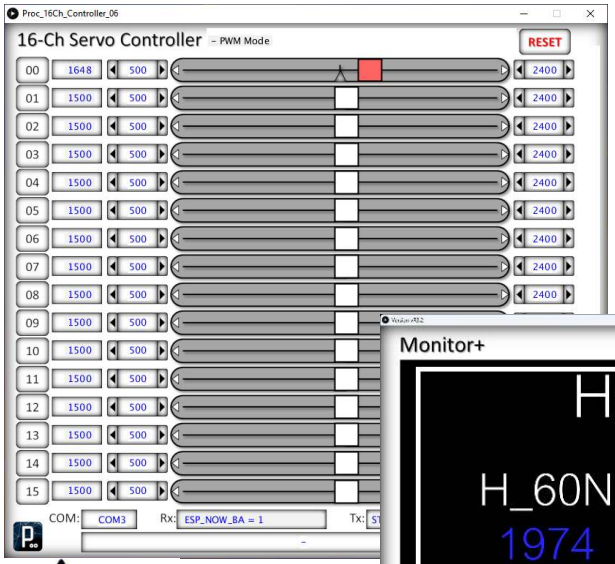
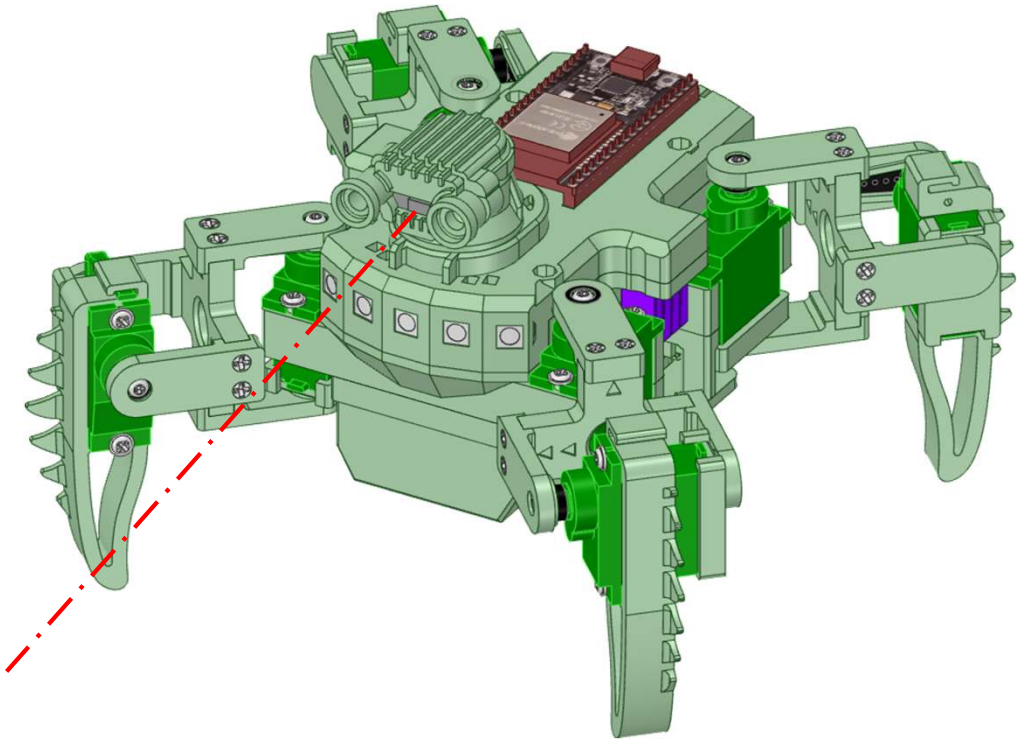
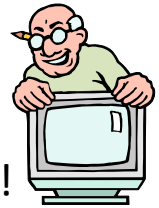
Autonomous Quadruped Robot ESP32 MkII

Calibration

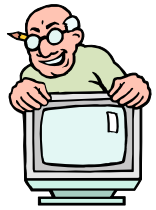






Servo calibration is an essential process!

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.

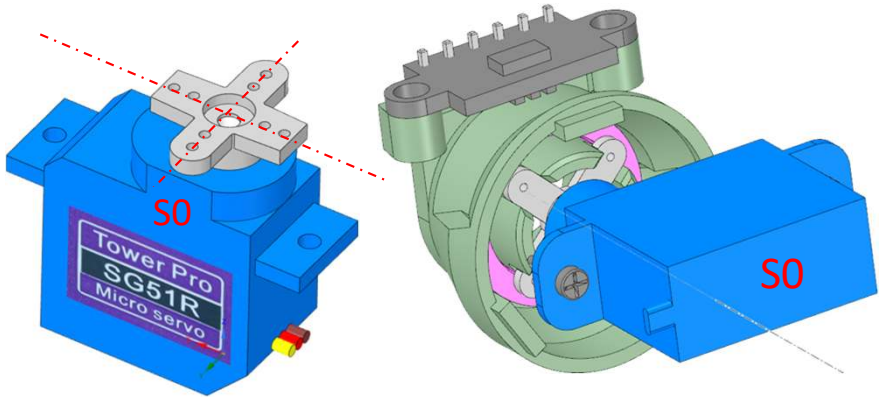
Overview

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 on of two custom Windows app, written specifically for this project by me.

Servo Tester

Course: The servo tester is used to set the centre position, when fitting the cross shaped leaver in the robots head. This is to ensure that the servo will have sufficient movement range in both directions when mounted in your robot.

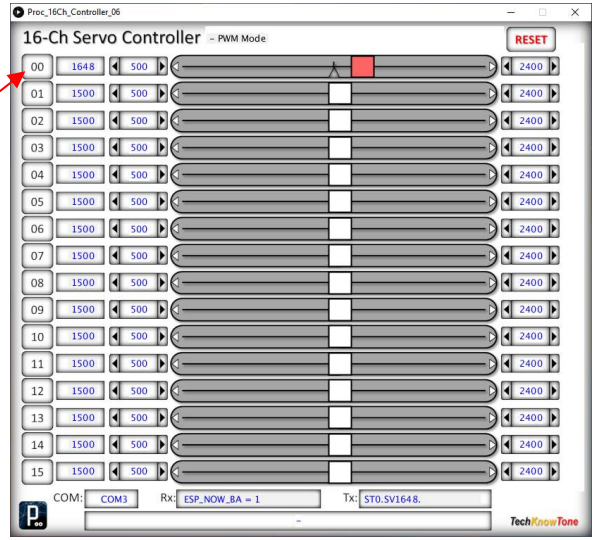
The PWM centre position of a servo is 1,500 µs (1.5ms).



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 will do. This cross shaped leaver mates with recesses in the neck, to drive the head in a pan motion, for the laser range finder. Note the cropped ends.

Fine: Once the head servo is installed in the robot, and the wiring of the chassis is complete, we can then use the 16-channel app to adjust the servo angles, to determine limits to be set in code, and also to set the leg servos.

16-channel app



The channels on this controller app correspond to:
00 - head servo angle in microseconds
01 – 08 - leg servos angles in microseconds (see numbered diagrams)

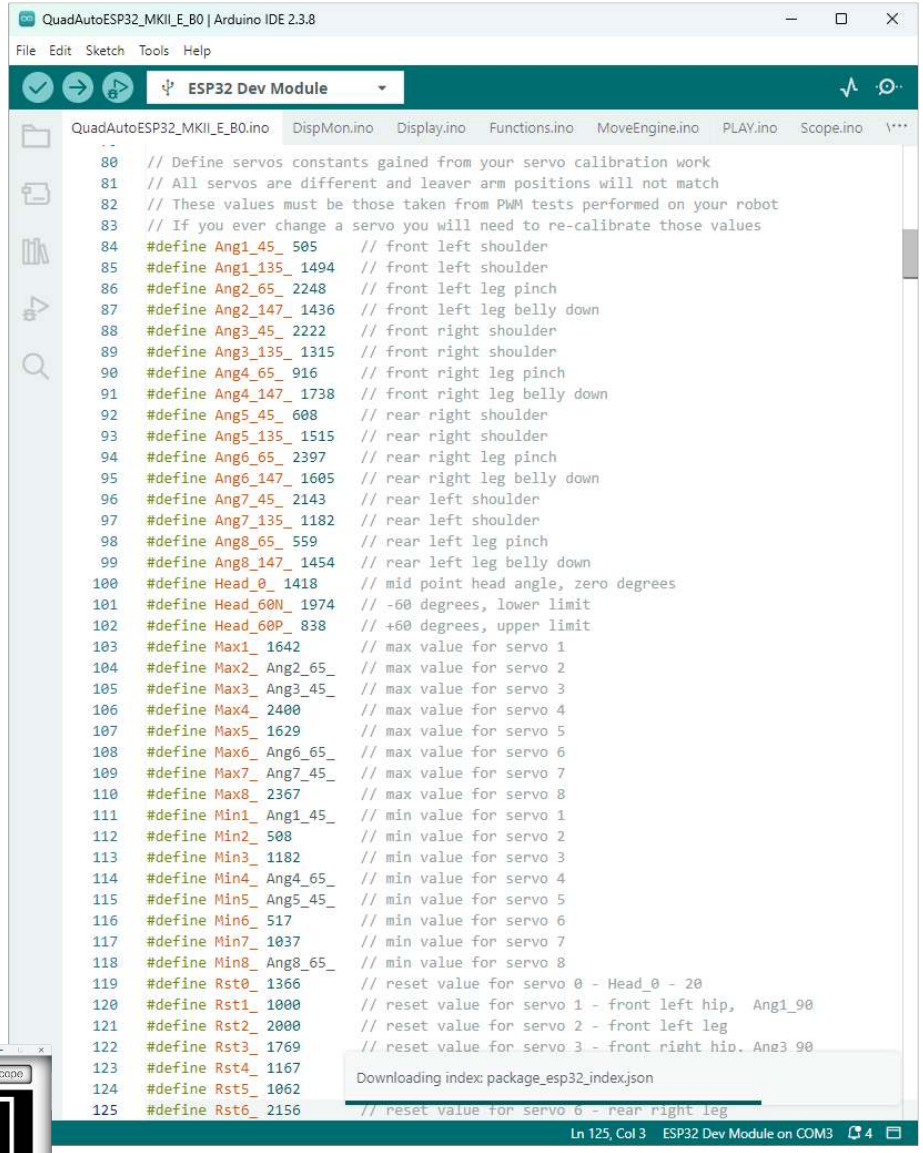
The robot should be placed on its stand, code set to TESTmode true, powered up and connected to your PC using a USB lead. You need to turn ON a channel, by clicking on the channel button, before it will receive PWM signals and respond to the slider values. Turning the channel OFF will remove the PWM signal. Alternatively you can use the Monitor+ app, see over.

EEPROM Memory

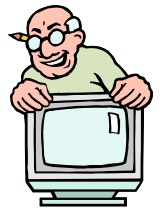
In earlier releases of the robots code, the servo calibration values were entered as fixed constants, using a table of definitions at the beginning of the code. This meant that if more that one robot was built, each had to have a unique build of code to match the servos used.

To overcome this limitation, and have only one version of code, these values are now stored in the ESP32's non-volatile EEPROM memory. You can still use the earlier method of typing in the values, or alternatively you can use the Monitor+ app, and custom screens, which allow you to adjust servo values directly, and save them to the EEPROM memory; where they will remain, unless they are over-written at a later date.

The calibration process described in this document remains the same, and you can use either Windows app to achieve the end result. The use of Monitor+ is described later in this document.



The code supplied, still contains the calibration values I determined in my project. You will need to determine values of your own.



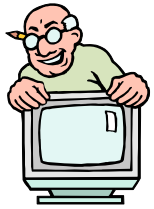
DO NOT walk your robot with my values, as you could damage it. Calibrate it first!

Issue: 1.2 Released: 18/03/2026 TechKnowTone

Head servo limits

Setting the head servo will get you familiar with using the Windows 16-channel app. The PWM numbers shown here, are from my robot. Yours will be different.

Code default values are shown in Green, and reference names are shown here in Blue. My values are in black.



Head_0

1418

1500

0°

These PWM limit values enable the robot code to calculate angles, and prevent the head drive mechanism colliding with mechanical stops.

Notice the small notches around the head scalp model, and the reference mark on the shroud to aid calibration.

