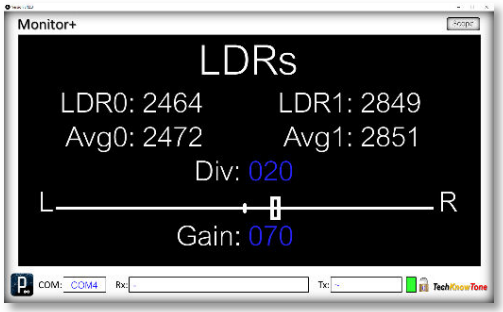
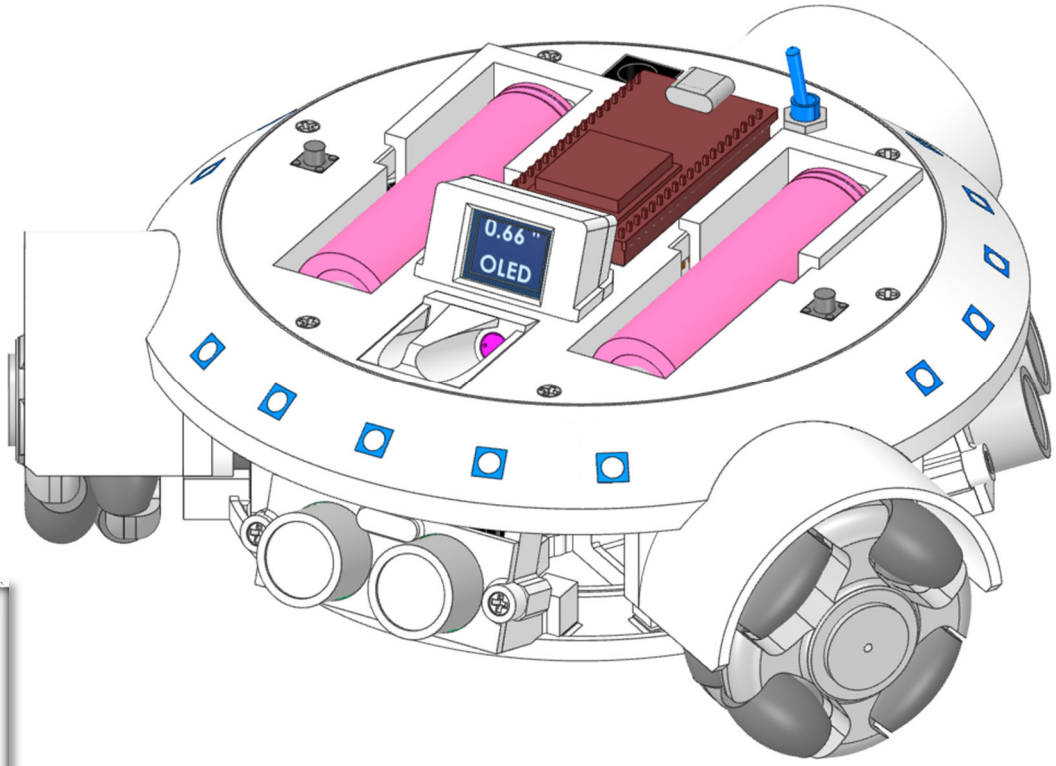
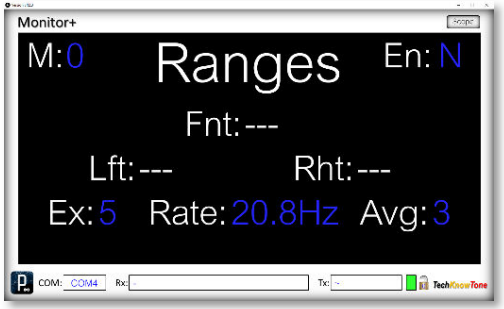
# Omni-Bot 3x3

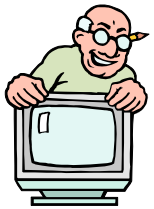
## Calibration



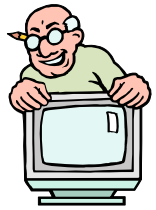





Some useful information on how to tune the Omni-Bot.

