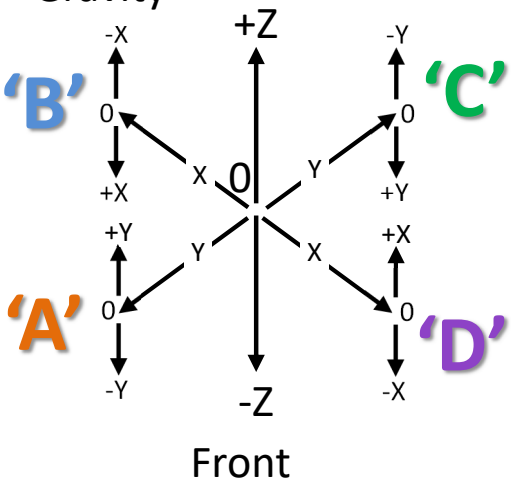


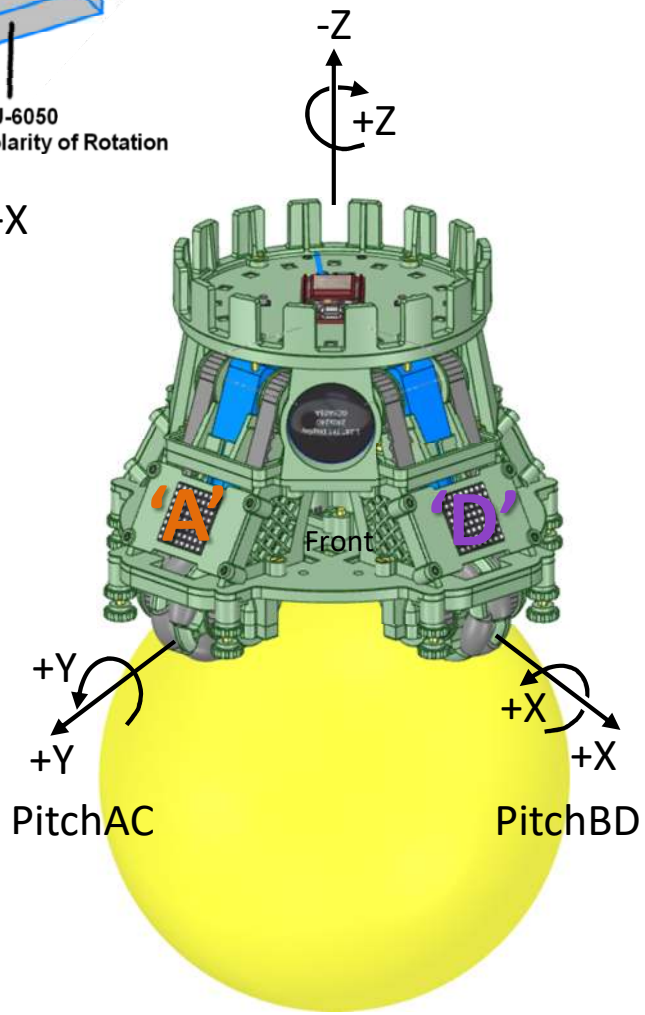


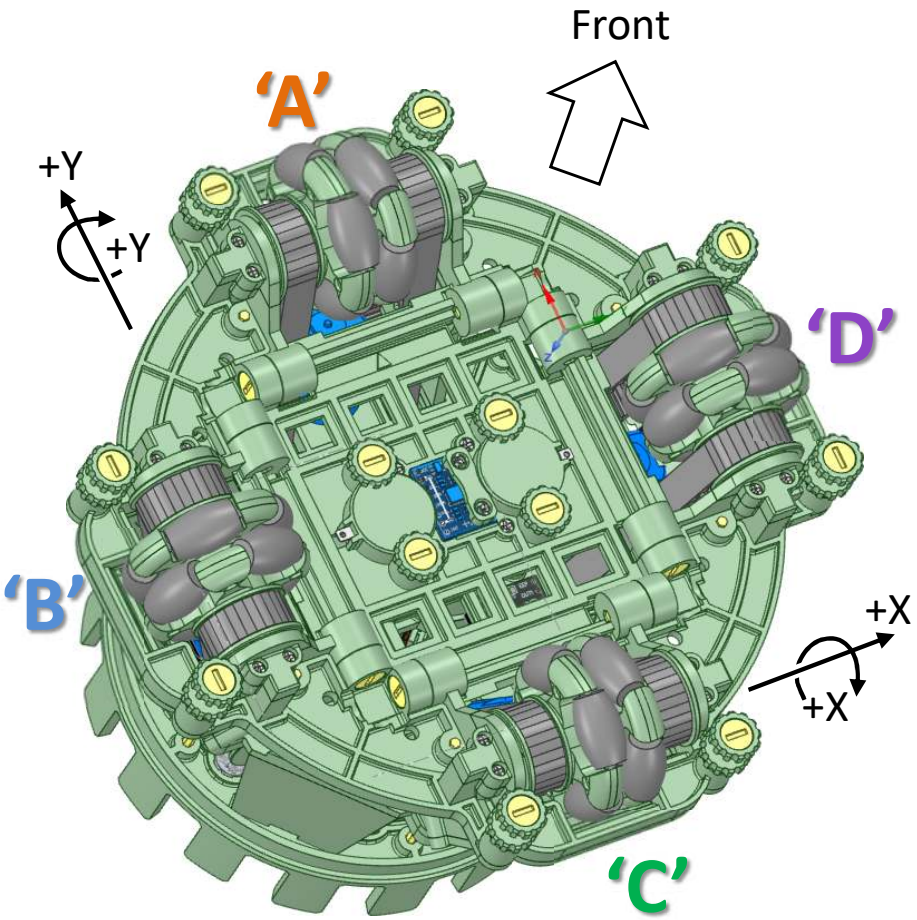




Gravity







Gyroscopes are set at +/-250 °/sec FSD.
Hence at 32,767 FSD; rotation rate of 1 °/sec = 131.
To convert this to a gyro angle we use the time between readings.
On 4ms cycle we would accumulate a count of 1310 over 250 cycles, when rotating at 1 °/sec.
So, delta angle per 4ms cycle = gyro rate * 0.00007633

PitchAC is positive when motor 'A' is raised above motor 'C'.
PitchBD is positive when motor 'D' is raised above motor 'B'.