These are set at 60 intervals.

Head_60N

1974

1900

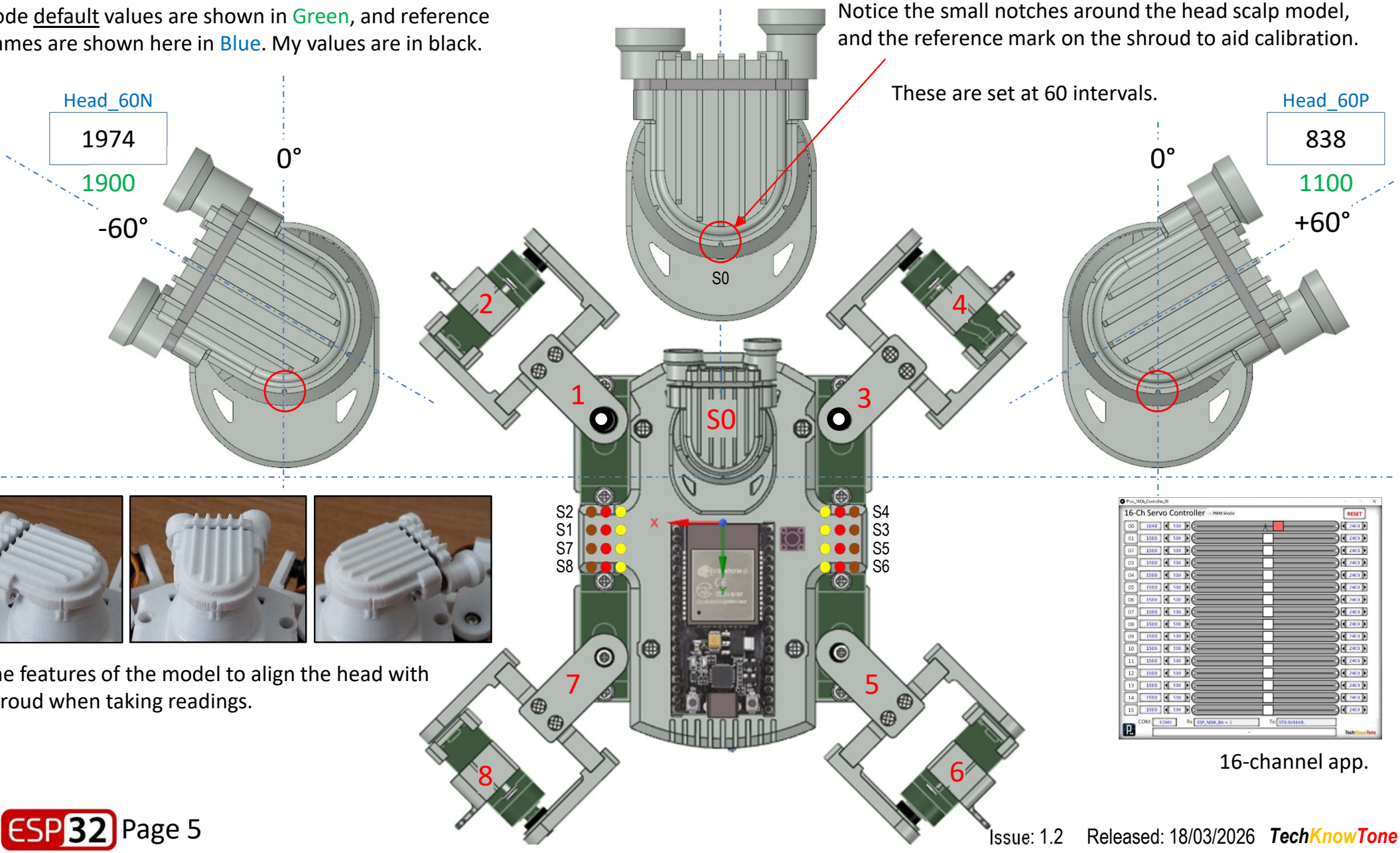
-60°

Head_60P

838

1100

+60°



Use the features of the model to align the head with the shroud when taking readings.

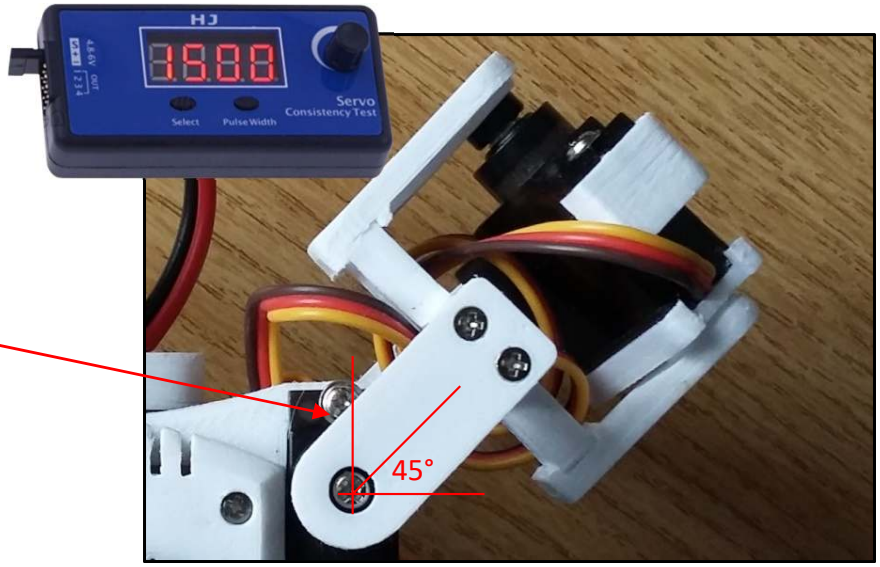
16-channel app.

Issue: 1.2 Released: 18/03/2026 TechKnowTone

Leg servo limits

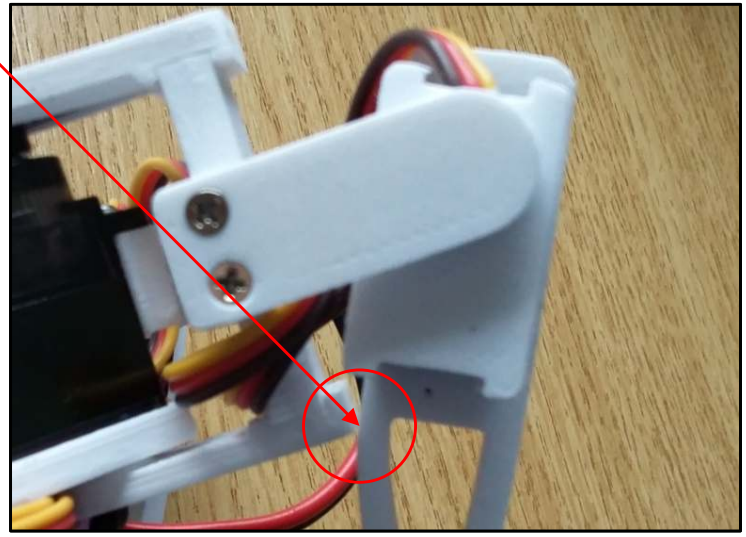
Just like the head servo, we use a servo tester when attaching the leavers to servos, before attaching the servos to your robot initially, to set the angles of the attached levers in their approximate positions. Leg servos can collide with each other and with the robot body, so fine settings stored in code prevent this.

Course: Leg hip servo leaver arms are set with the servo mounted in the robots body, at approximately 45° which corresponds to a PWM value of 1500µs. The centre of the hip servos mounting screw gives you an indication of this angle.



Fit the hip joints at 1500µs == 45°

The leg pivot servos have their arms fitted at a point where they touch the lower part of the cross mounting, with a servo value of either 800 or 2200µs, whichever is the appropriate value for that servo.

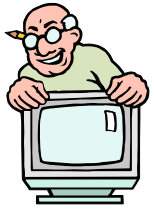


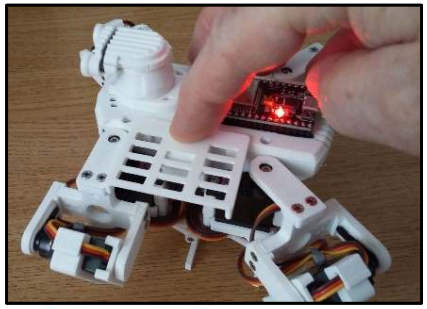
Fit legs just touching their collision points.

Fine: Once you have assembled and wired up your robot you can use the 16-channel application I have provided. You can select each servo in turn and move their sliders to determine values needed to achieve the calibration angles shown in the subsequent pages of this guide. Take care to avoid collisions, as this causes your servo motors to over-heat, and could permanently damage them.

Note that as no two servos are alike, the values I have given in this guide, and included in the sample code, will be similar but different from the ones you derive from your robot. That's the nature of servos!

Performing these tasks with patience and accuracy will lead to good robot performance and longer battery life.



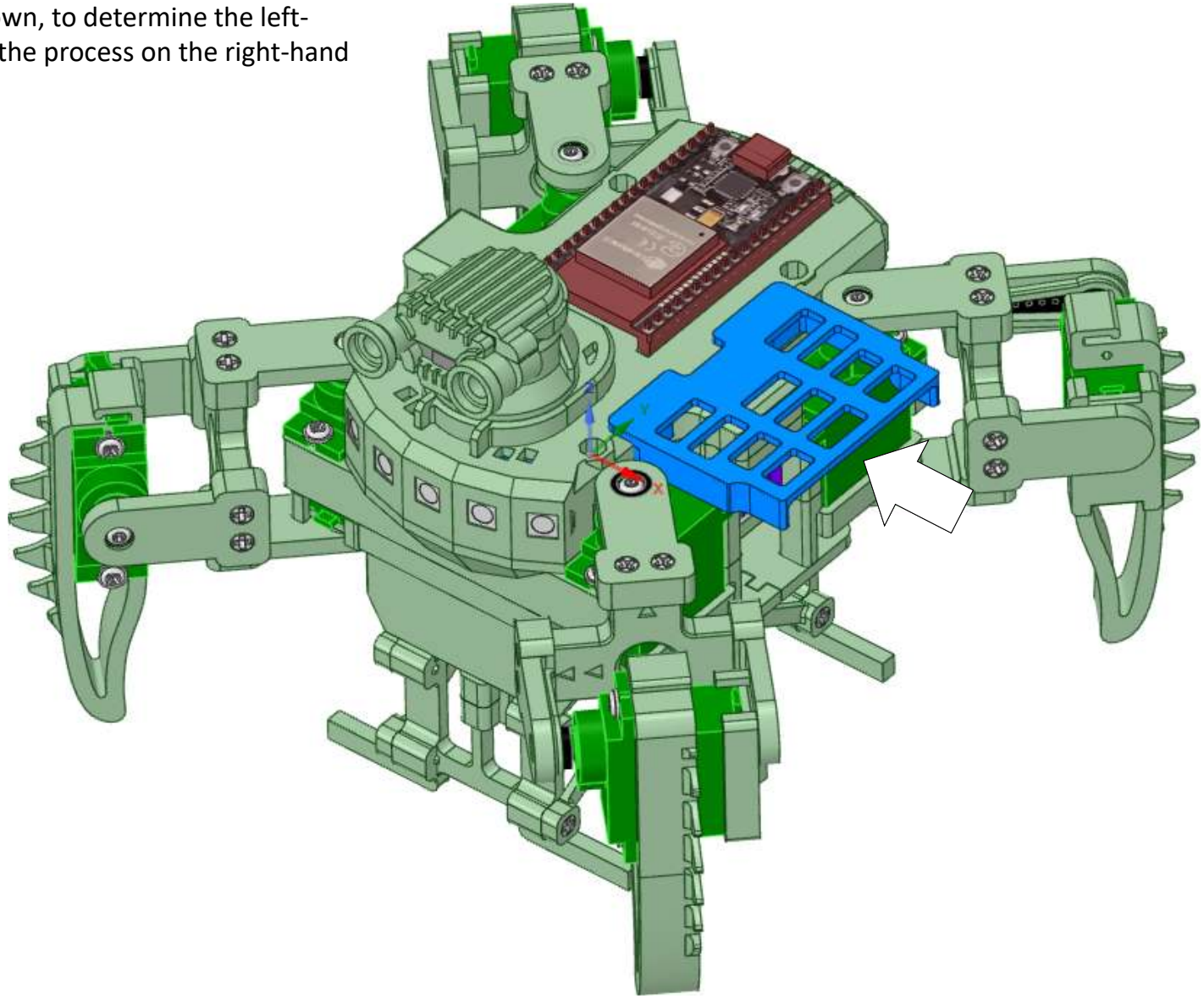


There is an angle gauge included in the model files which will help you determine the hip 135 angles. Simply hold it against the robot, whilst adjusting the appropriate slider in the app.