# 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.

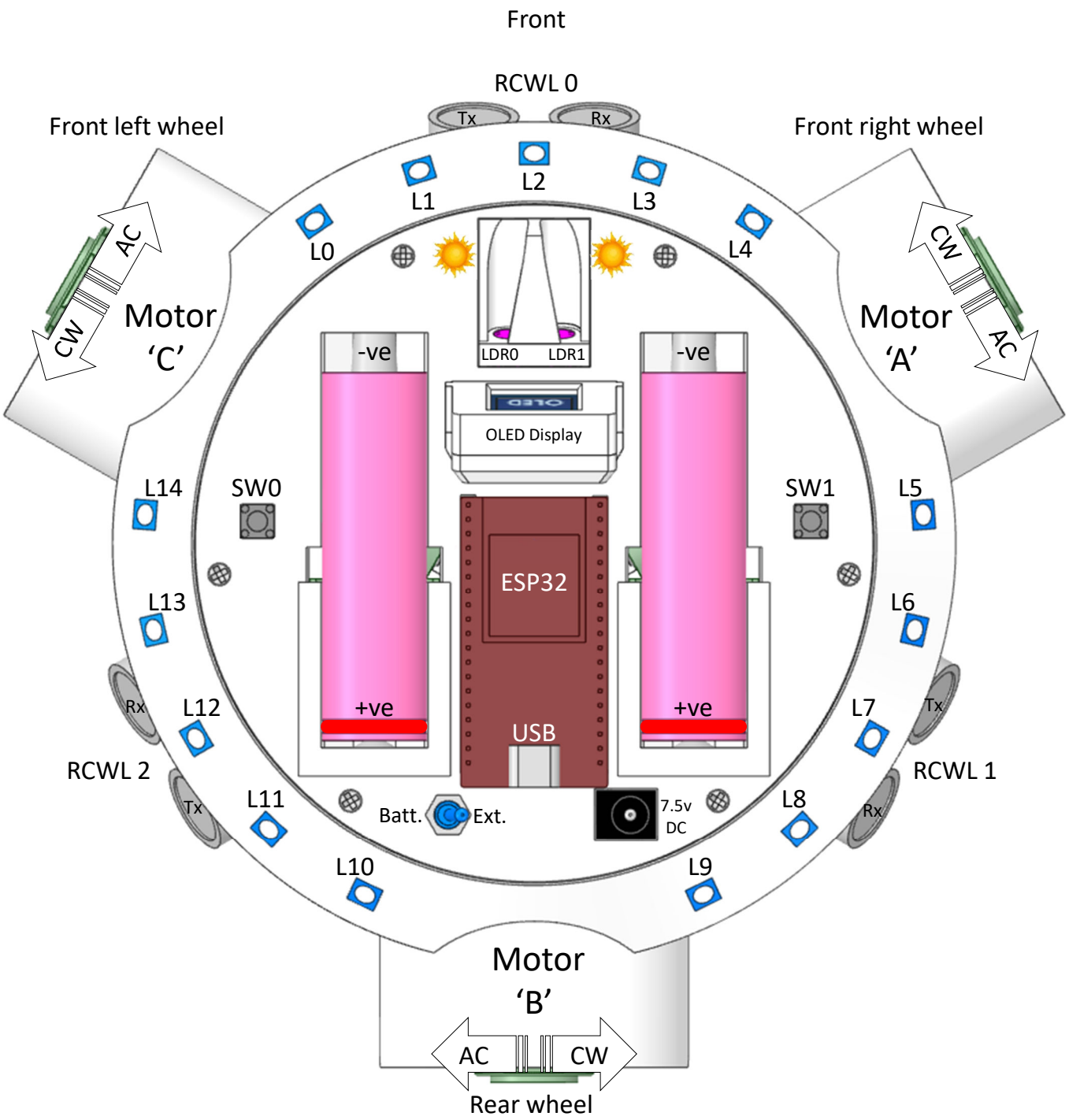
## Component Assignments

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.

Note in particular the orientation of the two 18650 Lithium batteries. It is recommended that you mark the positive ends of each battery, with a red indelible pen. And that you also mark the end of the battery holder as a reminder.

There is not fuse or blocking diode in this design. So, the incorrect insertion of the batteries could cause significant damage to the electronics, including the ESP32 microcontroller.







Issue: 1.1 Released: 28/11/2025 TechKnowTone

## Light Sensors

The light sensor assembly , at the front of the Omni-Bot contains two GL5537 photo resistors. As described in the wiring diagrams document, you select two components that have close to identical characteristics, in the region of 5 - 6kΩ, under normal lighting conditions. When the light level is low, the resistance value is high, and the voltage measured on the ESP32's ADC pin approaches the upper limit of 4095. Conversely, when the light level increases, the resistance value falls, moving the ADC value towards zero.

When connected to the Omni-Bot, the Monitor+ app has a display screen, which shows the raw values from the left-hand and right-hand sensors, along with their average values. It also draws a simple bar graphic which moves from left to right, depending on the relative value of the two sensors.

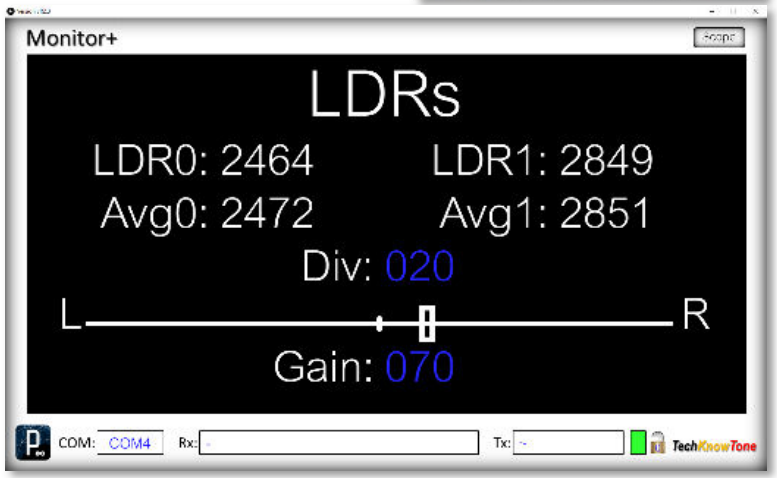
My code assigns blue text to items which can be clicked on, to change their value. On this screen we see that the averaging coefficient '**Div**' is set to 20, and the '**Gain**' of the control system is set to 70.

These photoresistor values are measured at a rate of 100Hz, on an alternating 5ms cycle in Core 0. Increasing the value of Div will reduce the effects of noise and transient changes, but it will also effectively slug or slow down the responsiveness of the system.

The Gain value effectively sets the motor drive response of the system to the difference between the two value. The aim is to adjust these values, to deliver a responsive system, but not one that tends to oscillate about a point of brightness.

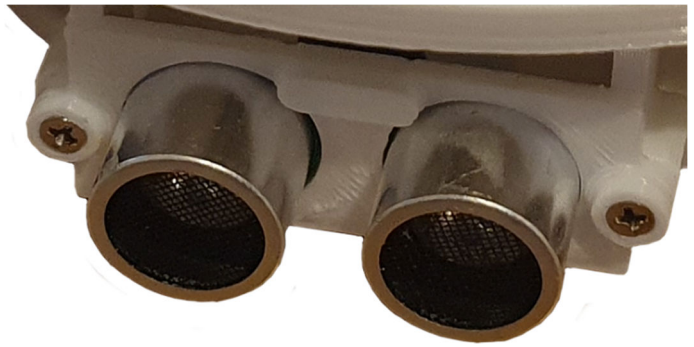
The **values** are shows as 3 digits (020 and 070), as each digit can be clicked on to change its value. Making it easier to go from a low to high setting, in a small number of clicks.


LDR's mounted at the front of the Omni-Bot. The geometry acts to provide shade on one sensor, when the light source is off centre.



