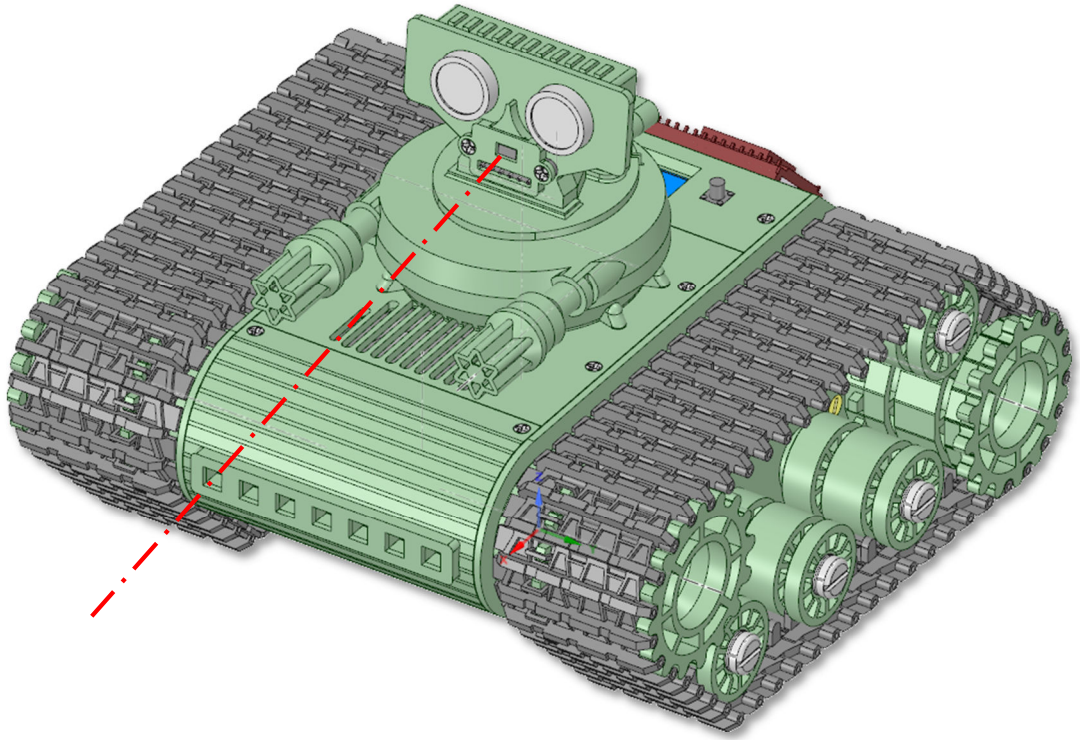
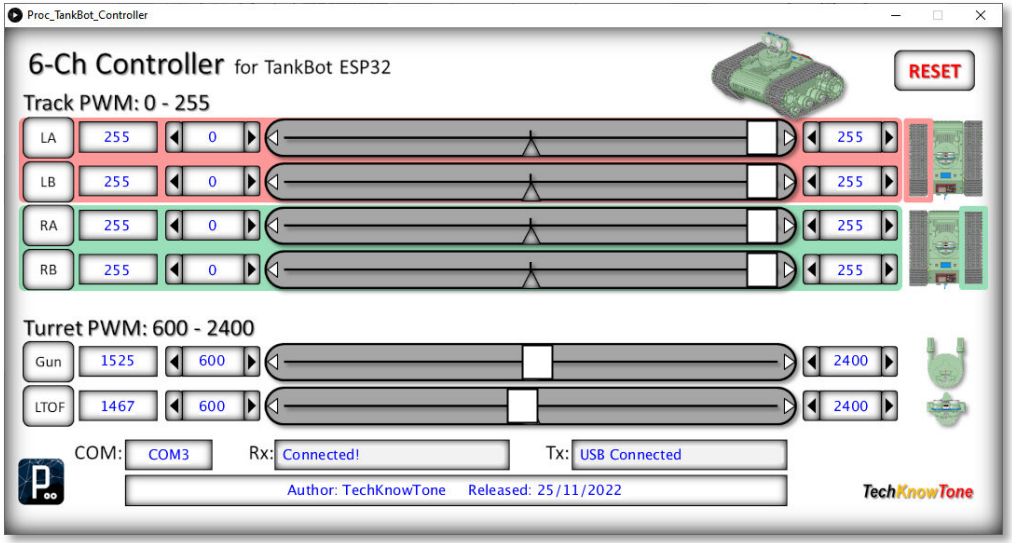
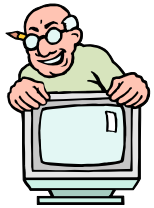
TankBot MK1 (ESP32)

Calibration

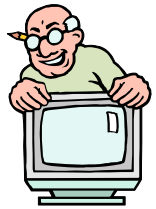






Servo calibration, is an essential process!

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.