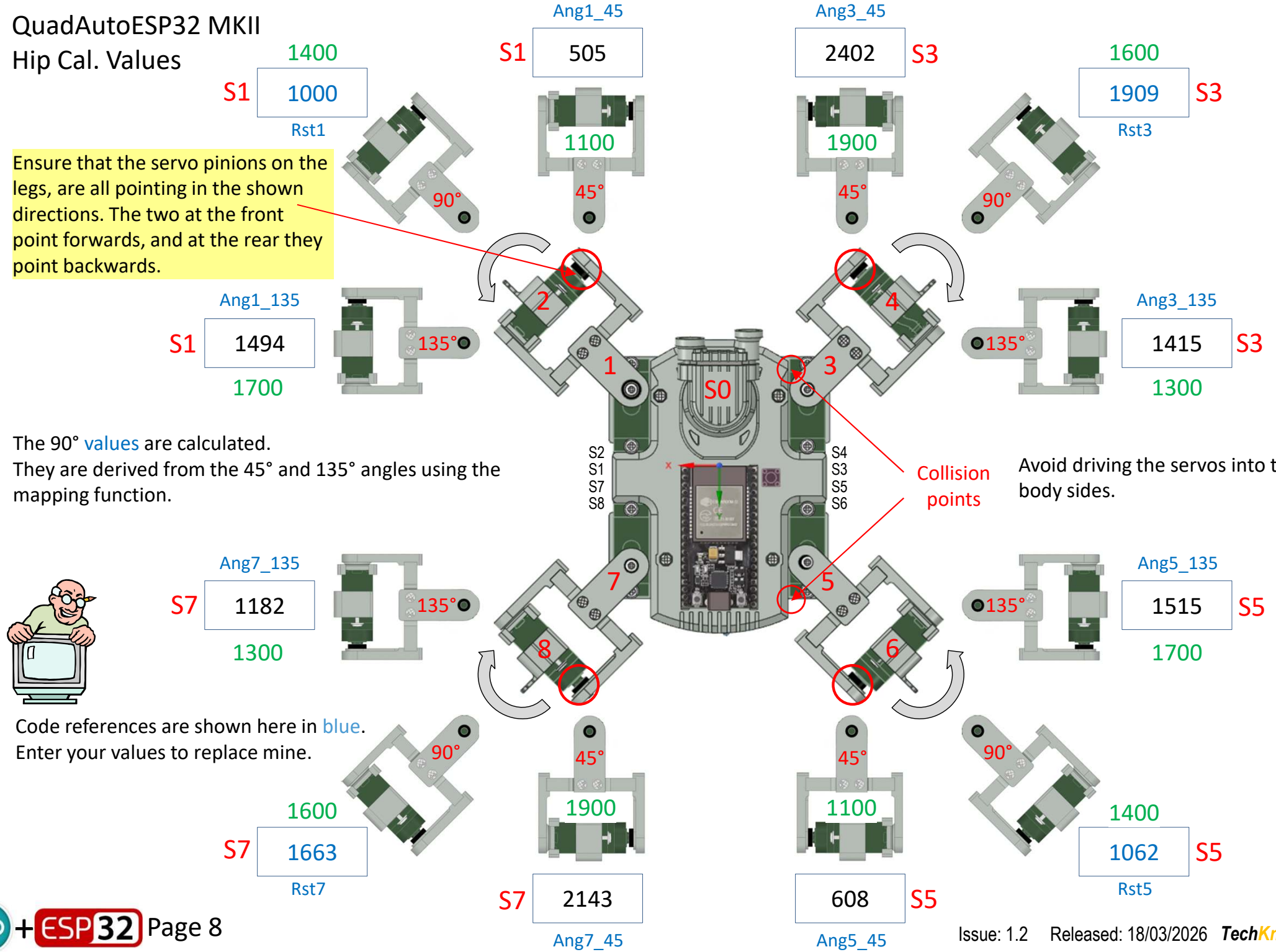
Leg servo Ang_135 gauge

Due to the components mounted on the top surface of the robot, it is not easy to determine the 90° angles for the leg servos; otherwise referred to as Ang_135 values.

You can use the Z-Angle Gauge component to aid this process. Hold the gauge against the side of the robot as shown, to determine the left-hand servo Ang_135 values. Then repeat the process on the right-hand side.



QuadAutoESP32 MKII
Hip Cal. Values



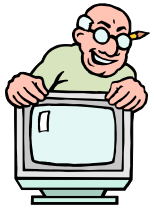
Issue: 1.2 Released: 18/03/2026 TechKnowTone

QuadAutoESP32 MKII
Hip Cal. Values

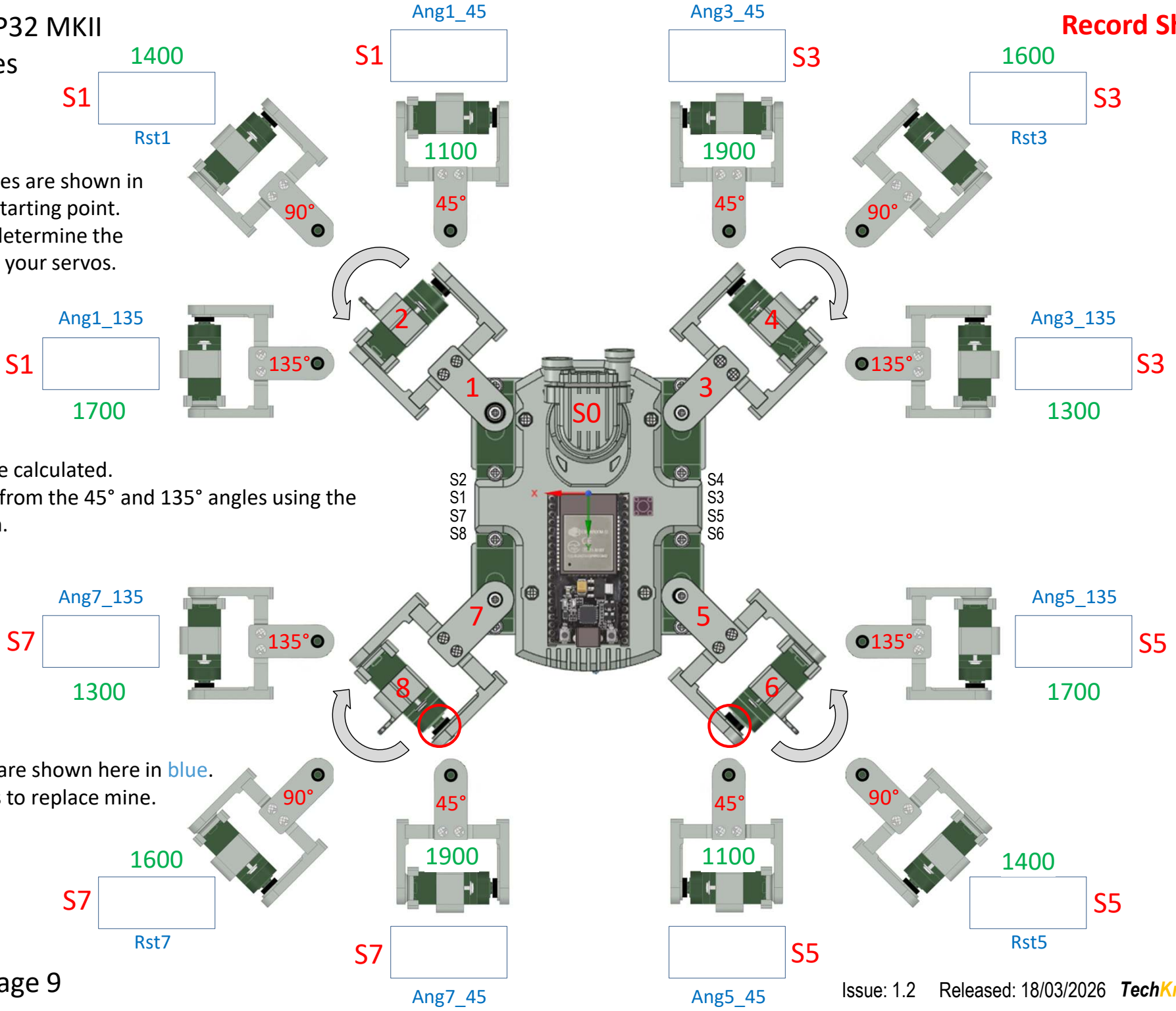
Record Sheet

Code default values are shown in **Green**, as a safe starting point.
You will need to determine the correct values for your servos.

The 90° values are calculated.
They are derived from the 45° and 135° angles using the mapping function.



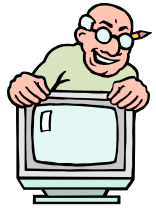
Code references are shown here in blue.
Enter your values to replace mine.



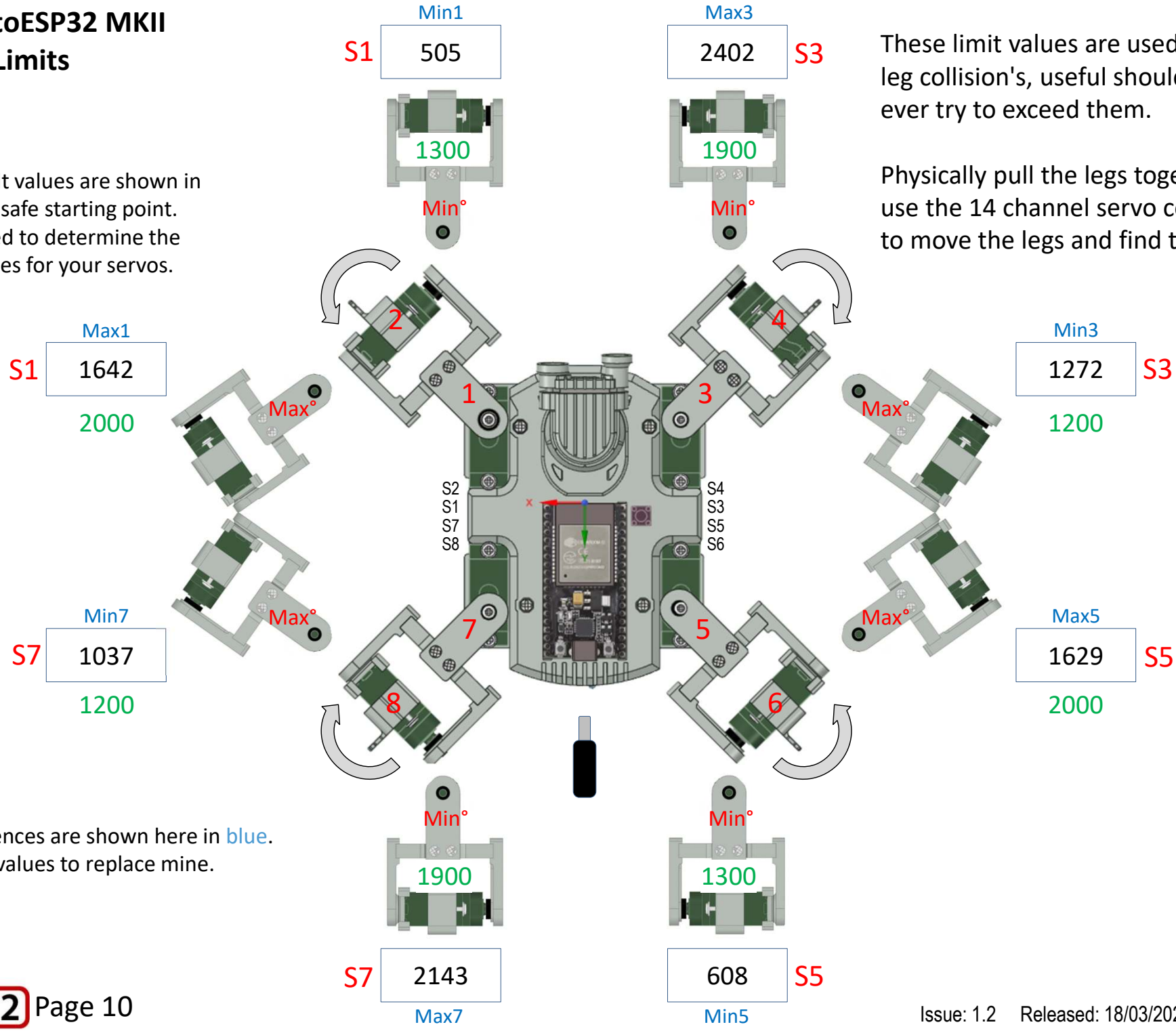
Issue: 1.2 Released: 18/03/2026 TechKnowTone

QuadAutoESP32 MKII
Hip Cal. Limits

Code default values are shown in **Green**, as a safe starting point.
You will need to determine the correct values for your servos.