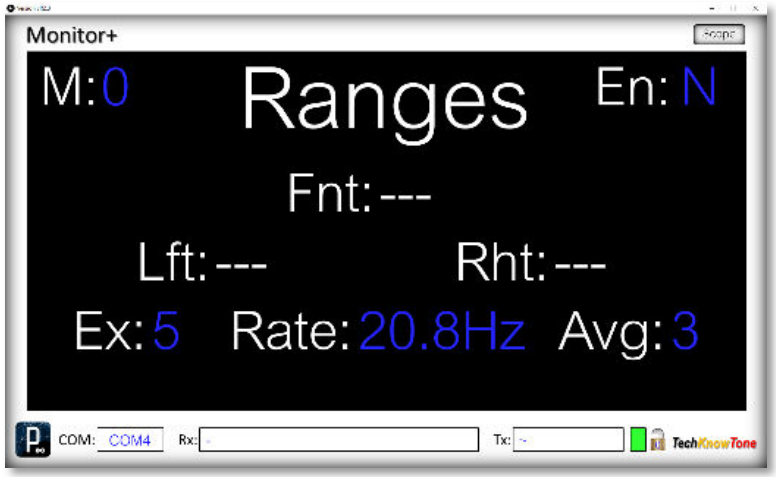
## Range Sensors

There are three RCWL-1601 acoustic range sensors in this design. These devices are similar to the well-known HC-SR04 device, but they work well at 3.3 volts, so are more suited to use with an ESP32 micro.



Originally, I intended to trigger all three devices at once, before reading their individual echo responses. But this idea didn't work too well, due to receiving multiple echoes in close environments. There is a page explaining how the sensor works in my Wiring Diagrams document.

So, they were re-wired such that each sensor can be triggered and read independently. In Monitor+ there is a screen which displays the three sensor range values, ad provides control features:



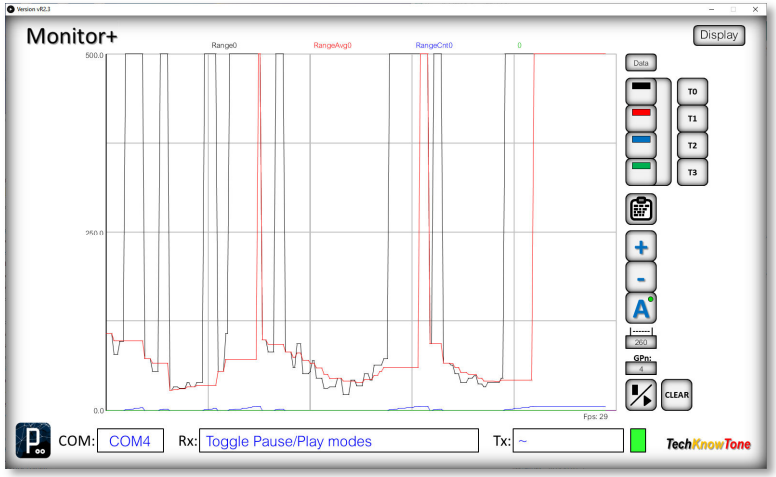

En:**Y** - toggles the sensor reading process ON/Off. Note that this is a status flag, so **Y** = ON, and **N** = OFF.

M:**n** - defines which sensor is active. Front = 0; right = 1; left = 2; all = 3-5, read in a circular fashion; and 6-9 in a pattern of F, R, F, L, F, R….

Ex:**n** - as these sensor can give false readings, particularly at longer ranges, the code has an exclusion filter, which effectively ignores out of range values for a short period of counts; using the previous good value.

Rate:**nHz** - is the frequency at which sensors are being read.

Avg:**n** - is a rolling averaging filter, to smooth consecutive readings.



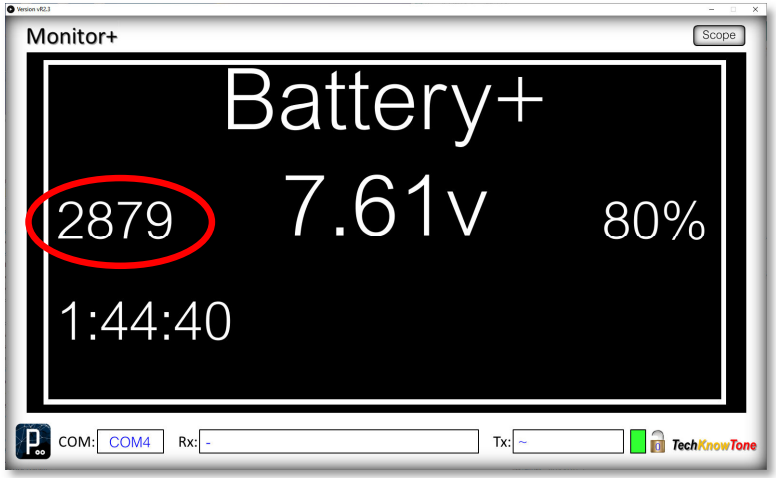

In Monitor+ you can also look at sensor values, using the 'Scope' function, to see how the sensors are performing in real time. They work best with flat surface, but the echo can easily be deflected by angled or round surface. A better option would be to use laser ranging devices.

## Control & Testing

The Monitor+ app can also be used for testing features of the code, and there is a display screen assigned to this.


Rather than pressing the button on the ESP32 micro, you can instruct the code to perform a ‘soft’ reset, even over Wi-Fi. The **RESET:NOW!** Option provides this feature.

In order to protect the life of the Lithium batteries, which suffer damage if taken into deep discharge the code is constantly monitoring battery voltage and checking for this condition. To test the codes reaction, you can use the **Batt: Flat** function; which effectively drops the average reading in the code to zero, thereby trigging a response. The Omni-Bot should display a warning message, then shut down power consuming features like the LEDs and OLED display before sending the micro into deep hibernation mode.




