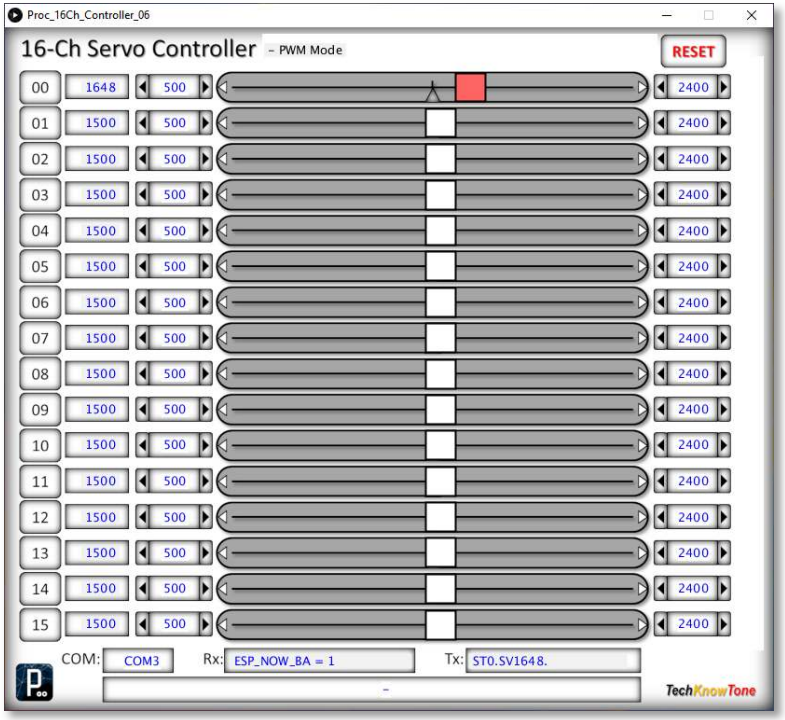
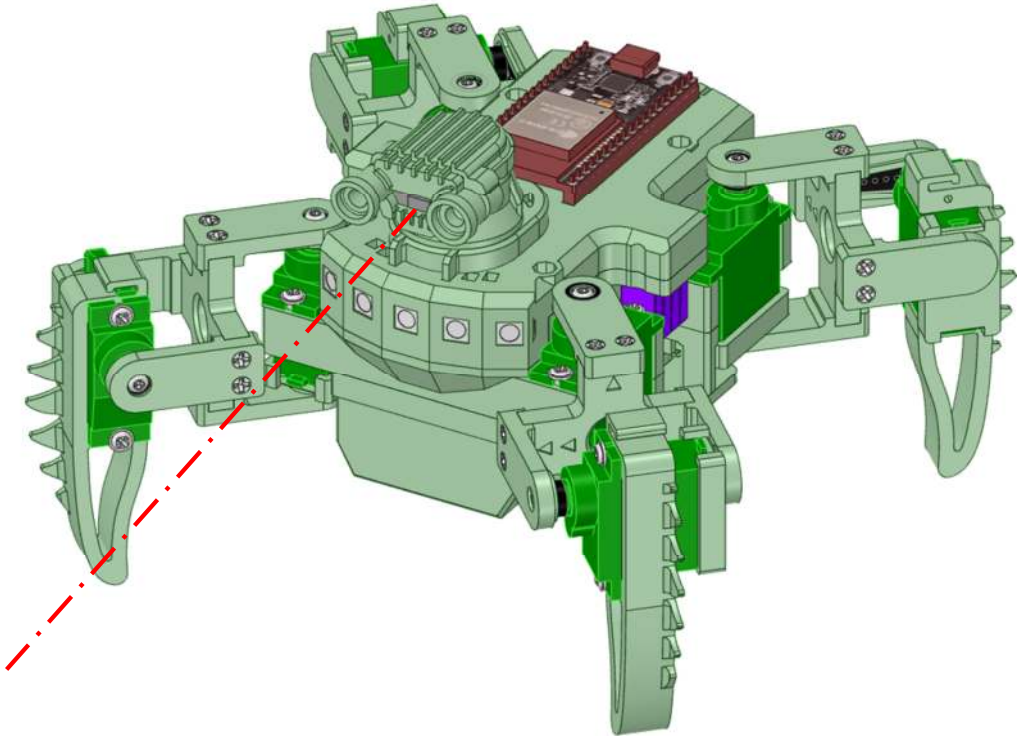
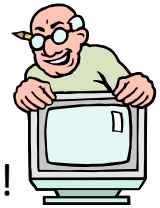
DRAFT