Code references are shown here in blue.
Enter your values to replace mine.



These limit values are used to prevent leg collision's, useful should your coded ever try to exceed them.

Physically pull the legs together, then use the 14 channel servo controller app to move the legs and find these values.

Issue: 1.2 Released: 18/03/2026 TechKnowTone

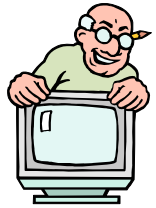
QuadAutoESP32 MKII
Hip Cal. Limits

Record Sheet

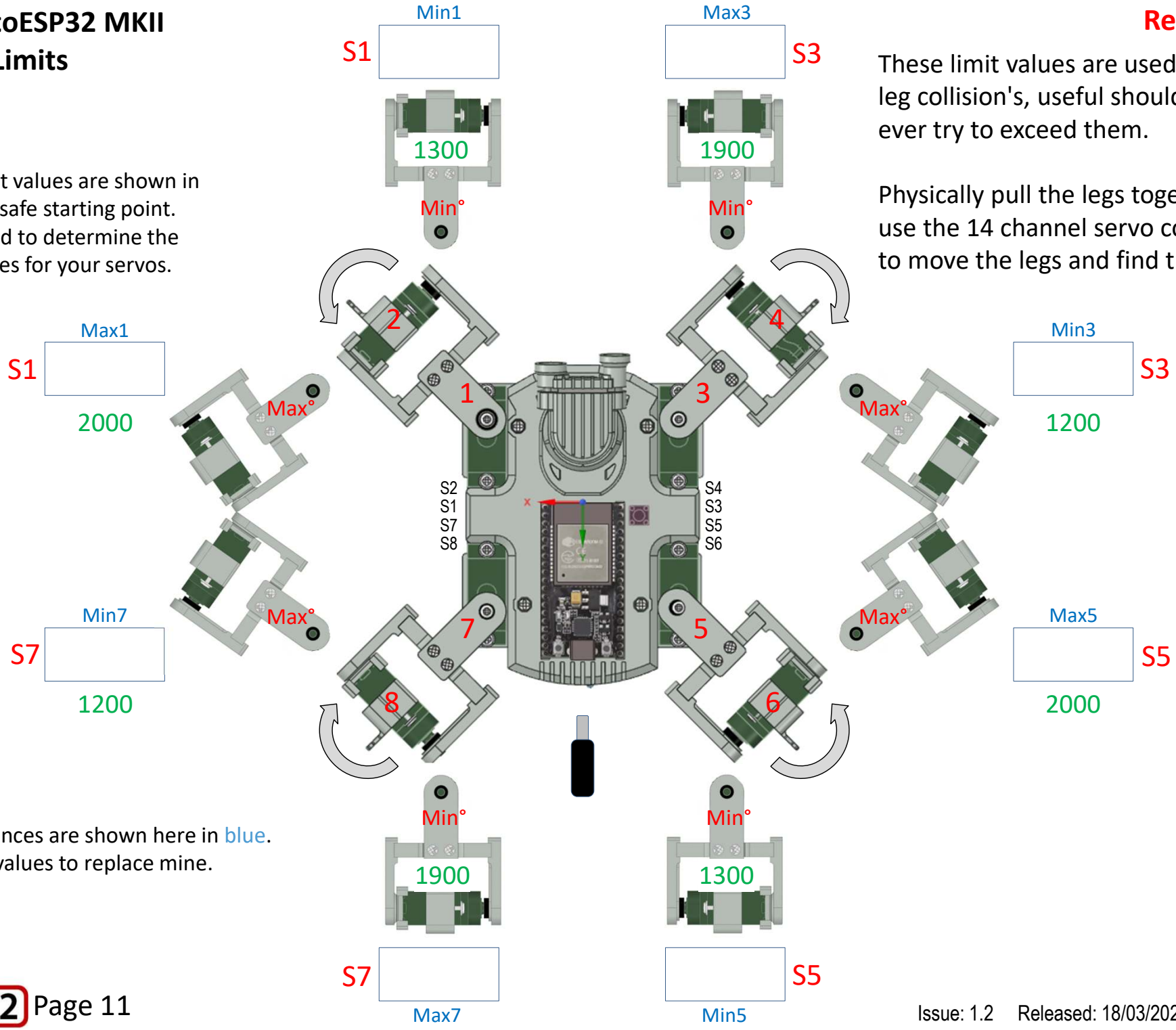
These limit values are used to prevent leg collision's, useful should your coded ever try to exceed them.

Physically pull the legs together, then use the 14 channel servo controller app to move the legs and find these values.

Code default values are shown in **Green**, as a safe starting point.
You will need to determine the correct values for your servos.

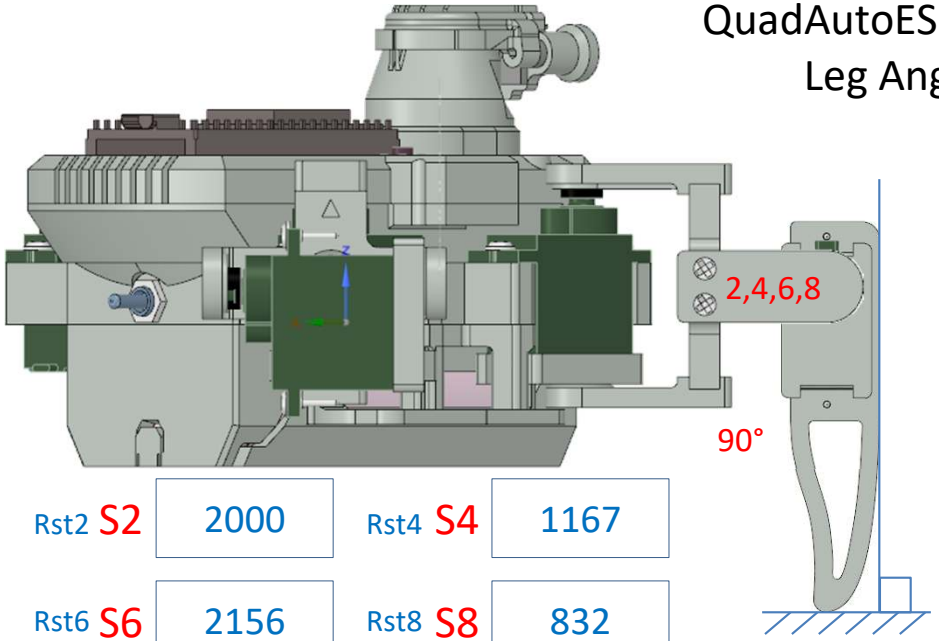


Code references are shown here in blue.
Enter your values to replace mine.