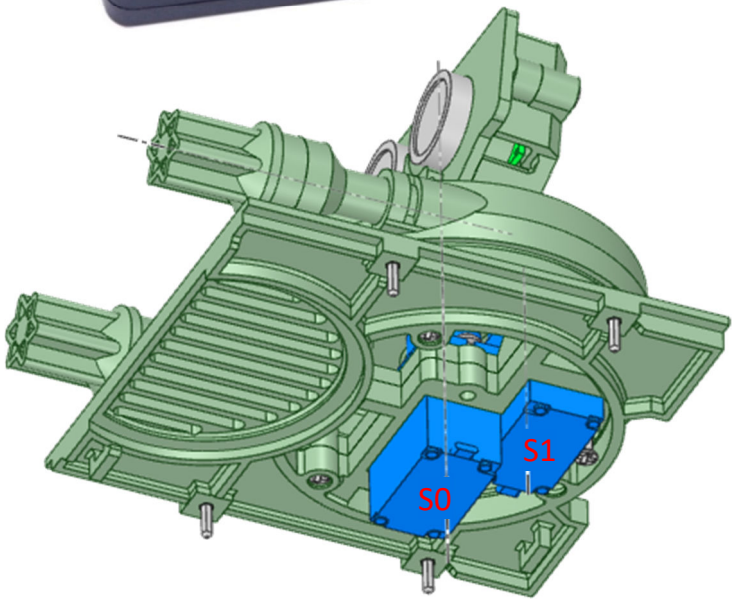
Overview

Servo calibration is a two step process; Course settings are used in the assembly process, when attaching servo leaver arms, and fine tuning is covered in this document. A Servo Consistency Tester is used for course settings, and fine tuning uses a custom Windows app, written specifically for this project by me.

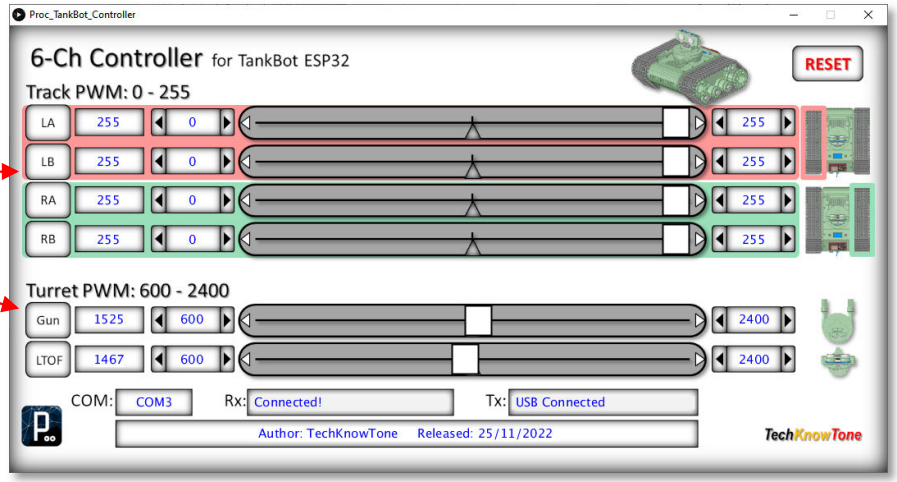
Course: The servo tester is used to set the 1500 µs centre positions, when fitting the leaver arms in the robots turret. This is to ensure that the servo will have sufficient movement range in both directions when mounted in your robot.

The PWM centre position of a servo is 1,500 µs.

It is simply a matter of powering up the servo tester, plugging in the servo and setting the testers output to 1500. Then fitting the servos cross shaped leaver onto the servos splined drive shaft in the correct position shown here. The splined drive shaft teeth positions may prevent you from getting it exactly right, but the nearest position possible compromise will do.



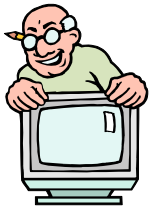
Fine: Once the turret servos are installed in the robot, and the wiring of the chassis is complete, we can then use the 6-channel app to adjust the servo angles, to determine limits to be set in code for the gun and sensor servos.



The channels on this controller app correspond to:
LA,LB,RA,RB - track DC motors PWM, ranging from 0 – 255.
Gun, LTOF - gun & sensor servos, ranging from 600 – 2400 µs.

The robot should be placed on its stand, code set to TESTmode true, powered up and connected to your PC using a USB lead. You need to turn ON a channel, by clicking on the channel button, before the app will send PWM values, and respond to the slider values. Turning the channel OFF will remove the PWM signal,

Gun Servo S0



The S0 servo is attached to the turret in the centre, and swings the guns left and right. Since the sensor servo passes through the turret roof, it is not affected by turret movement, however the aperture for this does limit the rotation of the turret to +/- 62°.