Autonomous Quadruped Robot ESP32 MkII

Calibration

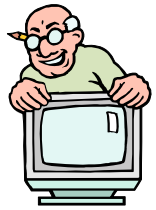






Servo calibration is an essential process!

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.

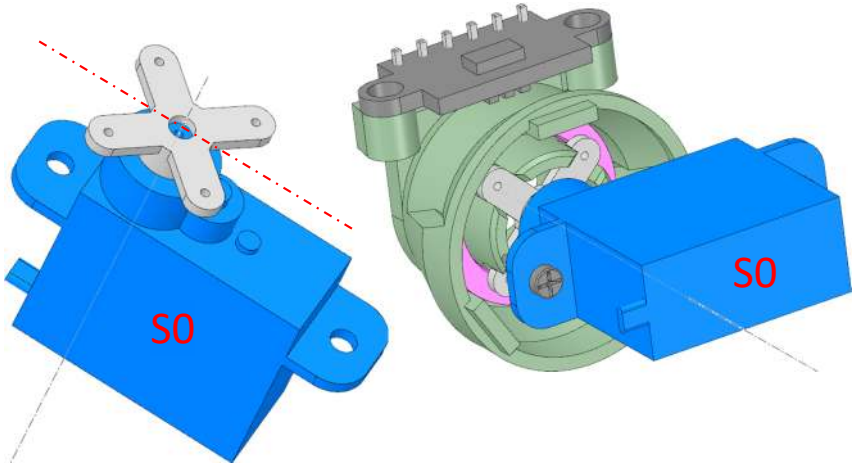
Carefully dispose of batteries that damaged, or discharged below their critical voltage.

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 a custom Windows app, written specifically for this project by me.

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.

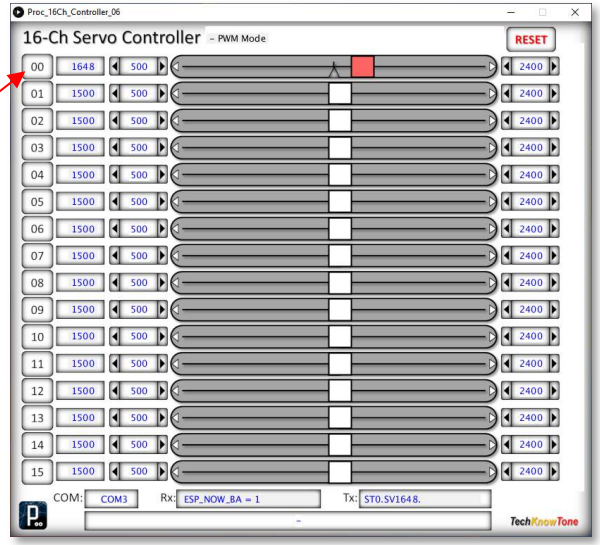
The PWM centre position of a servo is 1,500 µs.



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.

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 to set the leg servos.

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

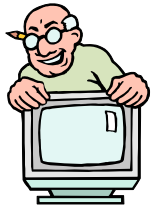


The robot should be placed on its stand, code set to TESTmode true, powered up and connected to your PC using a USB lead. You need to turn ON a channel, by clicking on the channel button, before it will receive PWM signals and respond to the slider values. Turning the channel OFF will remove the PWM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

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 references are shown here in blue.