You may want to add test features to your code, say for use during initial setup. The **N/T: TEST** flag, can switch between normal and test mode. For example, in TEST mode, the battery monitoring display switches to Battery+, and in doing so displays the ADC value read by the micro. The values displayed are used in calibrating the battery monitoring system. Note that TEST mode also draws outline boxes in the Monitor+ display, as a reminder it is in that mode.

Rather than pressing the SW0 and SW1 button switches, to select and set modes of operation, you can do this directly from the Monitor+ app:

**Mode: n** - selects the new mode to be used.

**Task: n** - selects the new subtask to be used.

**Go!** - then directs the Omni-Bot code to run the new mode and subtask.

# 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)

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 $V_{Batt}$.
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 = 3226 On $V_{ADC}$ (2 x 4.2v)

MAX: (100%) $V_M$ = 8.2v, gave A0 = 3136 on $V_{ADC}$ (2 x 4.1v)

HIGH: (80%) $V_H$ = 7.6v, gave A0 = 2872 on $V_{ADC}$ (2 x 3.8v)

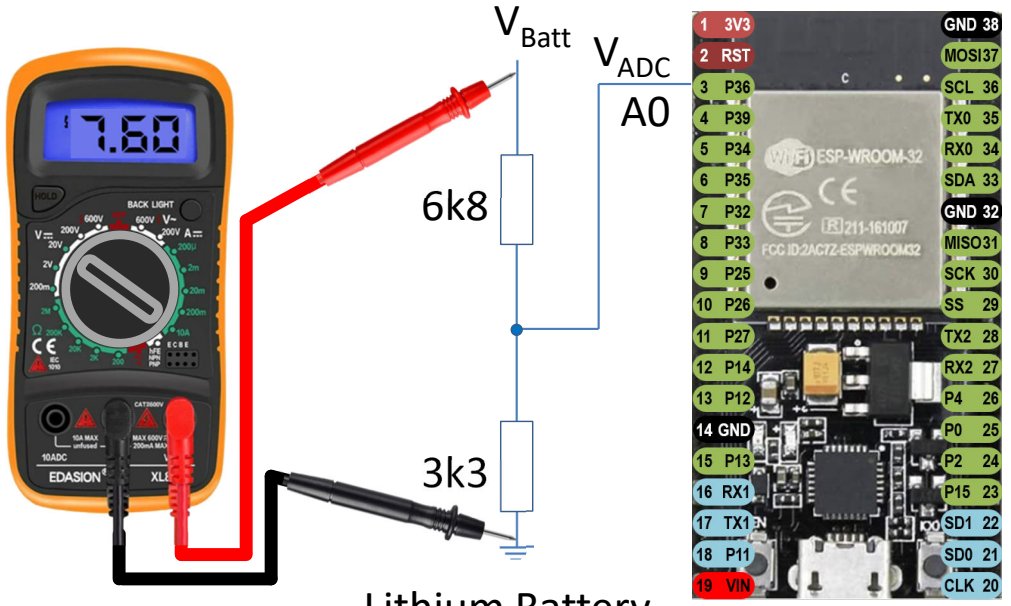
WARNING: (20%) $V_{BW}$ = 7.2v, gives A0 = 2710 on $V_{ADC}$ (2 x 3.6v)

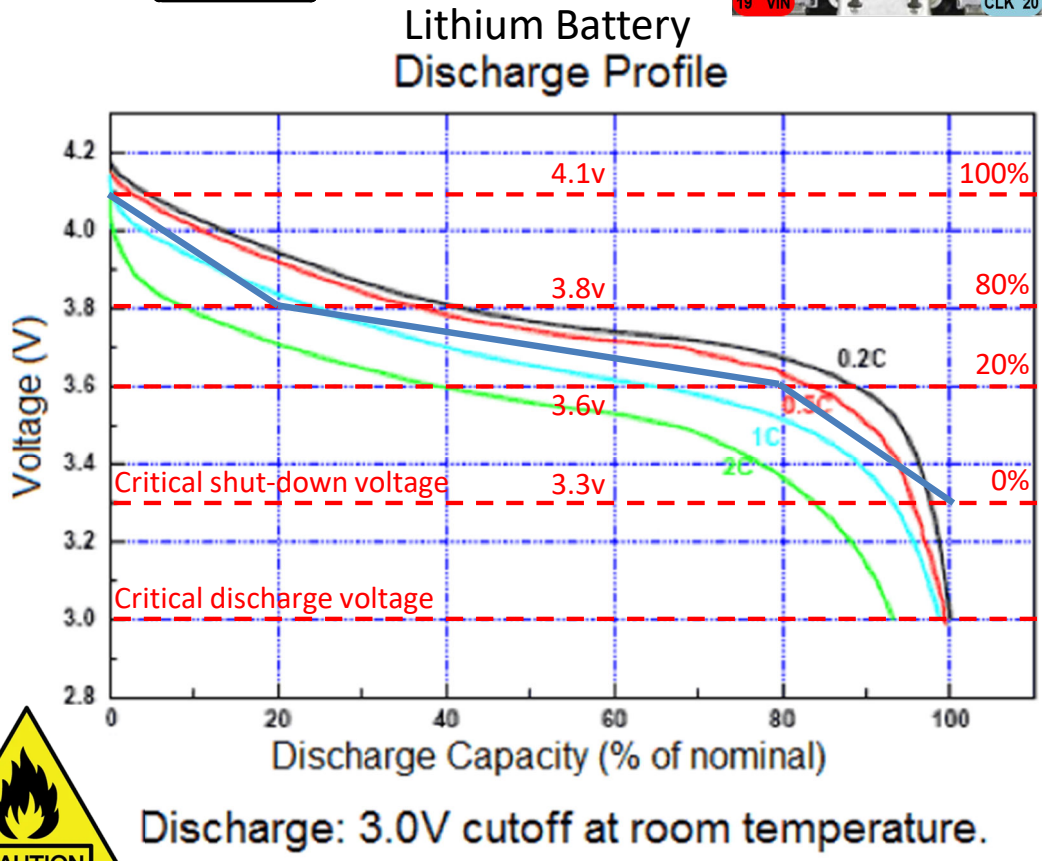
CRITICAL: (0%) $V_{BC}$ = 6.6v, gives A0 = 2460 on $V_{ADC}$ (2 x 3.3v)

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. It also detects no battery as USB mode.