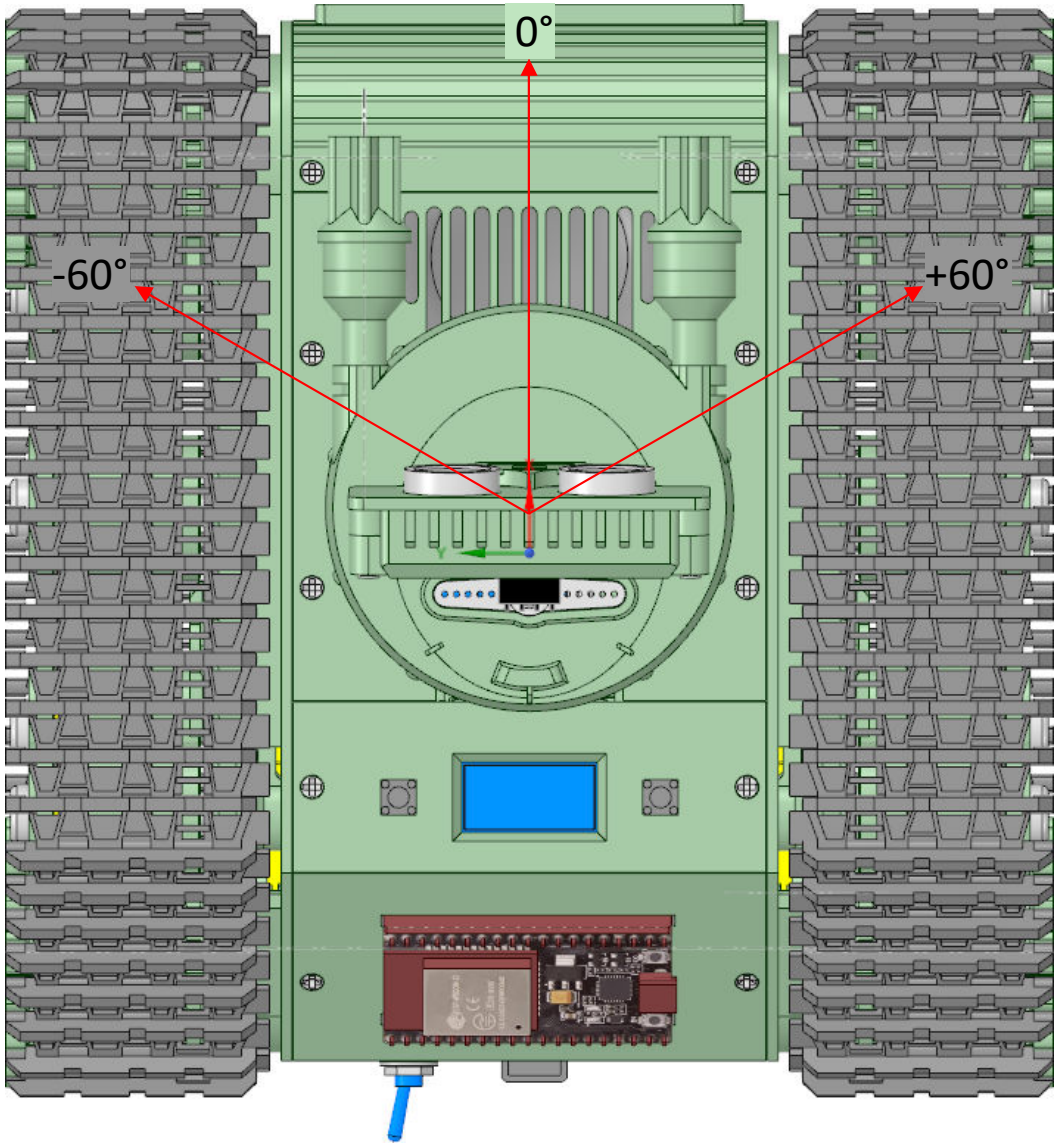
The rotating turret has a reference slot, which can be used as an alignment aid, when lined up with markers on the lower structure. Using the 6-channel controller app provided, select the Gun slider and determine the PWM value of the middle centre point. During assembly this was pre-set at 1500µs, so it should be close to that value.

Then cautiously determine the PWM values for -60° and +60° angles, remembering that there is a mechanical limit of +/-62° which we wish to avoid. If we run into the mechanical limit the servo motor is stalled and will draw a large current, and could burn out if left in this condition for long.

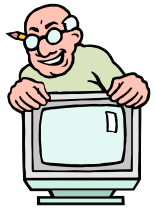
In the code we will use these PWM values as reference limits, like this:

Angle	Codename	PWM
0° centre	Gun0	1521
-60°	Gun60N	2090
+60°	Gun60P	879

Once you have determined the values, add them to your code.



Sensor Servo S1



The S1 servo is mounted inside of the turret, behind S0, and swings the sensors left and right. Whilst it does not have the mechanical limitations of S0, we want it to have the same range of movement, limiting its rotation to +/- 60°.