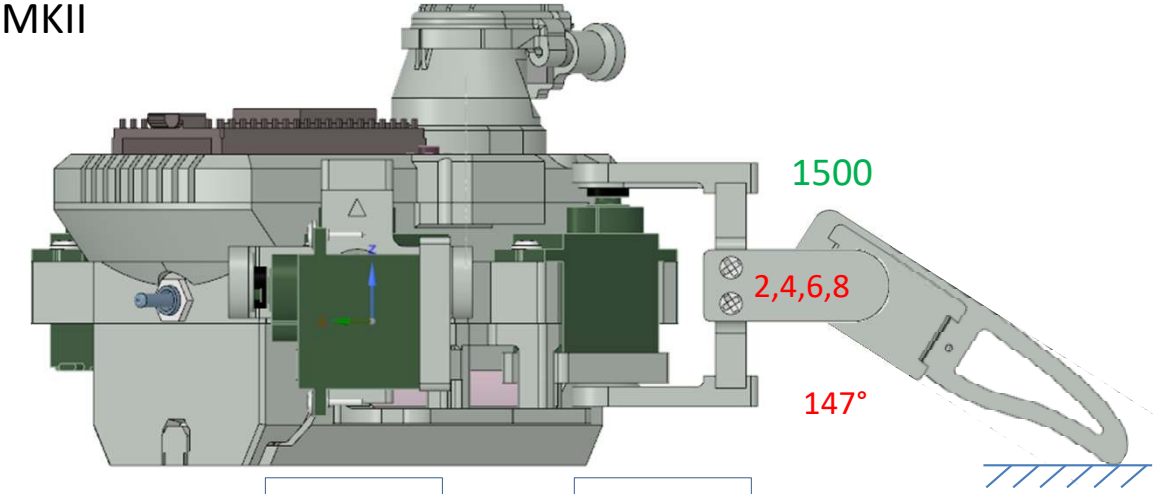


Issue: 1.2 Released: 18/03/2026 TechKnowTone

QuadAutoESP32 MKII
Leg Angles

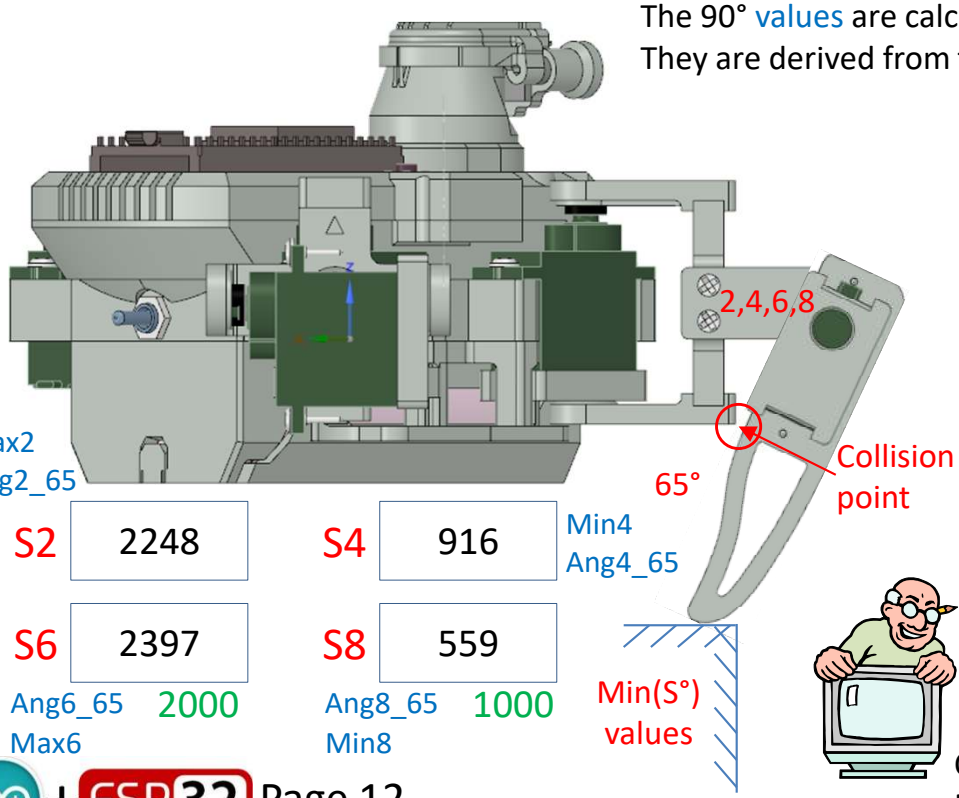


Rst2	S2	2000	Rst4	S4	1167		
Rst6	S6	2156	Rst8	S8	832		

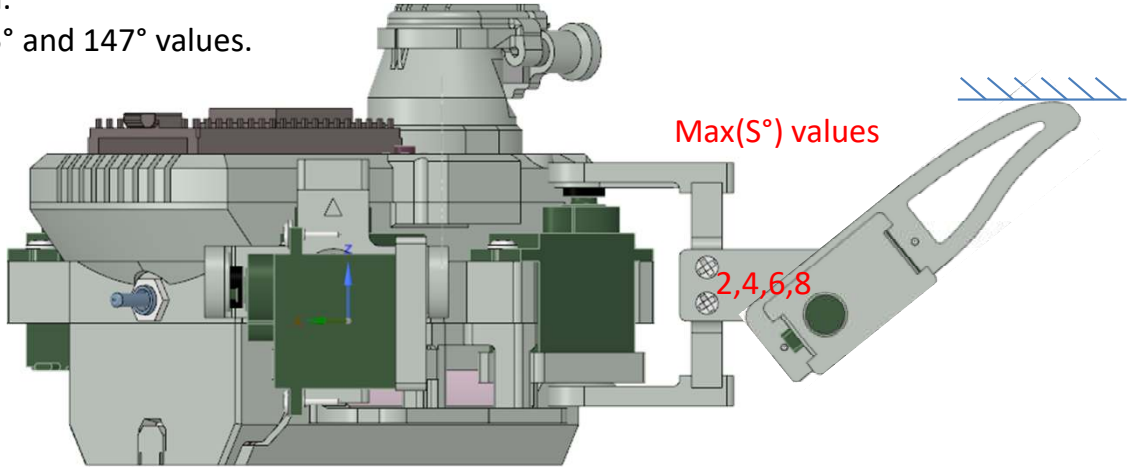


Ang2_147	S2	1436	S4	1738	Ang4_147		
Ang6_147	S6	1605	S8	1454	Ang8_147		

The 90° values are calculated.
They are derived from the 65° and 147° values.



Max2 Ang2_65	S2	2248	S4	916	Min4 Ang4_65		
	S6	2397	S8	559			
Ang6_65 Max6	2000		Ang8_65 Min8	1000			



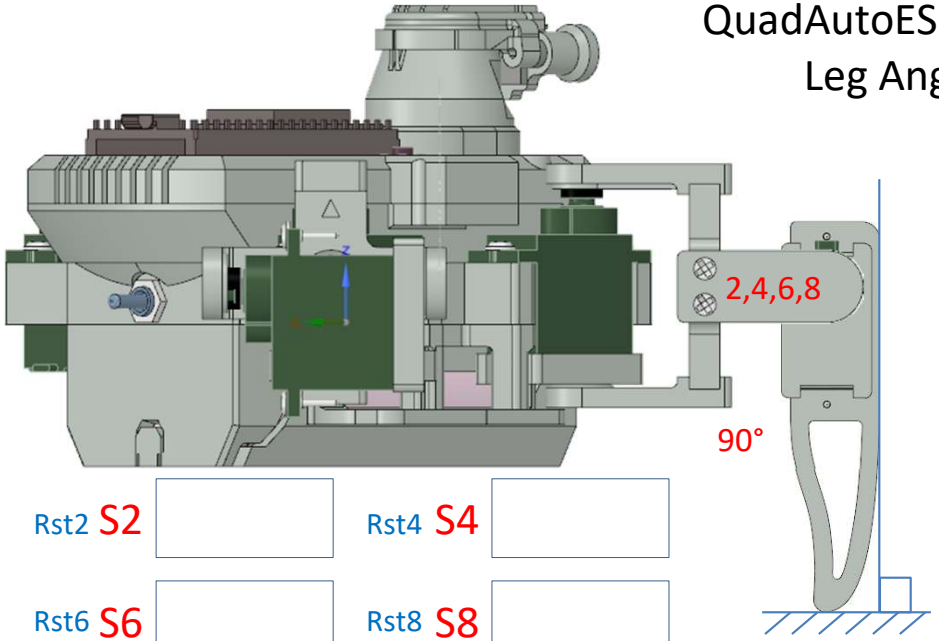
Min2	S2	508	S4	2400	Max4		
Min6	S6	517	S8	2367	Max8		
	900		2100				

Code references are shown here in blue.
Enter your values to replace mine.

Issue: 1.2 Released: 18/03/2026 TechKnowTone

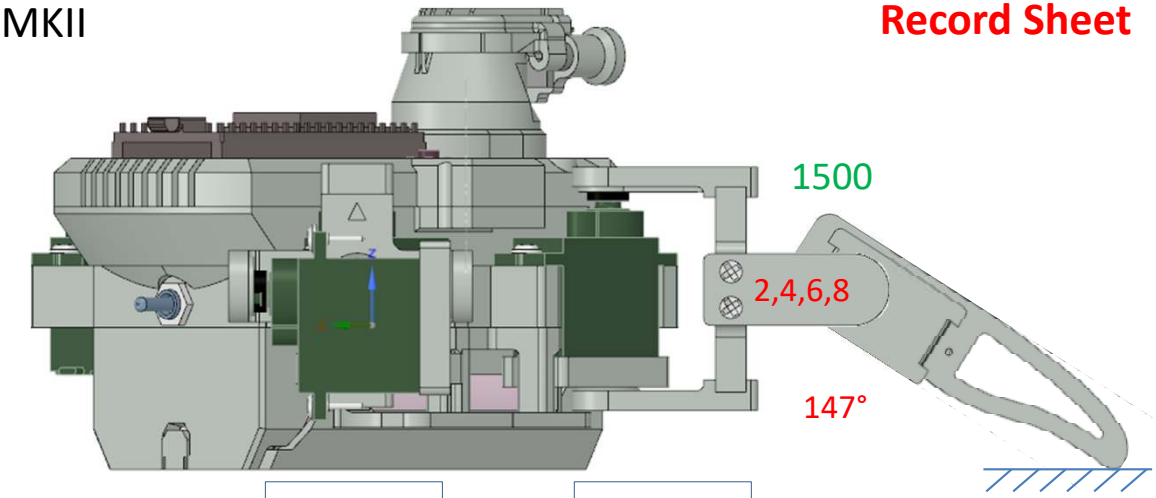
QuadAutoESP32 MKII
Leg Angles

Record Sheet



Rst2 S2 ____ Rst4 S4 ____

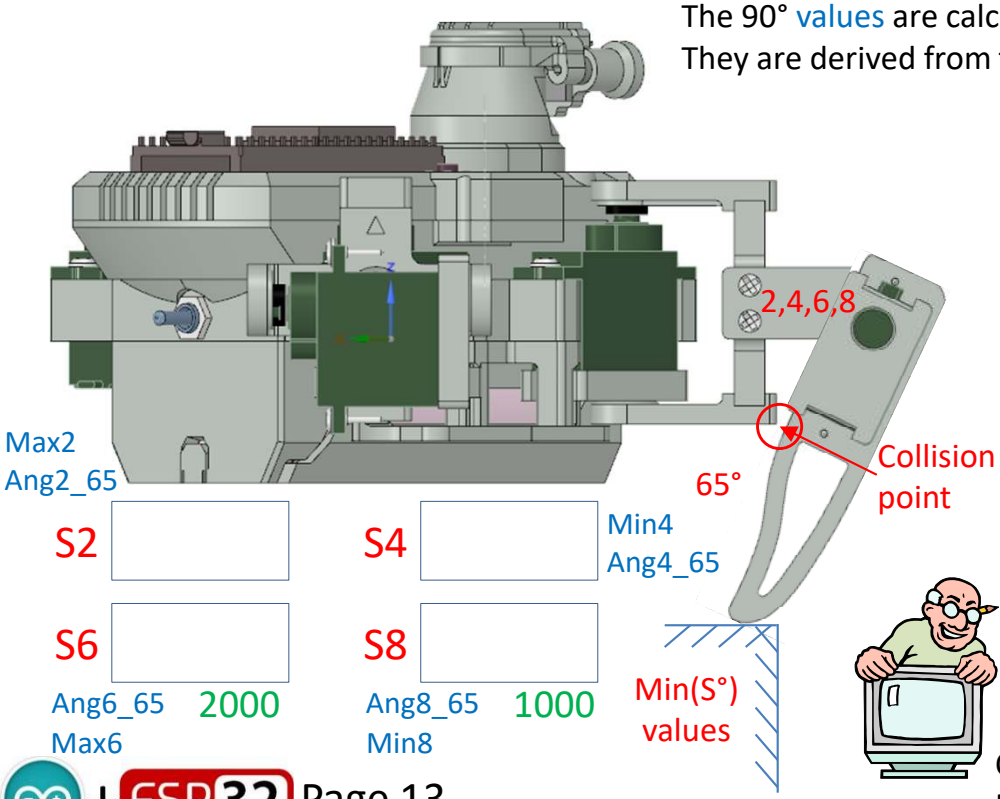
Rst6 S6 ____ Rst8 S8 ____



Ang2_147 S2 ____ S4 ____ Ang4_147

Ang6_147 S6 ____ S8 ____ Ang8_147

The 90° values are calculated.
They are derived from the 65° and 147° values.



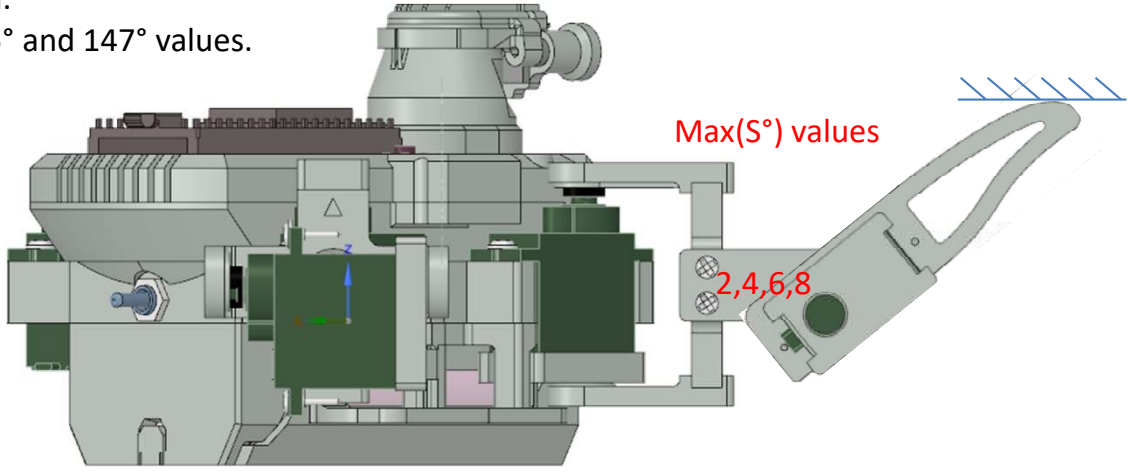
Max2 Ang2_65

S2 ____ S4 ____ Min4 Ang4_65

S6 ____ S8 ____

Ang6_65 2000 Max6

Ang8_65 1000 Min8



Min2 S2 ____ S4 ____ Max4

Min6 S6 ____ S8 ____ Max8

900 2100

Code references are shown here in blue.
Enter your values to replace mine.

Monitor+ App

This Windows based app can be connected either directly via a USB port to the robot, or over the Wi-Fi link, using the transceiver project. It displays bottom left the Windows COM port, which is being used for the PC connection.