In the code I have assumed a discharge curve ranging from 8.2v (100%) to 6.6v (0%) capacity, using the overlay lines shown. The rate of discharge is monitored and used to predict the life 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.



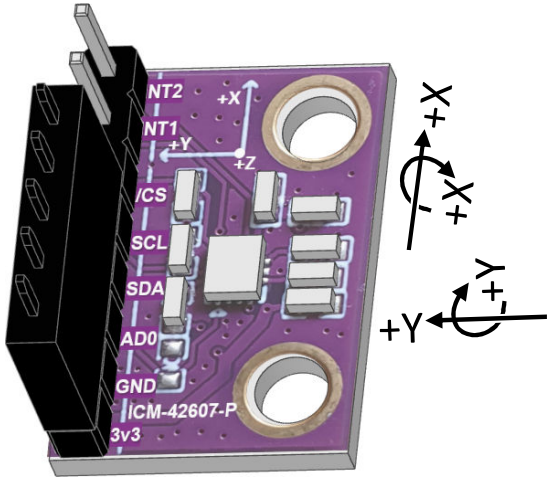


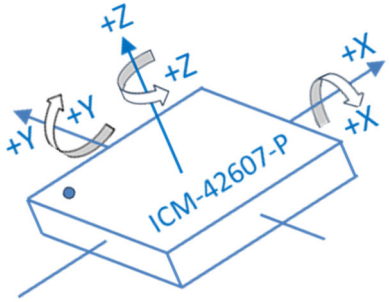


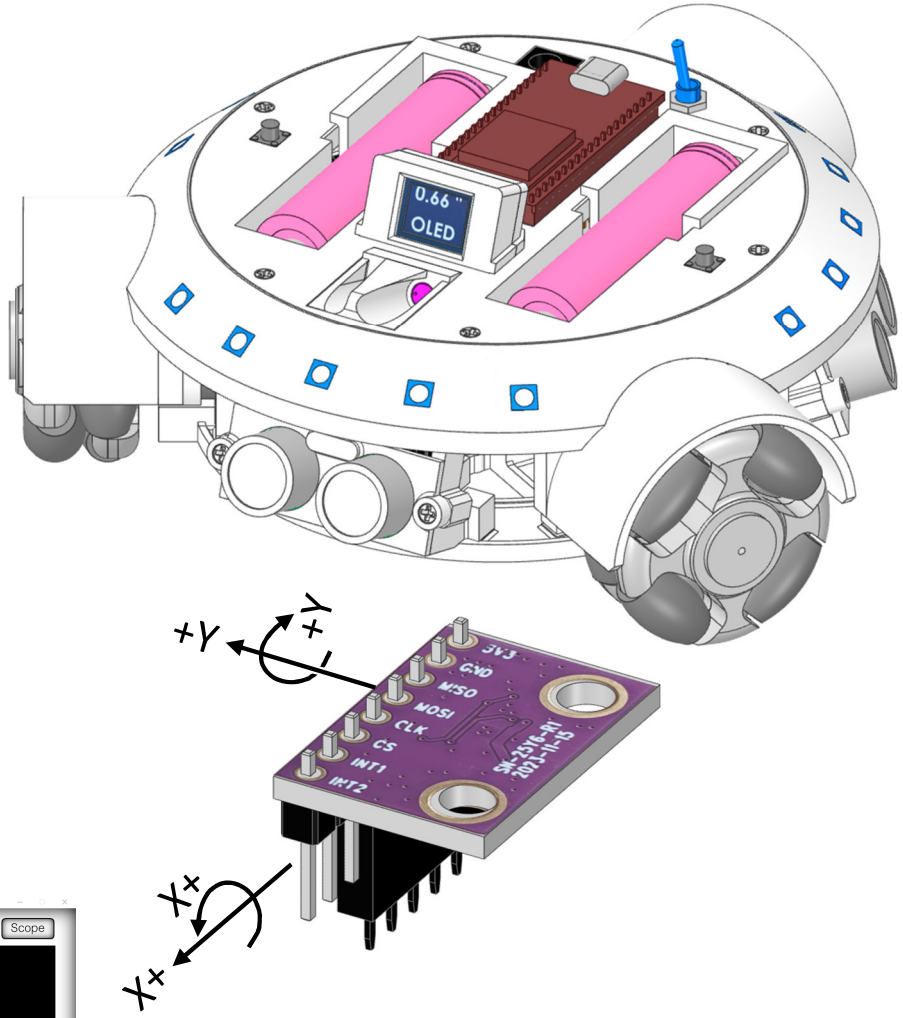
Issue: 1.1 Released: 28/11/2025 TechKnowTone

# ICM-42607-P Sensor Orientation

This diagram shows the orientation of the ICM-42607-P device within the Omni-Bot chassis, and how its internal accelerometers and gyroscopes relate to this. In this design we are only interested in the X and Y accelerometers, and the Z-axis gyro; as they allow us to calculate the speed and distance of the Omni-Bot, and its angle of rotation.

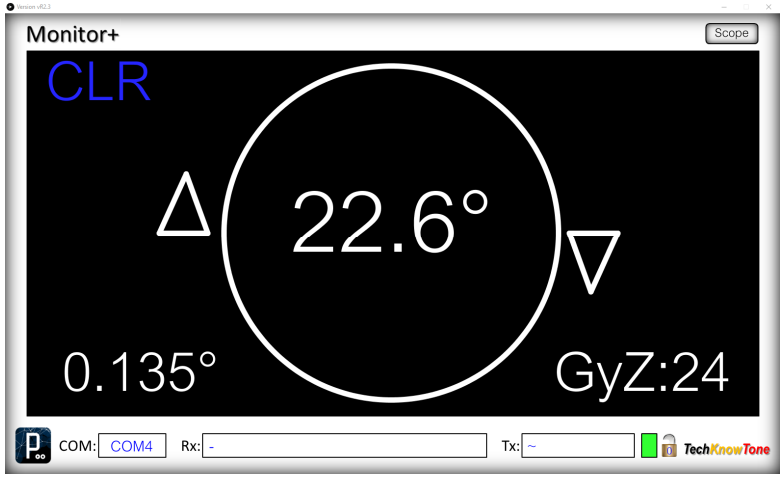



For example, the Monitor+ app can display the angle through which the Omni-Bot has been rotated.