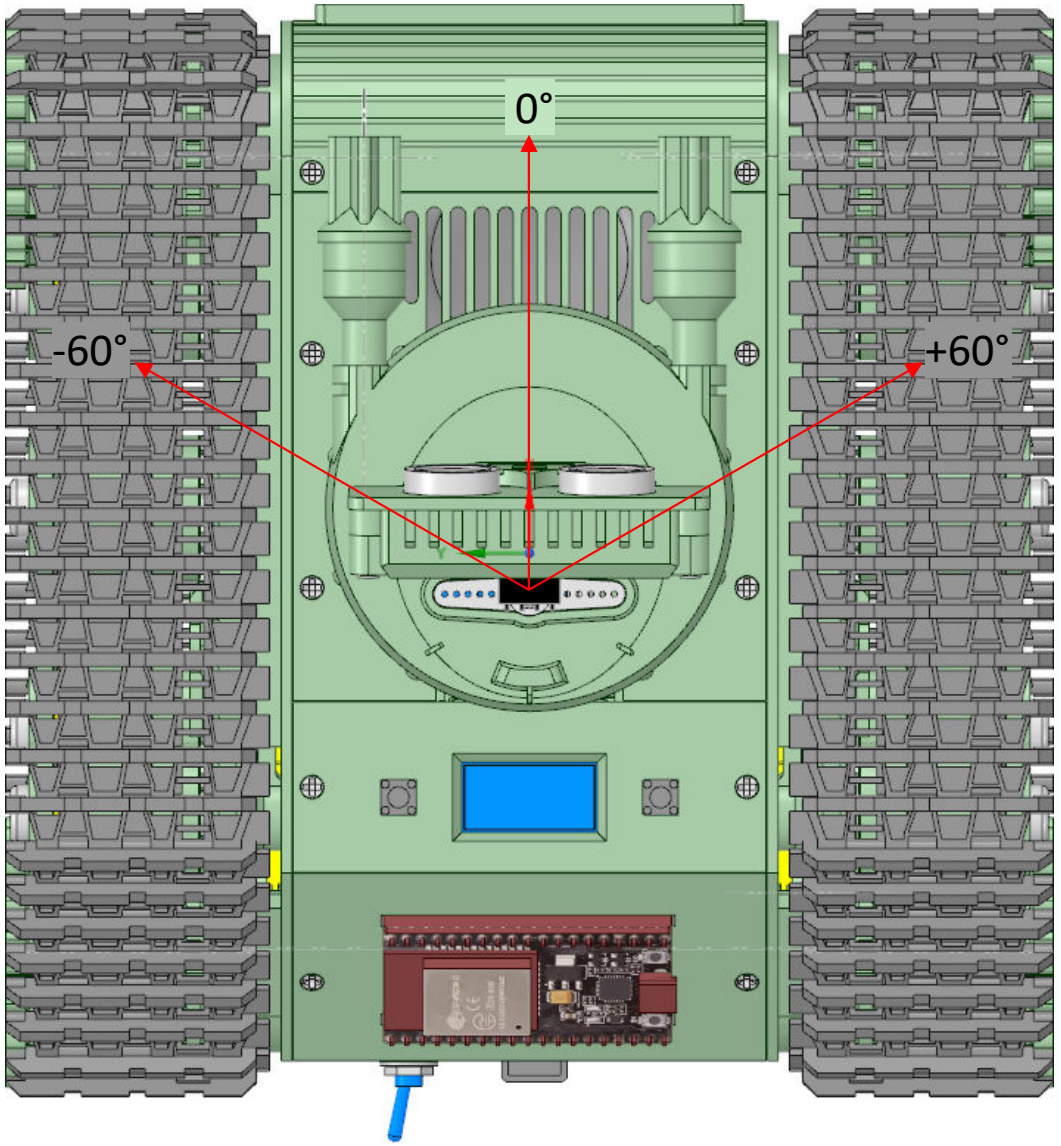
The sensor mounting plate has reference marks which align with slots in the turret roof for reference. Using the 6-channel controller app provided, select the LTOF slider and determine the PWM value of the middle centre point. During assembly this was pre-set at 1500µs, so it should be close to that value.

Then determine the PWM values for -60° and +60° in the same was as you did for the gun.