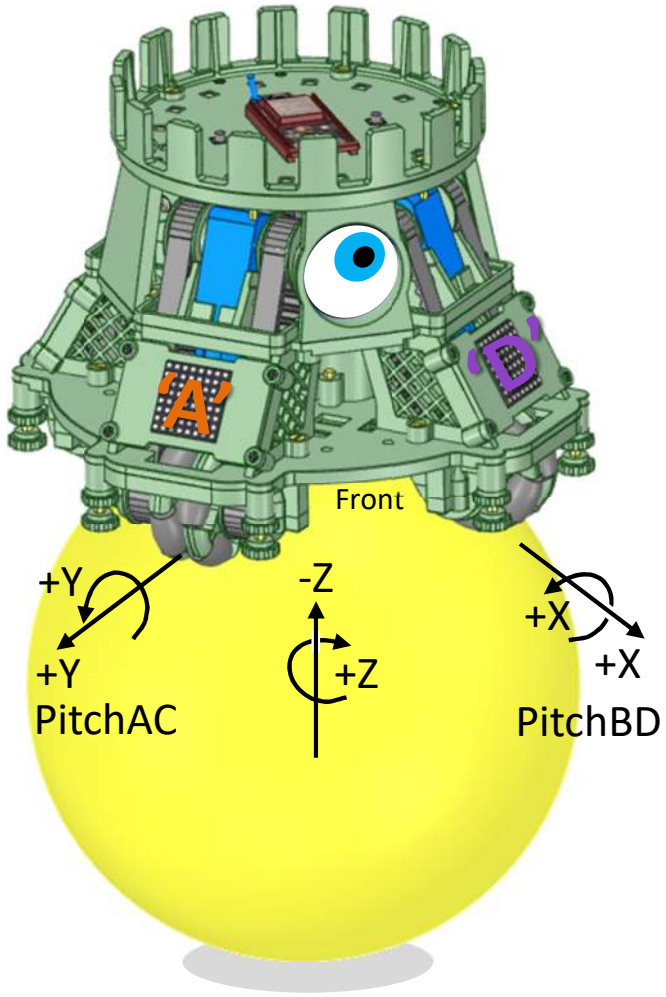
PID Controller Tuning

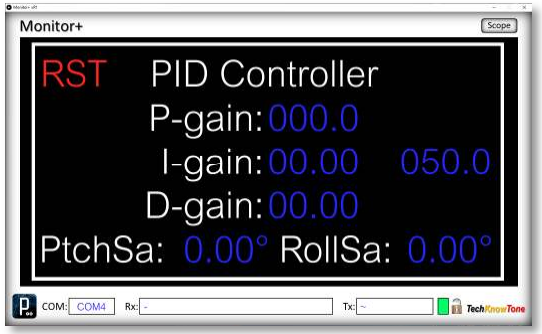
The MPU6050 motion sensor enables the BallBot to self-balance by providing data, at a high rate (250Hz), which is converted into a vertical angle and compared with a setpoint variable. The resulting difference (error) is used by the two PID controllers to drive the four wheels. If the control system senses that the robot is pitching forwards, in either pitch direction, it will drive the corresponding wheels to minimise the error, and similarly backwards, if it senses that it is pitching backwards. The same principal applies to both pitch axis. There are in effect two identical PID controllers, which both aim to maintain equilibrium about the setpoint angle. Both PID controllers use the same gain settings, but the balancing setpoints are likely to be slightly different, due to physical variations, in the robots build.

Each PID controller applies three gain settings to the angular Pitch and Roll error signals. A **P**roportional gain, which simply multiplies the error signals by a gain factor. An **I**ntegral gain, which effectively accumulates small errors, such that if the error is small near the setpoint target, it will become larger quite quickly over time. A **D**erivative gain, which in effect provides a response braking function, to avoid overshoot and instability, resulting from the other two gains.

To tune the BallBot's PID controller, run the Monitor+ app with your PC connected to the Wii transceiver serial port. This enables the PID controller app to send control messages directly to your BallBot over Wi-Fi in a hands-off fashion. Start with all PID gain variables set to zero.

[1] Initiate the balance state by placing the robot on top of the ball, in a vertical state. If it is within +/- 5° when you press SW1, the LEDs will turn yellow, entering the SafeMode 1 state. Then tilt the robot out of the vertical by 6°+, to shift to the spirit level SafeMode 2 state. Then use the LED lamps to find the vertical position, and the ACTIVE state, SafeMode 4 will be reached. With all PID variables at zero, the motors should not respond. [2] now slowly increase the P-gain value, whilst physically moving the robot forwards and backwards. You should start to feel drive from the motors assisting your movements. [3] increasing the P-gain will take you to a point where the robot almost self-balances, but more gain beyond that point makes the whole system go unstable. [4] repeat, and note the highest P-gain value possible, just before instability kicks in. Then back off the P-gain by about 33%. [5] now with P-gain set, slowly increase the I-gain value. You should find a point at which self-balance occurs; but again, increasing I-gain further will cause instability. Note the best I-gain value for self balance. The D-gain adjustment, dampens the response, to prevent overshoot and instability.