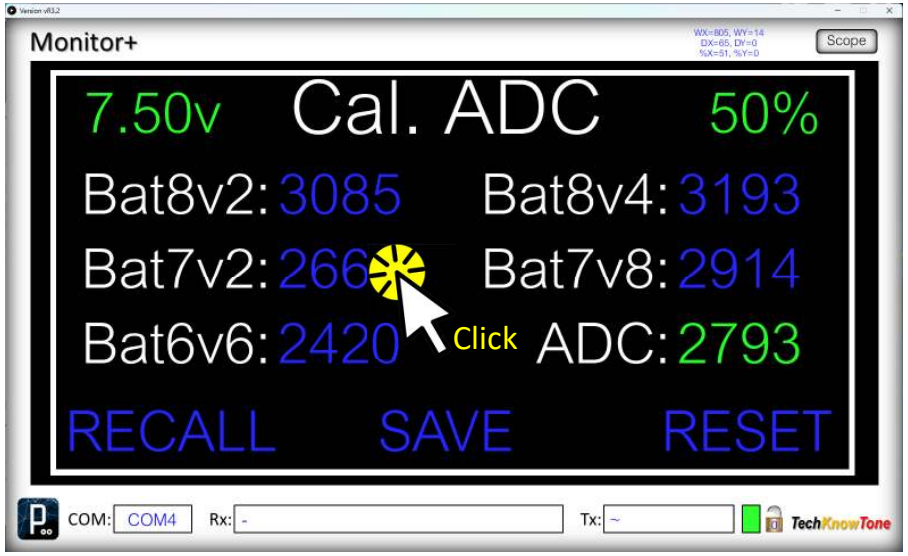
The code in the robot creates the screens, and the user can click on the displays left or right-hand regions in order to switch to another screen. The default screen temporarily displays the version of code in the robot, then moves on to displaying battery voltage and robot status. Text is normally white, but colours are used for emphasis. In particular the colour **BLUE** is used as clickable areas of the display. To start with click on the left-hand side of the display…

Some screens are only displayed when the robot is in TEST mode. To get into this special mode we need to click on the word **TEST** on the Control screen

Remember that **BLUE** text implies that the text is active, behaving like a button switch, or enabling the change of values by clicking on the field.

The word TEST is then replaced with **Normal**, and a white box is drawn around <u>all</u> screens, to indicate that TEST mode is active. This special mode then allows other screens to be displayed, that aren't normally seen. To get to the first screen 'Cal. ADC' simply click on the left-hand area of the screen again…

Monitor+ ADC Cal. Screen

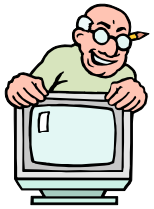


Cal. ADC: This screen shows the voltage threshold values, used to calibrate the ESP32's ADC against the supply voltage. As the battery discharge curve is not linear, and the ESP32's ADC may not be accurate, we use this screen to store key threshold values, against which more accurate readings can be calculated.

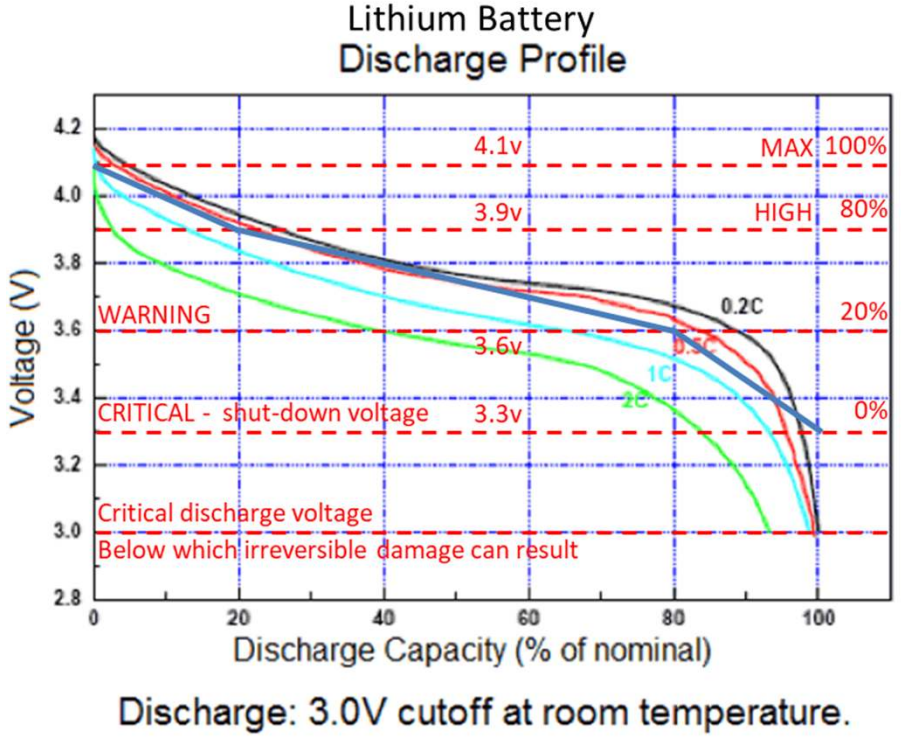
There are five threshold values, ranging from 6.6v to 8.4v. The lower figure is the critical shutdown threshold, at which it is assumed the battery is exhausted. The upper figure is the assumed maximum battery voltage. In TEST mode the ADC value is being constantly read, and displayed in **GREEN** on the right-hand side of the screen. The calculated voltage and estimated percentage capacity, are also shown at the top of the screen in green.

Connect the robot to a variable DC power supply, using the external power socket. It is also recommended that you attach a multimeter, to display the supply voltage, as power supply displays aren't always accurate. In TEST mode you can freely adjust the supply voltage in the range 6.0v to 9.0v, and watch the ADC value change accordingly. To calibrate the system, simply adjust the supply to match one of the threshold values, say Bat7v2 (7.2v), and click on the blue number next to it. That will cause the number to change to the current ADC value. Do this in turn for all of the five values, adjusting the power supply each time, to correspond. Once you have set all five values, then click on **SAVE** to store them in EEPROM.

Note: that the setting of values is constrained, such that the ADC value assigned to say Bat7v2, can not be higher than that assigned to Bat7v8, or lower than that assigned to Bat6v6.



It is recommended that you note down all of the values applied. In case you lose them by accident, or change the micro in the future. If you are only making one robot, then you can enter them into your code, as definition statements. They will then be loaded by default whenever the EEPROM is erased.



Discharge: 3.0V cutoff at room temperature.

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 3v3 voltage regulator, connected to the 3v3 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
Place the droid in TEST mode. Using a Multimeter and a variable DC supply, determine the following V_{ADC} values for corresponding threshold voltages:

MAX. O.C V_{OC} = 8.4v, gave A0 = 3295 On V_{ADC} (2 x 4.2v)

MAX: (100%) V_M = 8.2v, gave A0 = 3200 on V_{ADC} (2 x 4.1v)

HIGH: (80%) V_H = 7.8v, gave A0 = 2997 on V_{ADC} (2 x 3.9v)

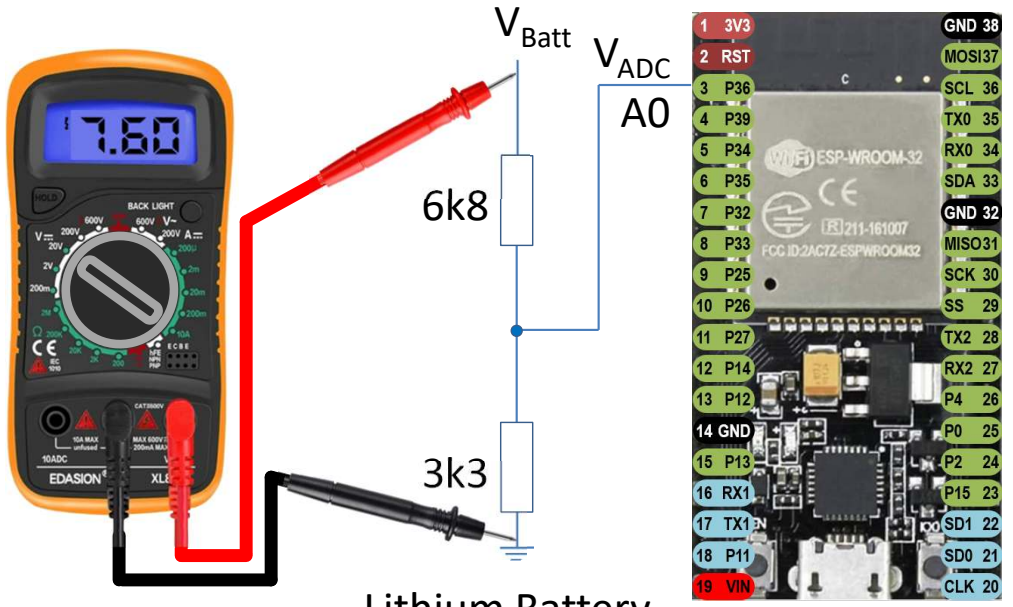
WARNING: (20%) V_{BW} = 7.2v, gives A0 = 2762 on V_{ADC} (2 x 3.6v)

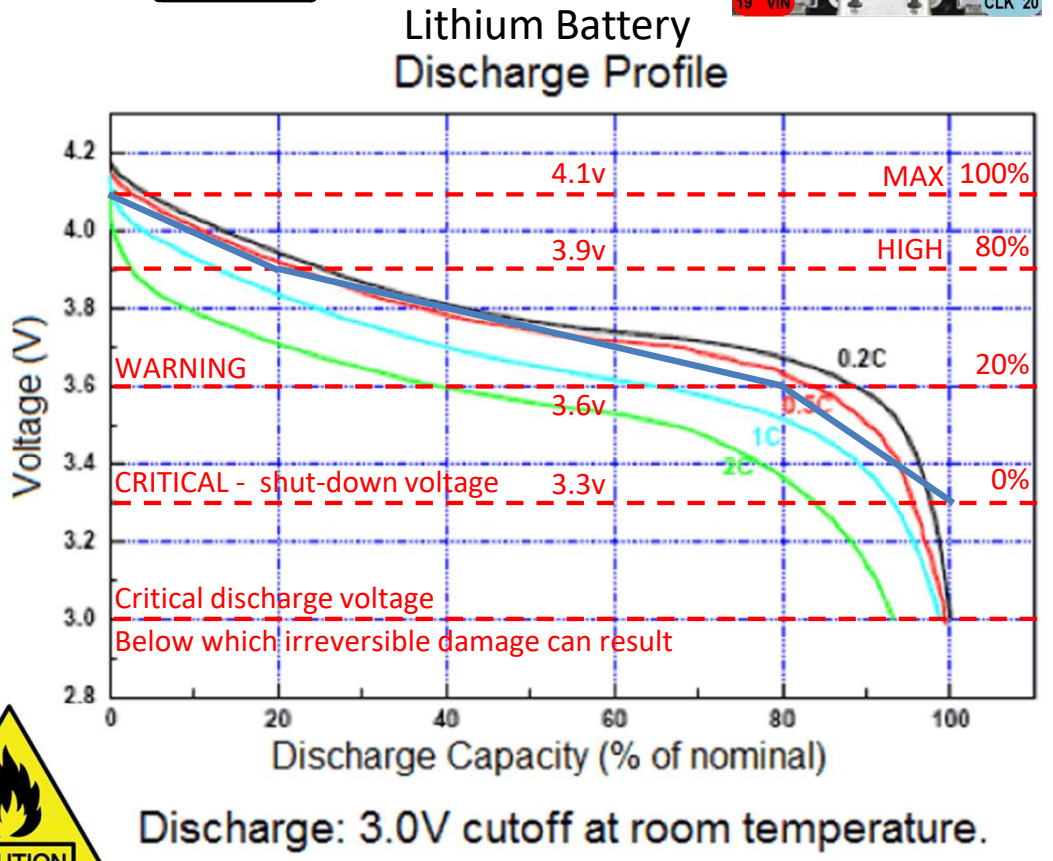
CRITICAL: (0%) V_{BC} = 6.6v, gives A0 = 2513 on V_{ADC} (2 x 3.3v)

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.

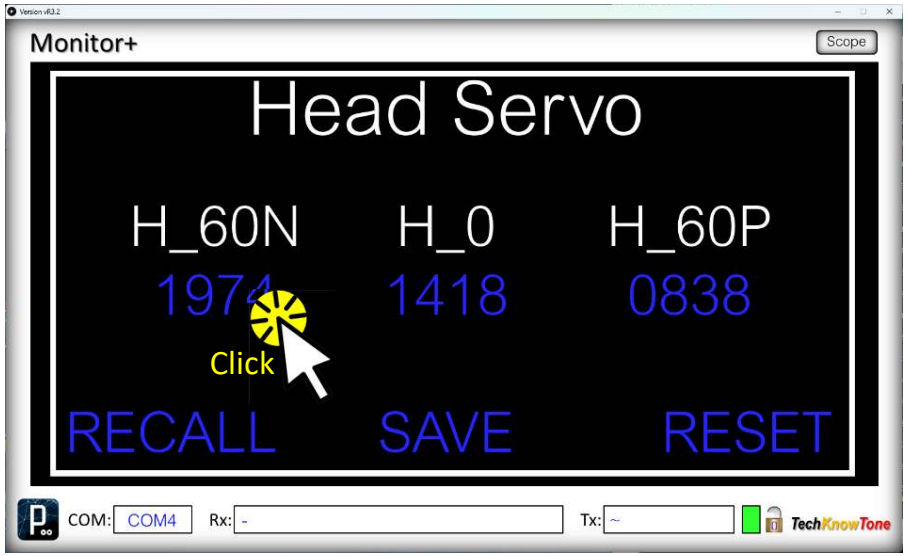
Note: If connected to USB port with internal battery switched OFF the ADC will read a value 5 volts (A0 = 1919) or less. So, if the micro starts with such a low reading it knows that it is on USB power, which limits functions available.