In the code we will use these PWM values as reference limits, like this:

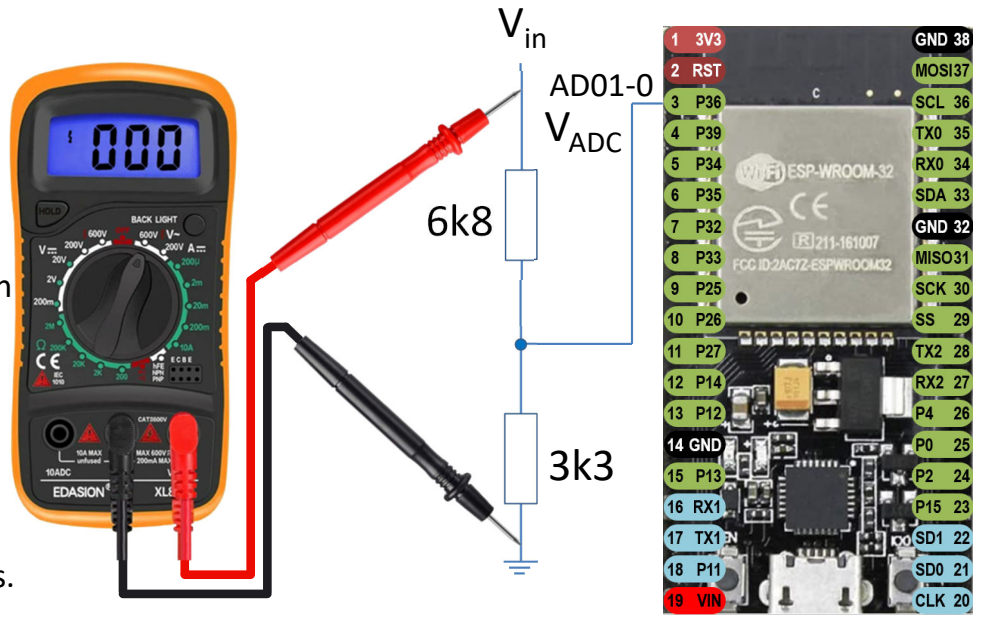
Angle	Codename	PWM
0° centre	Sen0	1438
-60°	Sen60N	2098
+60°	Sen60P	817

Once you have determined the values, include them to your code.



Battery Voltage Health Monitoring

See 18650 discharge curve obtained from the internet below. In this robot both batteries are assumed to be identical, and connected in series. With the robot powered, from either its batteries or external source, and in sleep mode, press SW0 to get the robot to repeatedly display battery readings. Measure the V_{in} voltage with a multimeter; note that and the average V_{ADC} value displayed left on the OLED at that time. I had 7.68 volts for V_{ADC} = 2810.



We determine a constant: BatCal = 2810/7.68 = 365.9 conversion factor.
This conversion constant will be entered in our code for display purposes.

Now that we have the value of BatCal, we can determine V_{ADC} values for each of our limits in the code as follows. Remember that we have two batteries in series.

BatMax 100% is when V_{in} = 8.2v converted gives V_{ADC} = 3000

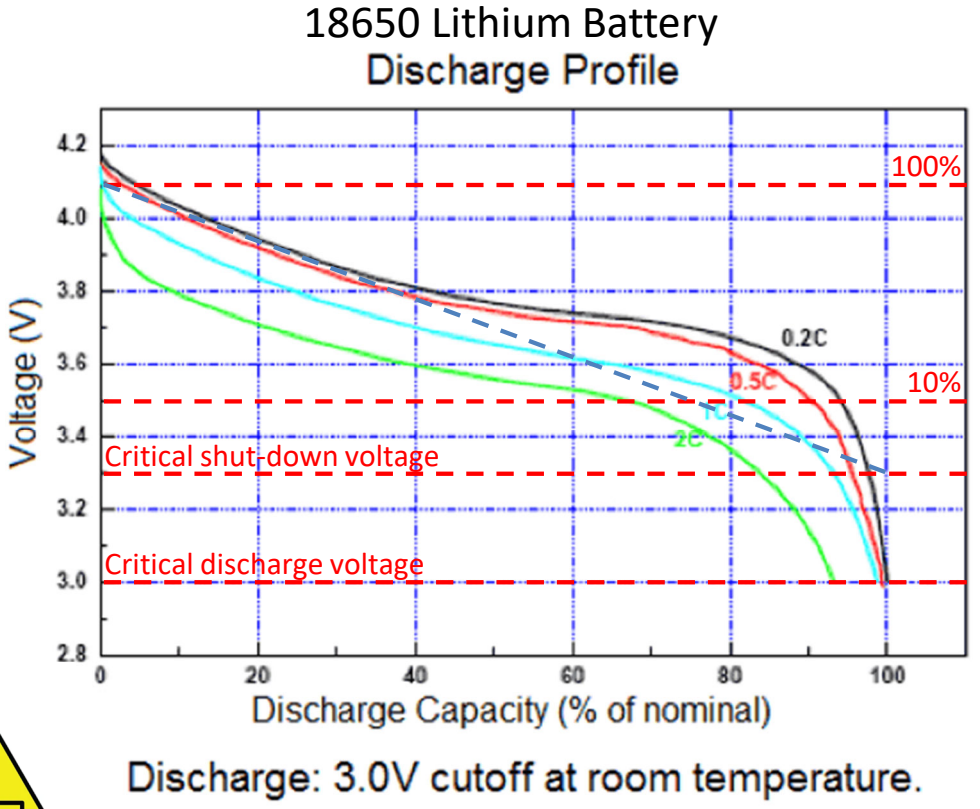
BatWarn 10% warning point at V_{in} = 7.00v converted gives V_{ADC} = 2561