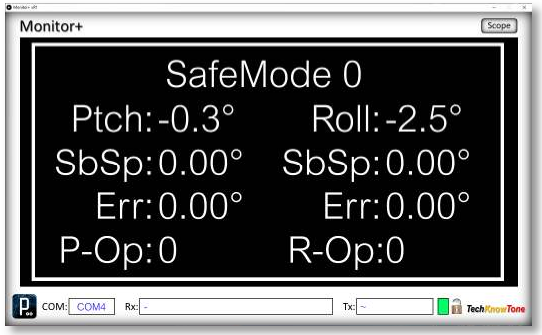


Monitor+ app

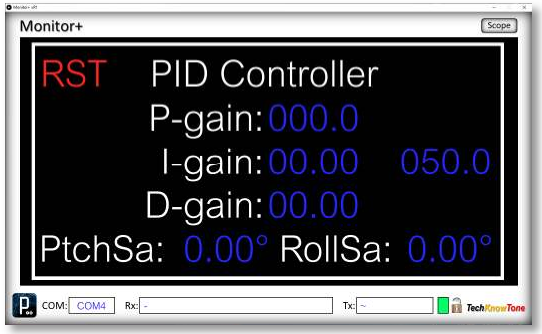
PID Controller Tuning Continued

[6] you can now bring the D-gain variable into play. Increasing D-gain should allow you to have higher values of I-gain and P-gain, before instability occurs. Higher values should mean that the robot effectively stiffens up about the setpoint angle, and doesn't wander or vibrate at that point.

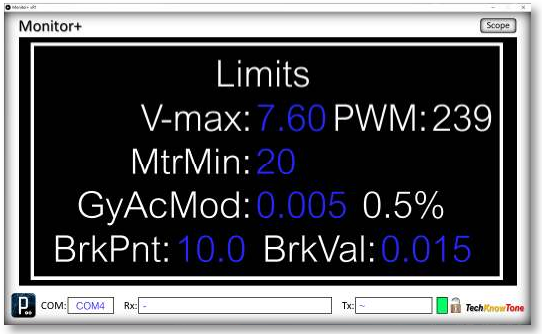
The PID gain values are applied equally to both the Pitch controllers, as they are symmetrical systems; albeit operating on different axis. This tuning method is empirical, as all robots will have different setpoints, and there is no steadfast way of arriving at the PID gain values. Once you have determined the three gain numbers, they can then be entered into the C++ code in the table of definitions.



The Monitor+ SafeMode display shows the Pitch and Roll angles, their respective self-balancing setpoints their angular error values, and their PID outputs which are applied to the motors. Note that the self-balancing setpoints are derived automatically whilst the robot is moving, to ensure balance. Once you get your robot balancing in a reliable fashion, you can look at the self-balancing setpoints. You use these two values as start angles for your robot, as this will make it much easier to initiate balance, and avoid the robot from lurching at the start.