This a relative measure, as there is no accurate way of determining an absolute figure.

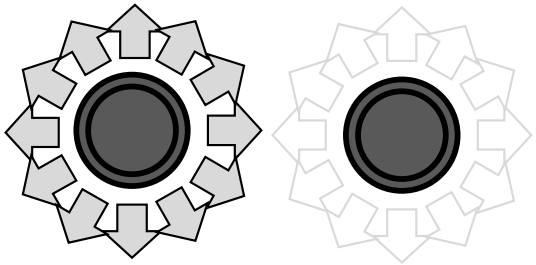



Gyroscopes are set at +/-250 °/sec FSD.
Hence at 32,767 FSD; rotation of 1 °/sec = 131.068
Change in gyro angle come from the time between readings.
On a 4 ms code loop cycle, we would accumulate a count of 32,767 over 250 cycles when rotating at 1 °/sec.
So delta angle per 4 ms cycle = gyro rate * 0.0000305

# Omni Steering

This diagram shows the wiring of the chassis motors and their associated H-bridge drivers.

The code uses the left-hand joystick values, to determine the drive characteristics of the motors, in terms of direction of rotation and relative power.

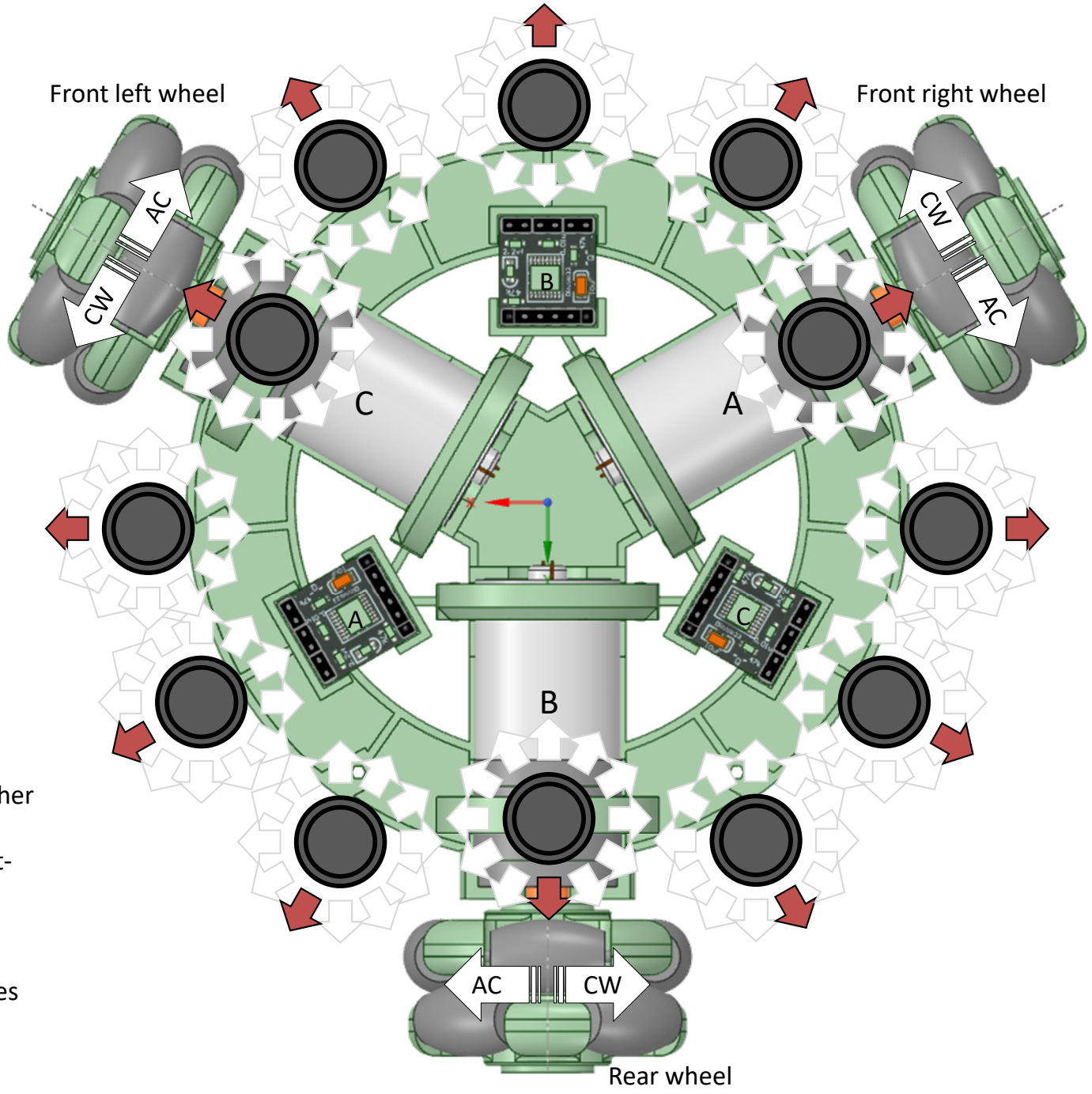


$$\text{Vector} = \sqrt{(X^2 + Y^2)}$$

$$\text{Angle} = \tan^{-1}(X/Y)$$

This then is translated into 12 directions of travel, rather than an infinitely variable system. This method was chosen, as a compromise, as the sensitivity of the left-hand joystick is only 0 – 63, centred on 32.