BatCritical 0% cut-off point at V_{in} = 6.60v converted gives V_{ADC} = 2415

Enter these #define values into your code, and recompile it. You now have a built in calibrated digital voltmeter, and useful warning limits

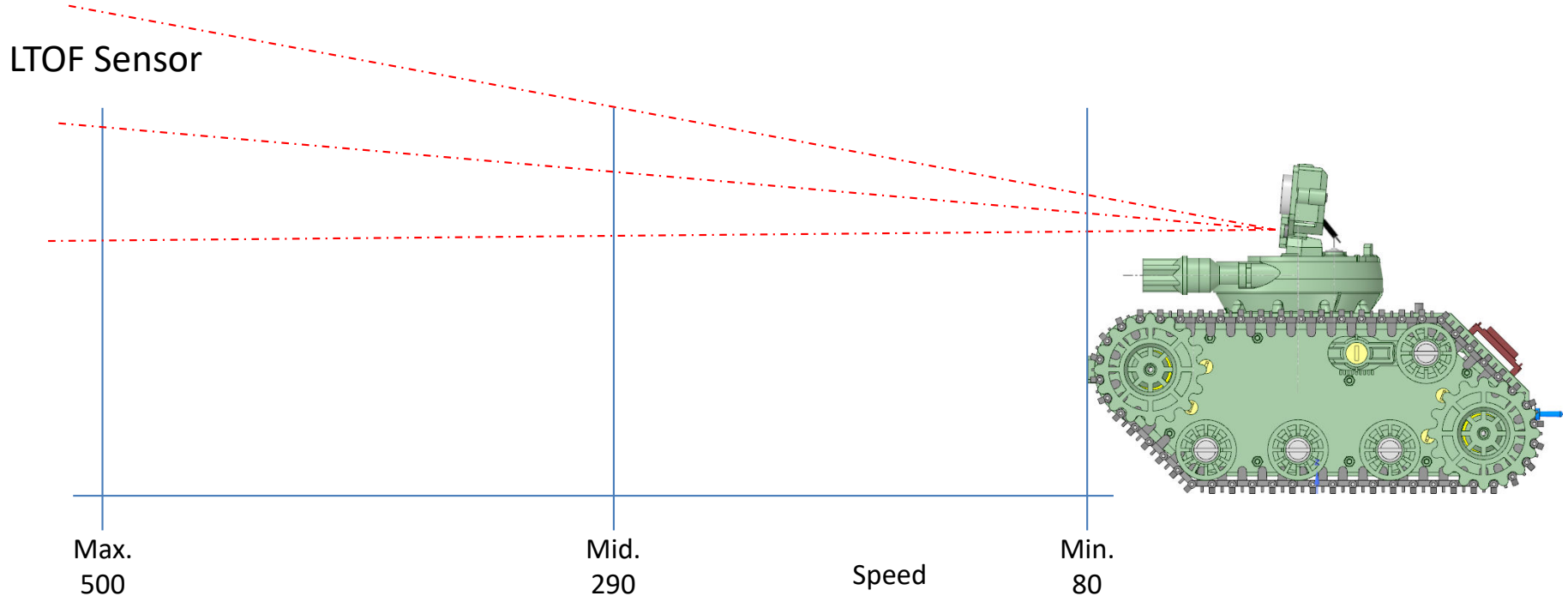
Notes:
The code will sample the battery voltage on power-up to ensure it is sufficient, then at regular interval, calculating an average to remove noise. Given the relatively light current drawn I have assumed a linear discharge curve ranging from 8.2v (100%) to 6.6v (0%) capacity. The rate of discharge is monitored and used to actively predict and display the life of the battery in use.