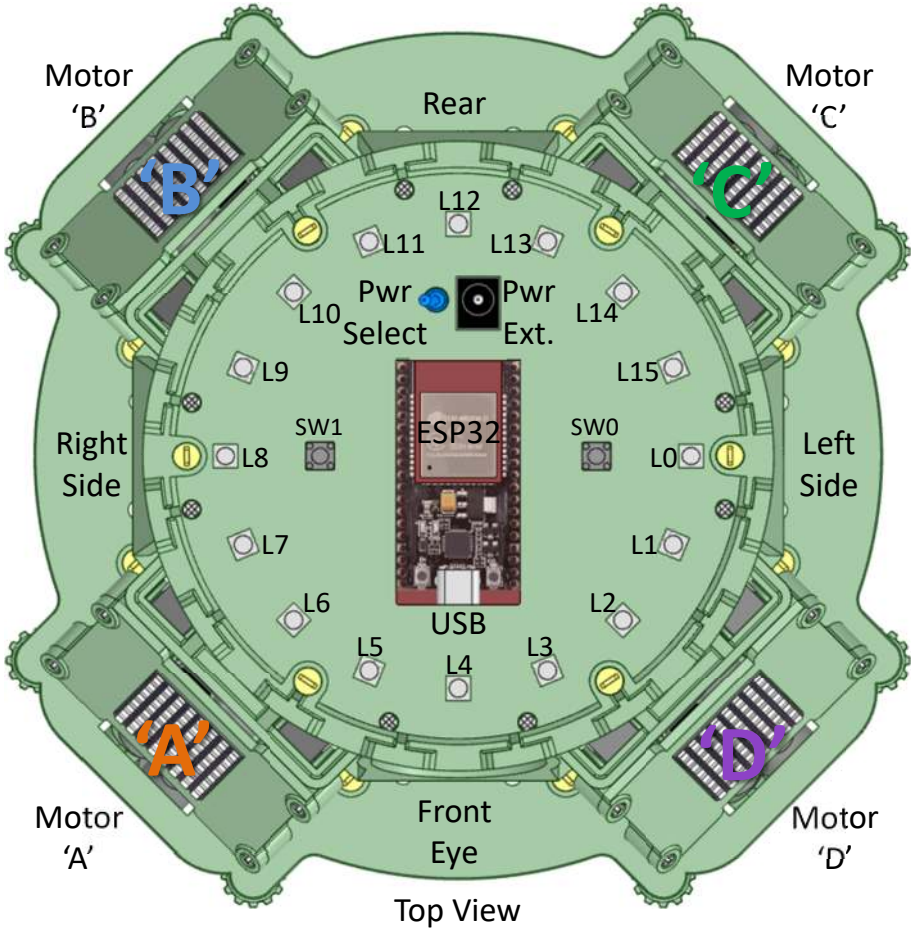
The Monitor+ display screens that have coloured text, indicates that the text fields can be clicked to invoke an action. Clicking on the **RST** field will reset all of the values to the defaults defined in the code. Whilst clicking on **blue** fields will change their values, with immediate effect. Each digit can be adjusted via the left/right mouse button. The field to the right of the I-gain value is the Imax limit. Initially set to **050.0**, to make the robot less lively; and later set it to **255.0** to improve overall stability, particularly when driving along.



The Limits display provides the ability to adjust V-max, which is the nominal battery voltage. If the battery voltage exceeds this limit, the motor PWM values are reduced proportionately, so that the motor does not provide more power than intended. Due to motor gearbox friction, there will be PWM values below which the motor shafts will not turn. This dead spot can be removed by setting MtrMin to a value around 20. Avoid setting this too high. The GyAcMod value is the amount of accelerometer angle, which is used to correct the gyro angles; which has no absolute value. It needs to be enough to correct any gyro drift, but too high a value will inject vibration noise into the PID error signals. The BrkPnt and BrkVal brake values, prevent the PID controllers from being wound up, when the robot is pushed along on a slope, or by hand.

Spirit Level Functions

This is how the spirit level code is mapped out, in terms of the number of LEDs that are lit.

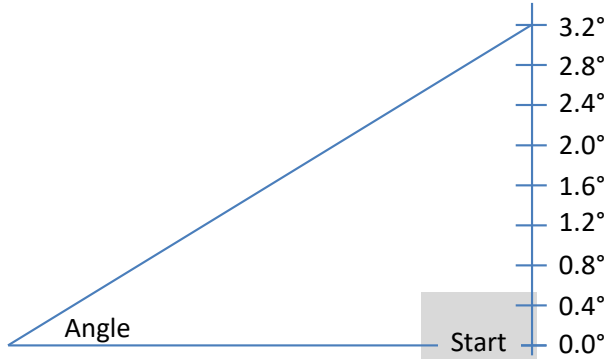


Spirit level LEDs are driven depending on the PitchAC and PitchBD angles. We light more LEDs when the difference in the two angles is small, as this is when the robot is close to being vertical.

PitchAC 'A' high point, light LED L6.
PitchAC 'C' high point, light LED 14.
PitchBD 'D' high point, light LED L2.
PitchBD 'B' high point, Light L10.