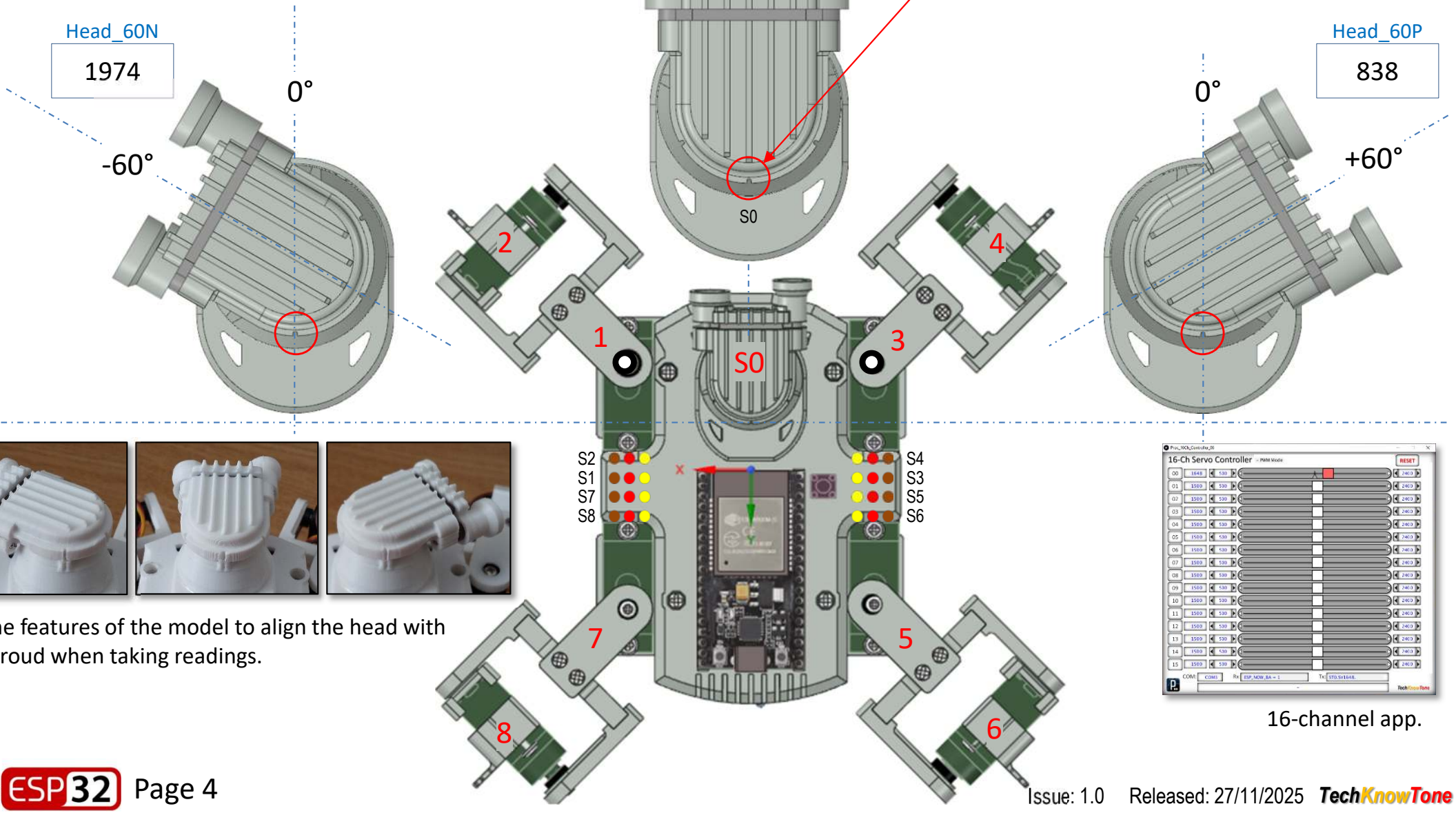


Head_0

1418

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.



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

16-channel app.

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.