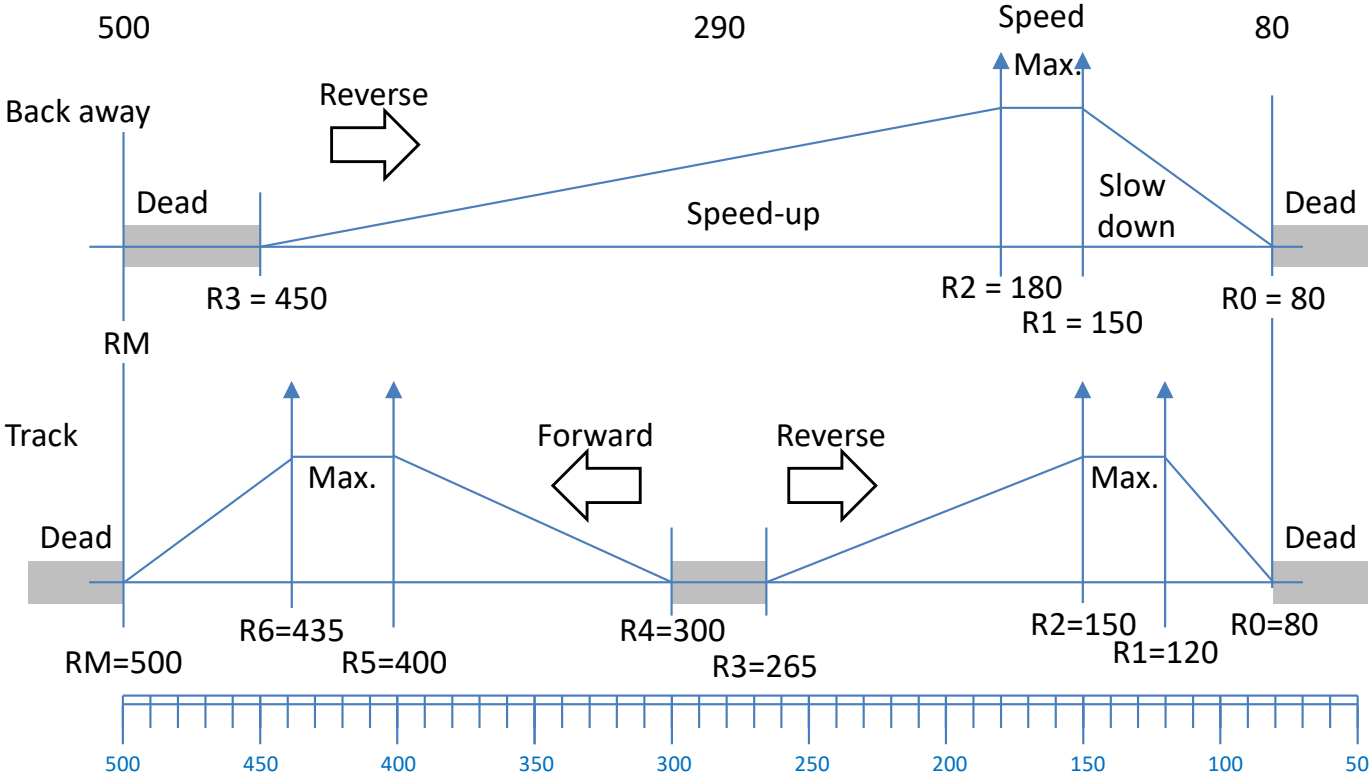
Note: If connected to USB port with internal battery switched OFF the ADC will read a value 5 volts (A0 = 1919) or less. So if the micro starts with such a low reading it knows that it is on USB power.