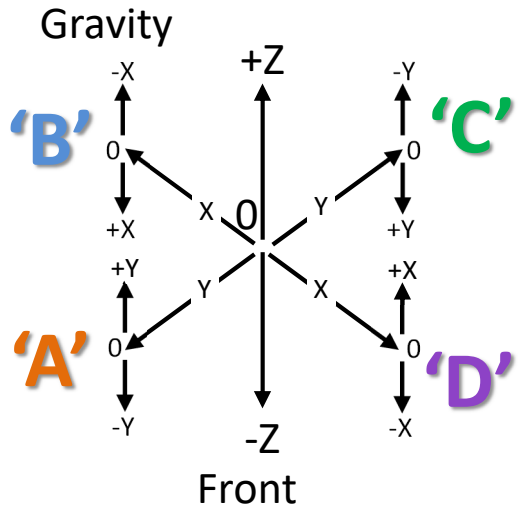
There are 9 levels displayed for max error angle:

1 LED @ > 2.7°
3 LEDs @ <= 2.7°
5 LEDs @ <= 2.4°
7 LEDs @ <= 2.1°
9 LEDs @ <= 1.8°
11 LEDs @ <= 1.5°
13 LEDs @ <= 1.2°
15 LEDs @ <= 0.9°
16 LEDs @ <= 0.6°



The LEDs are centred on where the max angle error is.
If both angles are equal, then the LED is between them.
We assume a linear relationship between error angles.

In practise the user tilts the robot towards the red LED, to the point where all LEDs have come on.