Discharge: 3.0V cutoff at room temperature.

Monitor+ Servo Screens



Head Servo: This screen shows the 3 head servo calibration values. Values are limited in the range 500 – 2500µs, and you change them by simply clicking on the digits. As you do so, the servo will also be driven to that PWM angle, so take care, as it is easy to set values that are beyond the limit of the servo. To move the servo to a PWM value, without changing it, simply hold down the **SHIFT** key, before clicking the mouse button. The values are also held within a range, so H_0 can not be set larger than H_60N. The lower options are as follows:

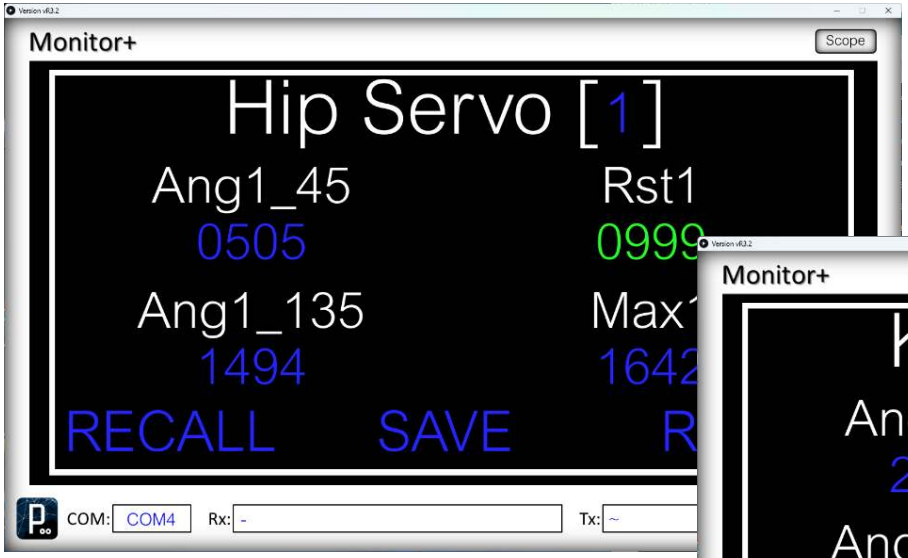
RECALL	Returns the variable values to EEPROM stored values.
SAVE	Overwrites EEPROM values with these new settings.
RESET	Replaces the 3 variable values with those defined in your code.

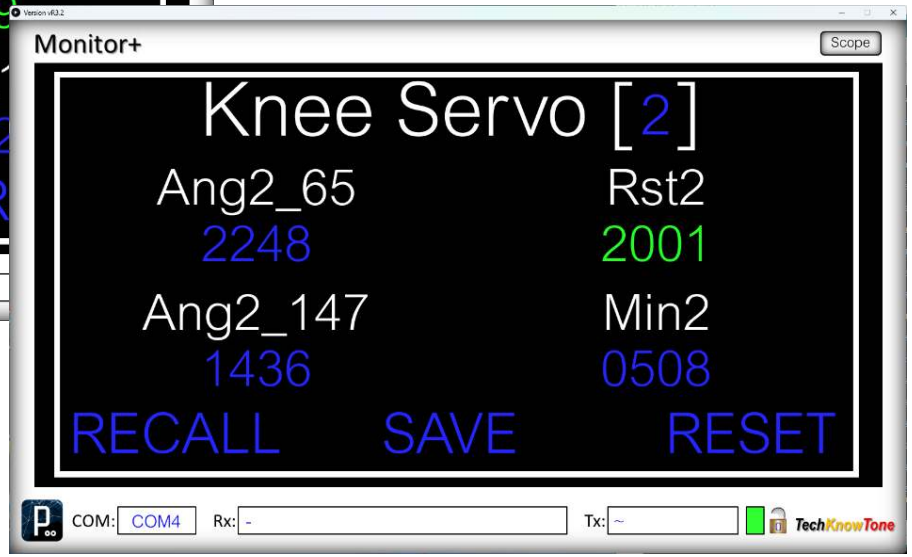
If you click on the left-hand side of the display you will move to other screens...

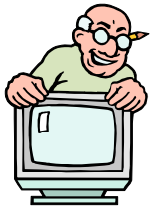
The hip and knee servos, in each leg, have 3 calibration values, and a GREEN calculated value.

You select the servo number by clicking in the top [**?**] field. These screens work in the same way as the head servo screen.

So, you can use the Monitor+ app to determine and store all of your servo calibration values.







It is recommended that you note down all of the values applied. In case you lose them by accident, or change the micro in the future. If you are only making one robot, then you can enter them into your code, as definition statements. They will then be loaded by default whenever the EEPROM is erased.

EEPROM RESET

This screen gives you the ability to reset all of the EEPROM values to the default code defined state. Unlike servo screens, which just reset the three values shown, this screen resets all of them.

This is effectively what happens when you first load code into a new ESP32 microcontroller. There is nothing in the EEPROM memory, and the check byte data is invalid. So the micro will load all of the code defined values as defaults, and save them to EEPROM; for use when it next boots up.

Tips:

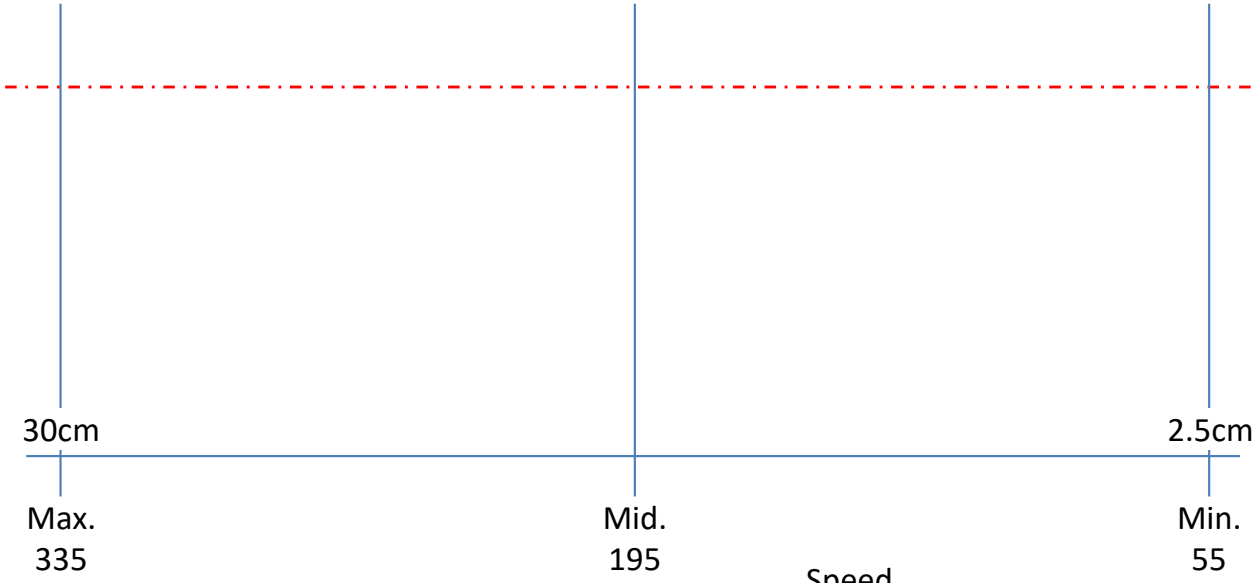
- For the head servo, start by adjusting the H_0 (Head_0) value, to place the head in the centre facing position, in the region of 1500µs. Note, my values will be incorrect. Then set H60N at 300µs higher, and H_60P at 300µs lower. Then carefully adjust them to the correct +/- 60° angles.

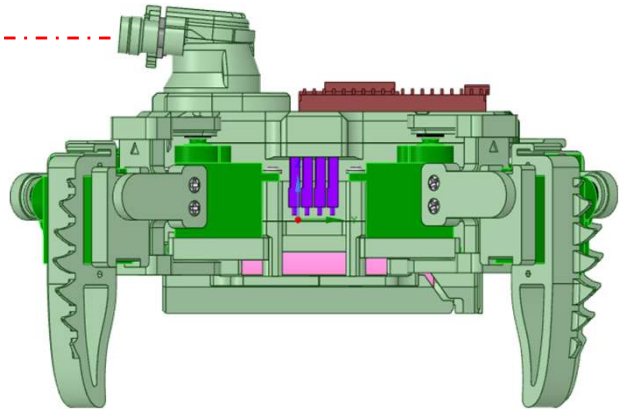
- For hip servos, start by setting the Ang?_135 values, as these are the 90° angles, and the servos are less likely to have collisions in this region. Then on Max screens, set the Ang?_45 values 500µs lower; and on the Min screens set the Ang?_45 values 500µs higher. Then adjust Ang?_45 values from there. This avoids collisions with the robots side body.

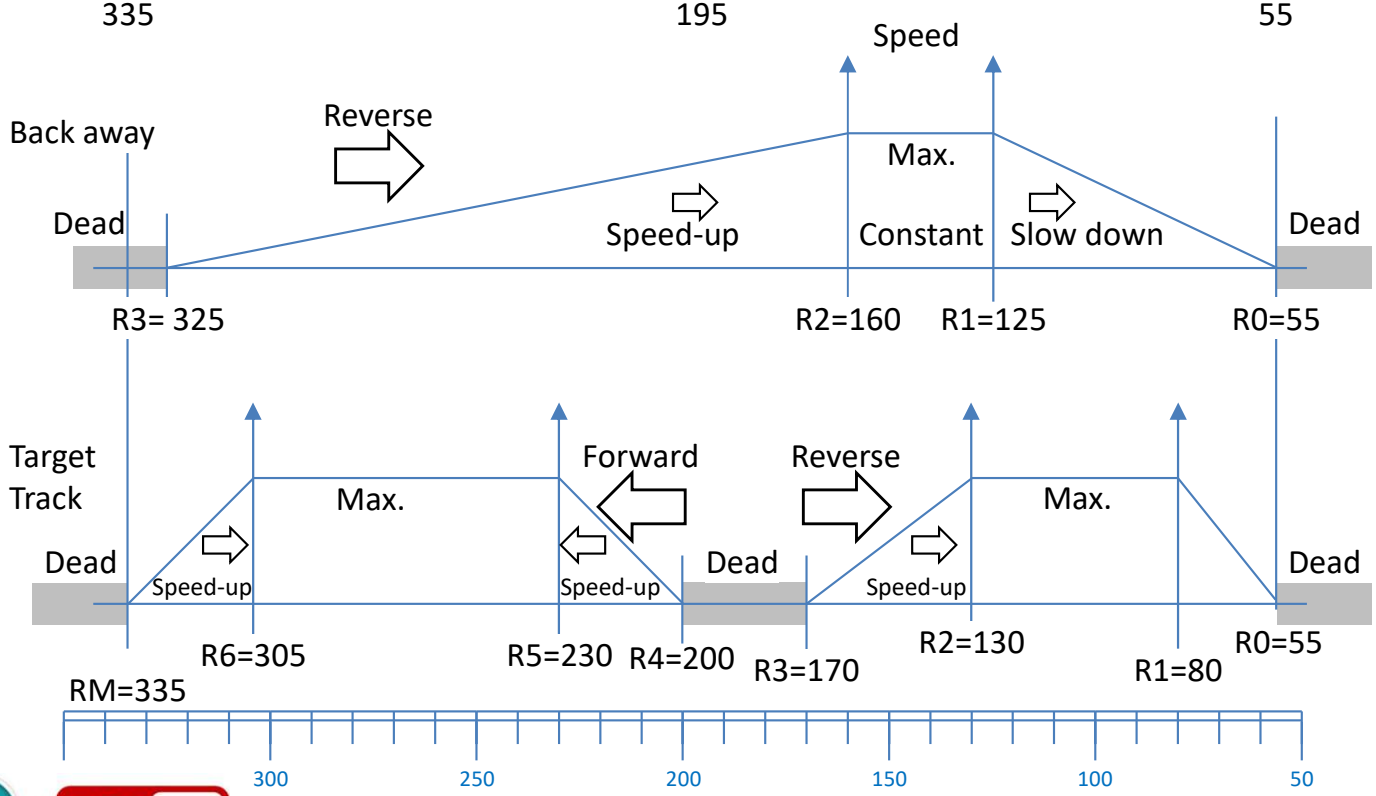
- For knee servos, stat by setting the Ang?_147 values to 1500µs, which should put the legs out at a safe angle. On screens with Max values, start with Max values being 300µs higher, and Ang?_65 values 300µs lower as there starting point. On screens with Min values, start with Min values being 300µs Lower, and Ang?_65 values 300µs higher, as their starting points.