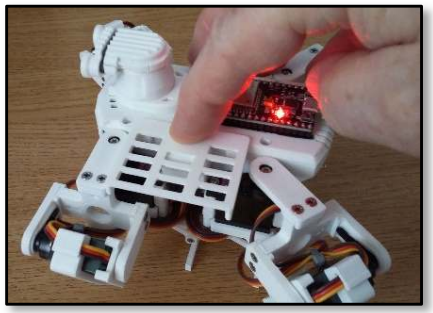
Course: Leg hip servo leaver arms are set with the servo mounted in the robots body, at approximately 45° which corresponds to a PWM value of 1500µs. The centre of the hip servos mounting screw gives you an indication of this angle.

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.

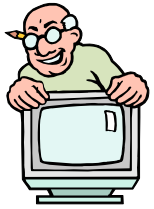
Fine: Once you have assembled and wired up your robot you can use the 16-channel application I have provided. You can select each servo in turn and move their sliders to determine values needed to achieve the calibration angles shown in the subsequent pages of this guide. Take care to avoid collisions,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

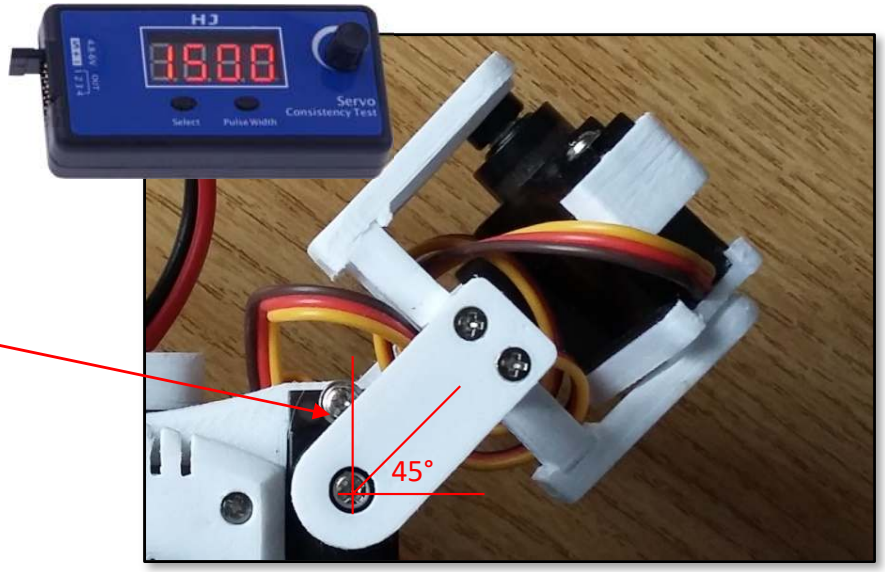
Note that as no two servos are alike, the values I have given in this guide, and included in the sample code, will be similar but different from the ones you derive from your robot. That's the nature of servos!

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

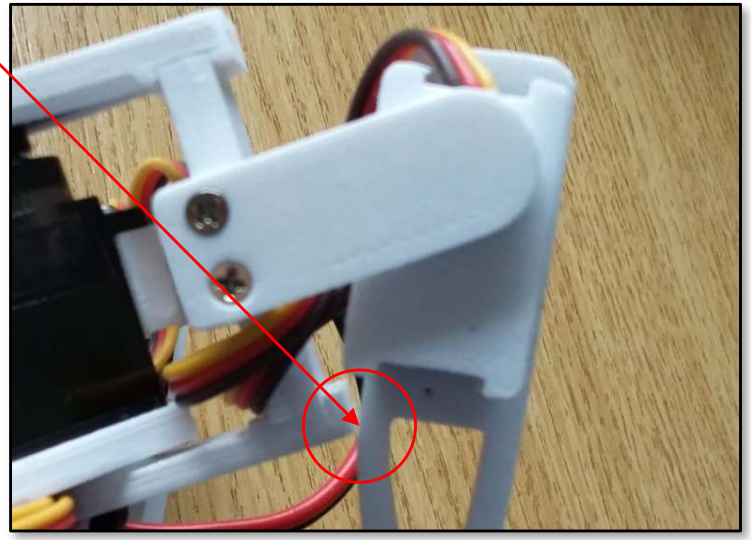


There is an angle gauge included in the model files which will help you determine the hip 135 angles. Simply hold it against the robot, whilst adjusting the appropriate slider in the app.