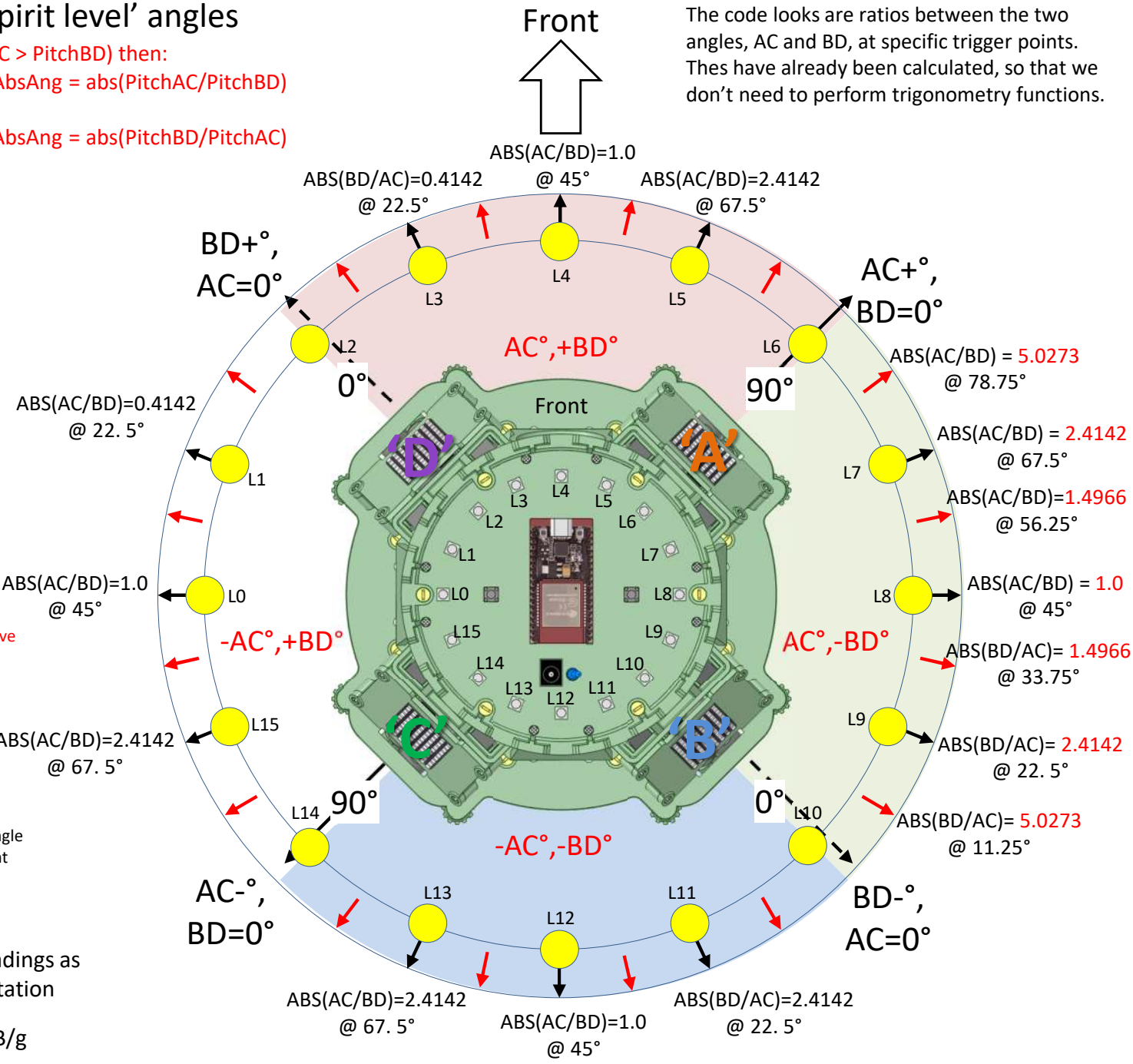
LED 'spirit level' angles

If (PitchAC > PitchBD) then:

AccAbsAng = abs(PitchAC/PitchBD)

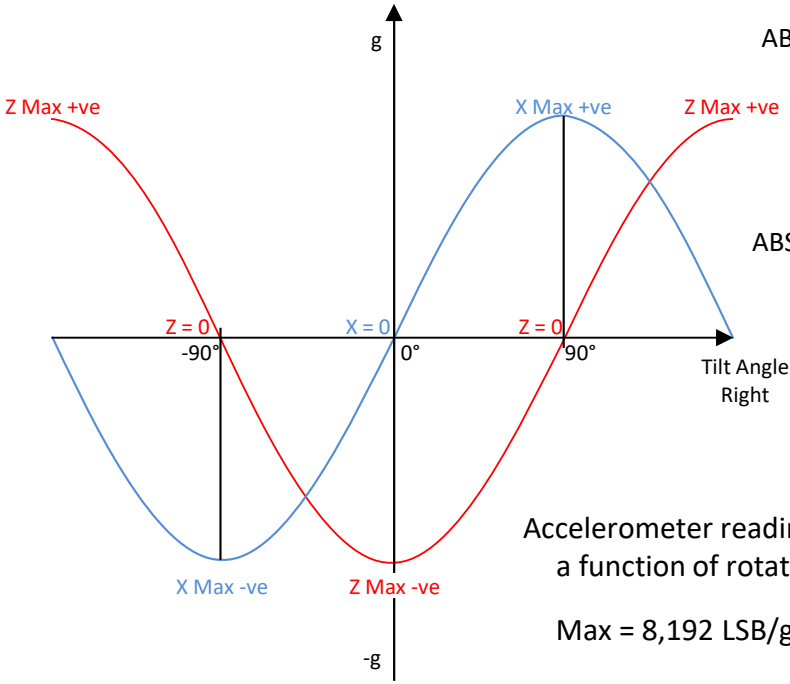
else:

AccAbsAng = abs(PitchBD/PitchAC)



The code looks are ratios between the two angles, AC and BD, at specific trigger points. Thes have already been calculated, so that we don't need to perform trigonometry functions.

Accelerometers read 0 when horizontal, and maximum +/- when vertical. Gravity can be used to aid the measurement of angle, but it is greatly affected by accelerating forces.



Accelerometer readings as a function of rotation

Max = 8,192 LSB/g