- The time spent and accuracy of your setting will determine how well your robot performs. So apply time and patience, and always record you values once you are happy with them.

VL53L0X LTOF Sensor Ranges



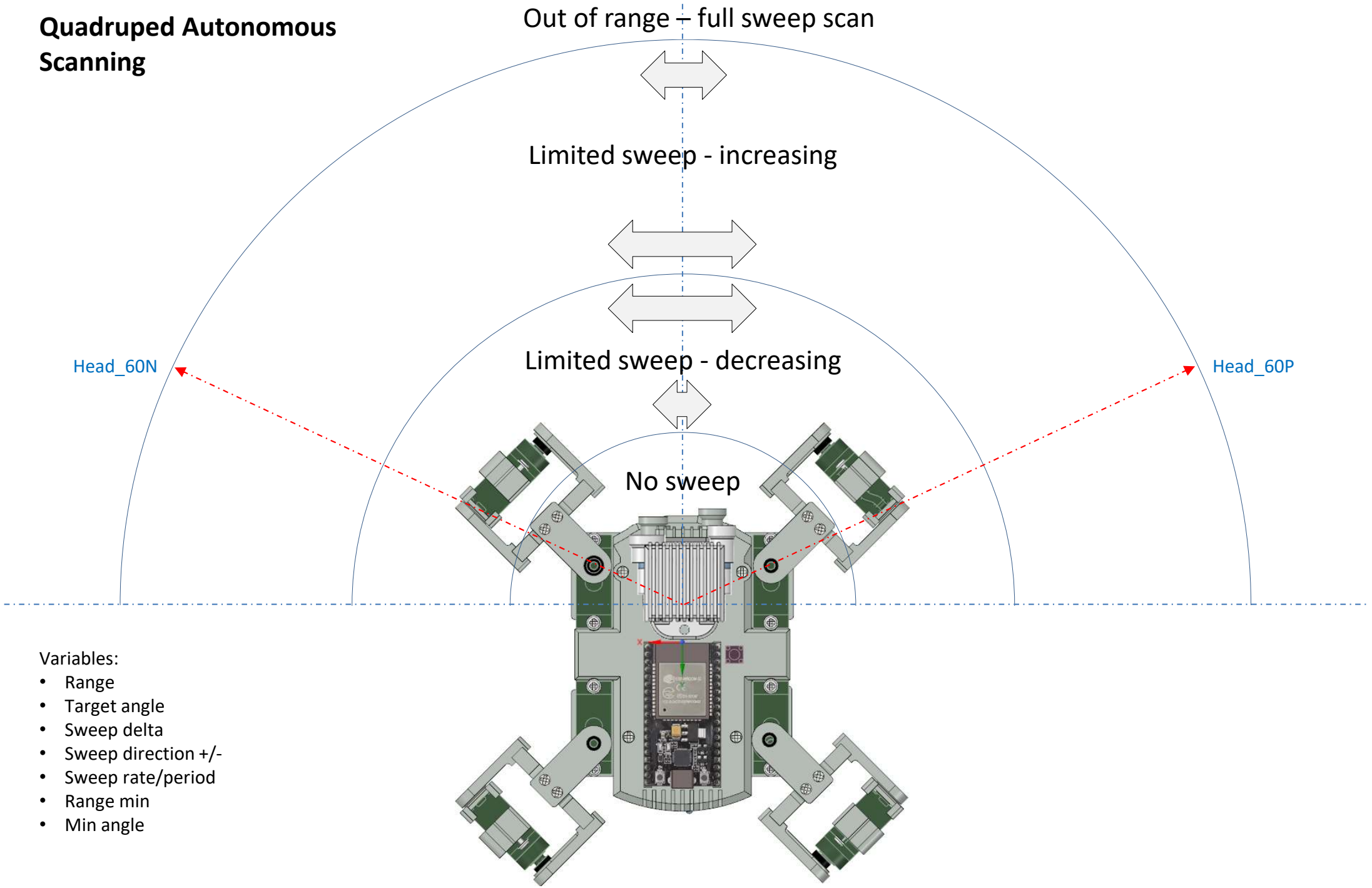




These are the values I determined from my LTOF sensor, and the speed maps I developed for the back away and target tracking functions.

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.

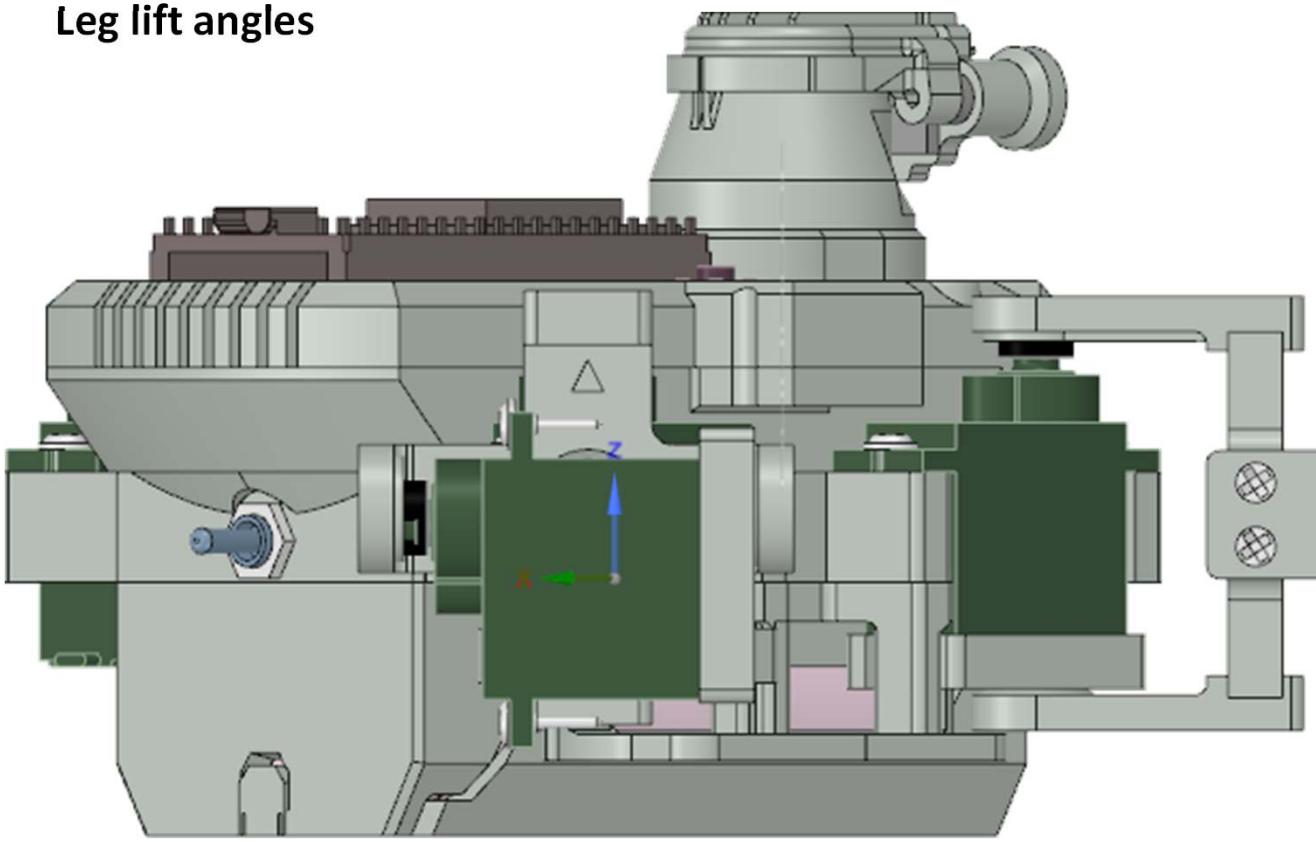
Quadruped Autonomous Scanning

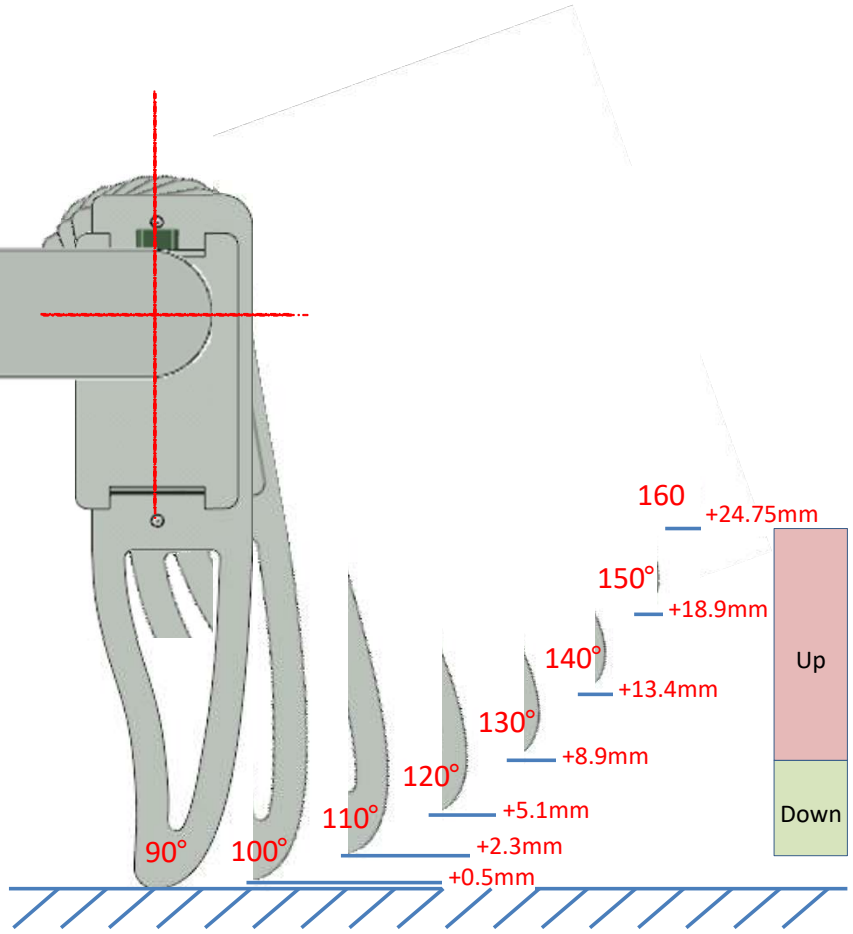


Variables:

- Range
- Target angle
- Sweep delta
- Sweep direction +/-
- Sweep rate/period
- Range min
- Min angle

Leg lift angles





This diagram gives an indication of how high the legs lift from the ground for a given servo angle; where 90° is calibrated as the vertical ‘standing’ angle.

When walking on a hard floor you will only need a small amount of lift on the freely moving legs for it to walk; where as in thick piled carpet the robot will sink in and a higher leg lift will be needed to avoid unnecessary drag. You could change the code to use options to set different heights.

Note: If the down angle is set below 110° the legs will begin to drag on the surface, as they lift. Increasing the down angle, lowers the robot’s belly height clearance. The minimum up angle must always be greater than the max down angle, for leg lift to occur.