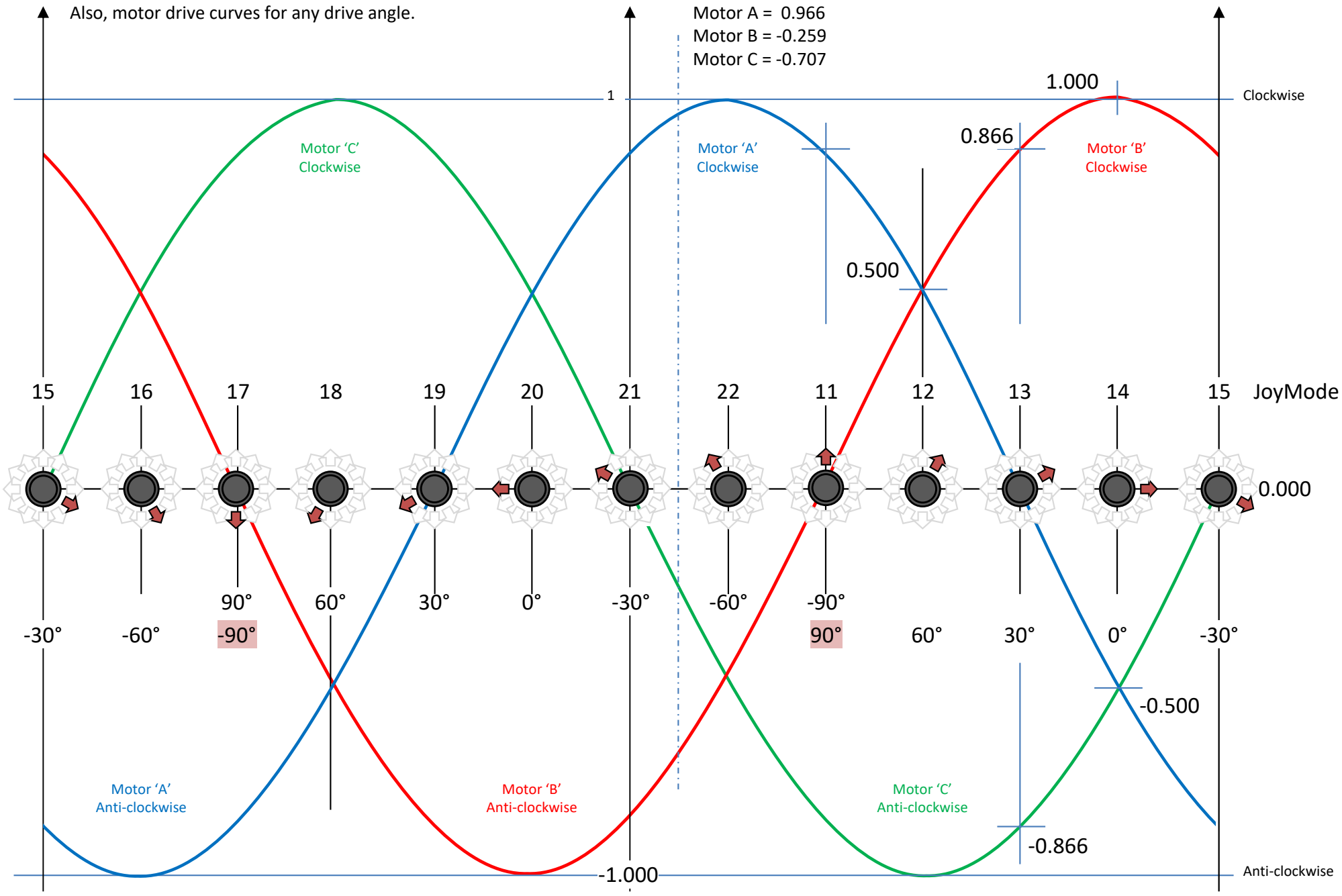
For an infinitely variable system, you can use the diagram on the following page to determine the values needed to drive the Omni-Bot at any angle.