Issue: 1.1 Released: 08/12/2025 TechKnowTone

LTOF Sensor

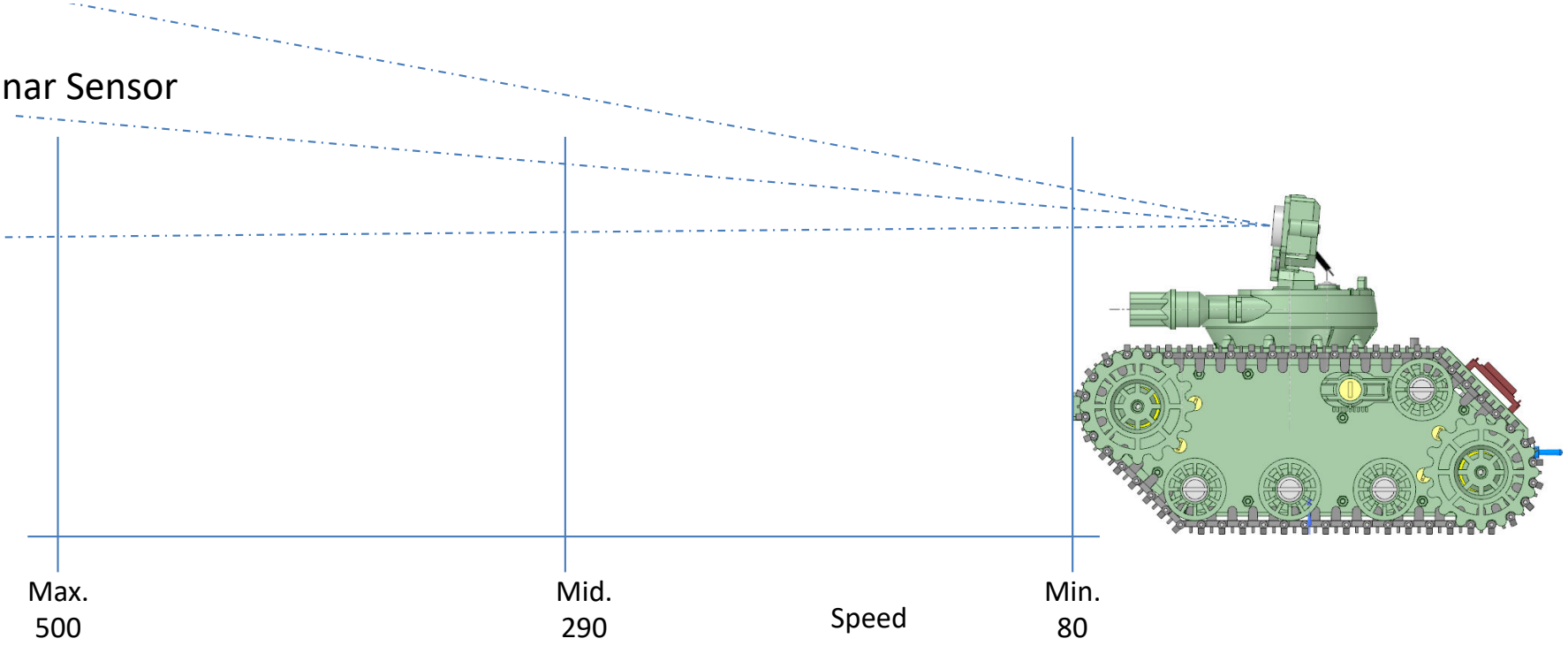


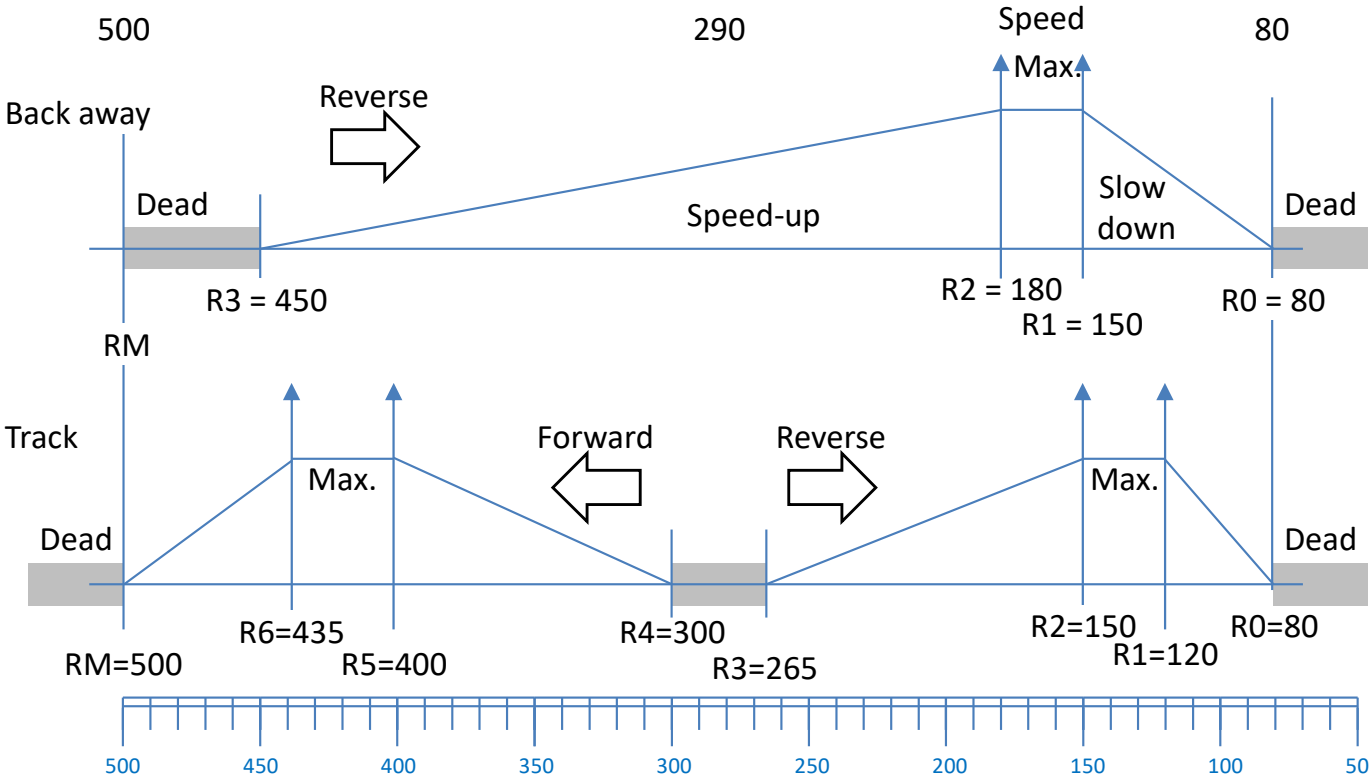


These are the values I determined from my VL53L1X sensor, and the speed maps I developed for the back away and wall tracking functions.

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 serial monitor in the IDE along with Serial.print() functions to display your values.

Sonar Sensor





These are the values I determined from my ultrasonic sensor, and the speed maps I developed for the back away and wall 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 serial monitor in the IDE along with Serial.print() functions to display your values.