Issue: 1.1 Released: 28/11/2025 TechKnowTone

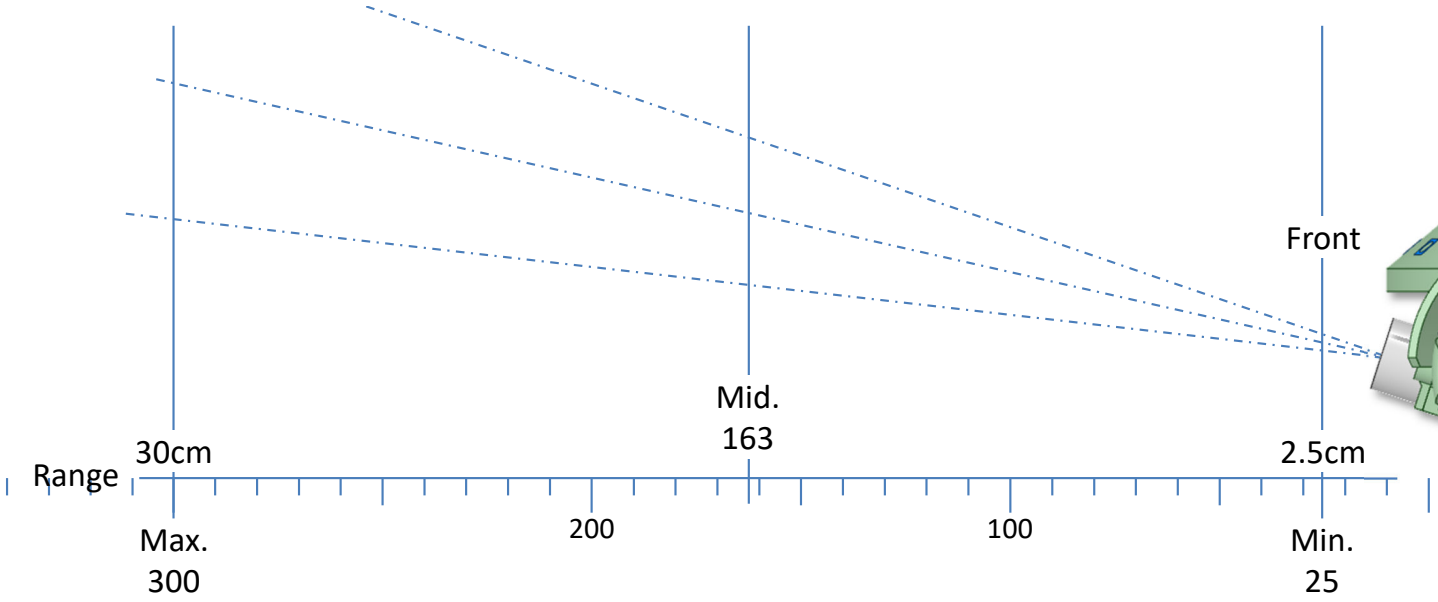
# Motor drive factors v joystick angle

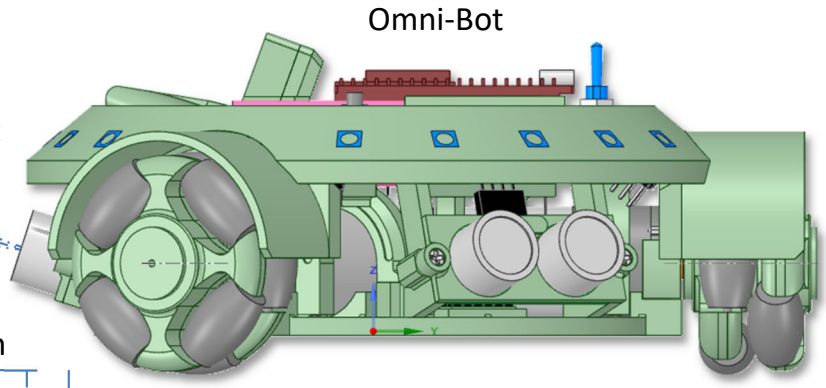





Issue: 1.1 Released: 28/11/2025

# Ultrasonic Sensor

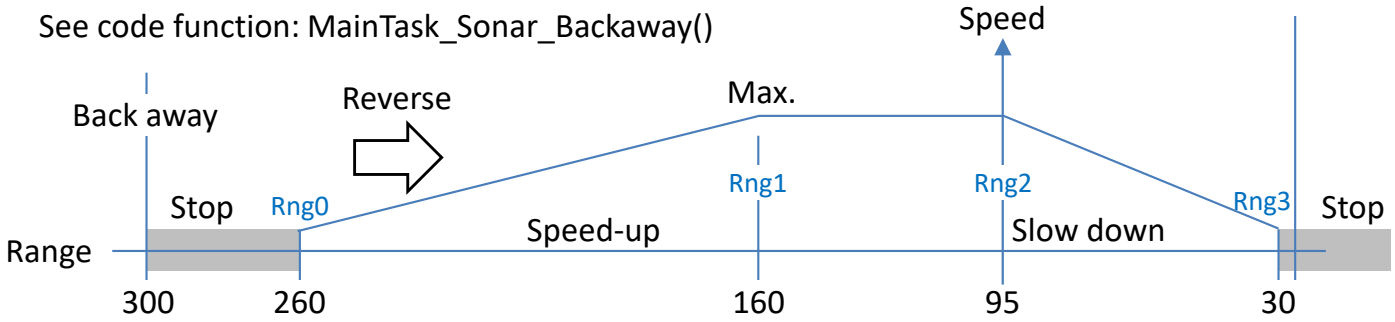


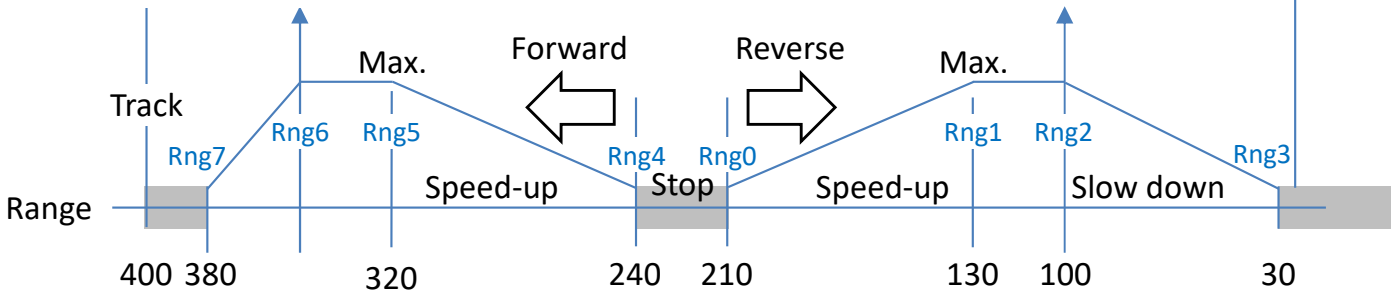

These are the values I determined from my front ultrasonic sensor, and the speed maps I developed for the back away and hand tracking functions.

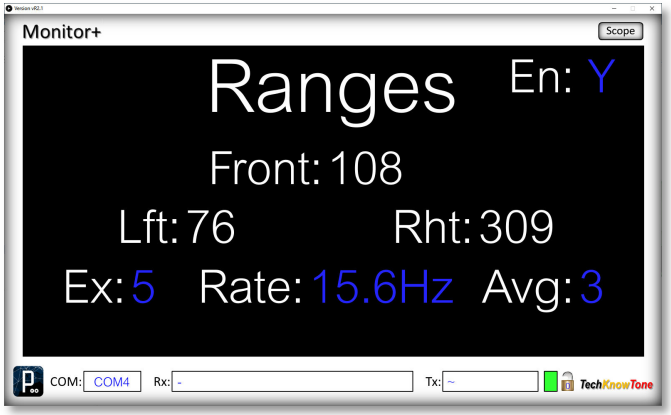
You should be able to find these values in the code, and if necessary, substitute the values you have determined from your sensor. Use the Monitor+ app to display your values.

See code function: MainTask_Sonar_Backaway()




Track has not been coded






Issue: 1.1 Released: 28/11/2025 TechKnowTone