Fit the hip joints at 1500µs == 45°

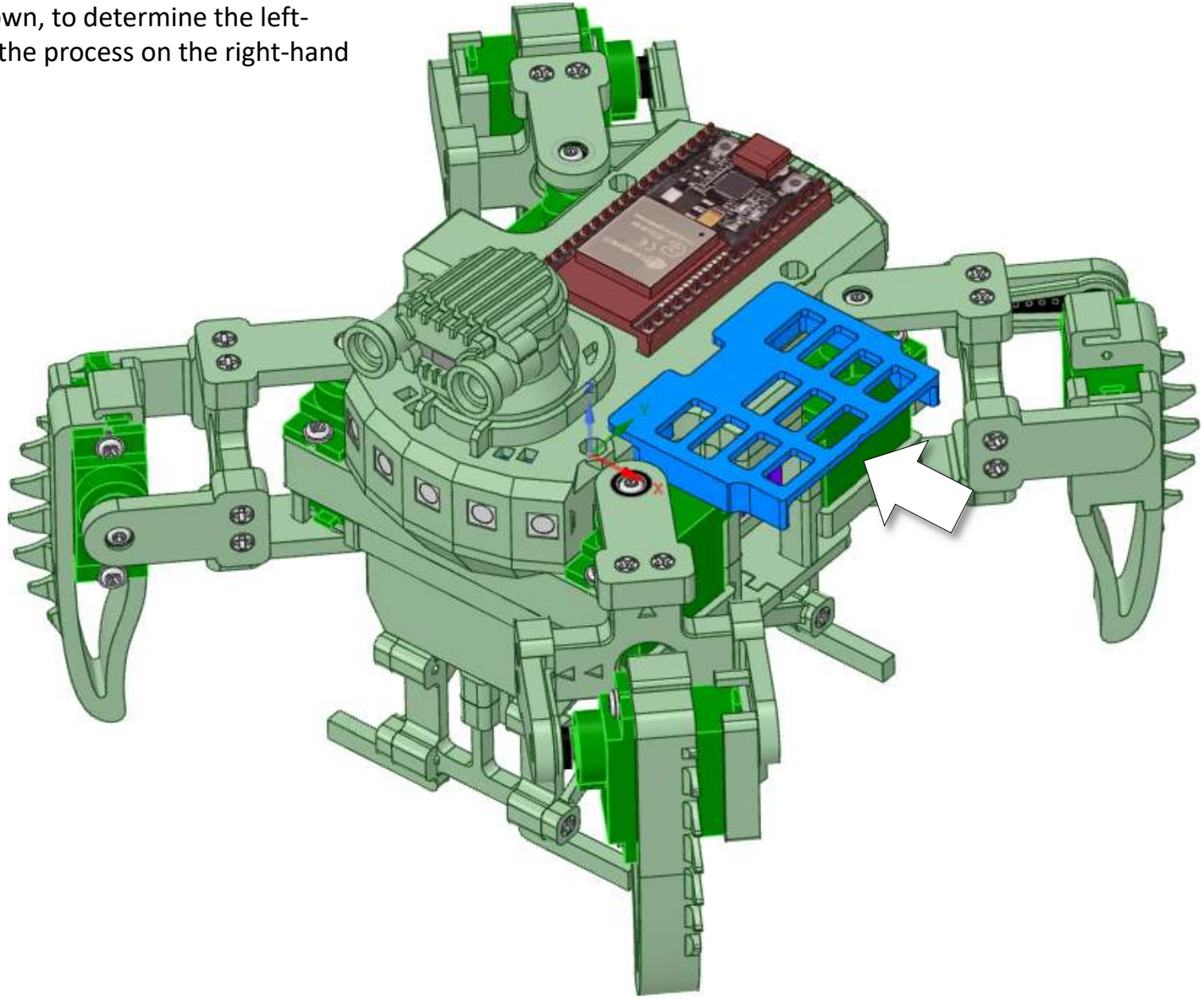


Fit legs just touching their collision points.

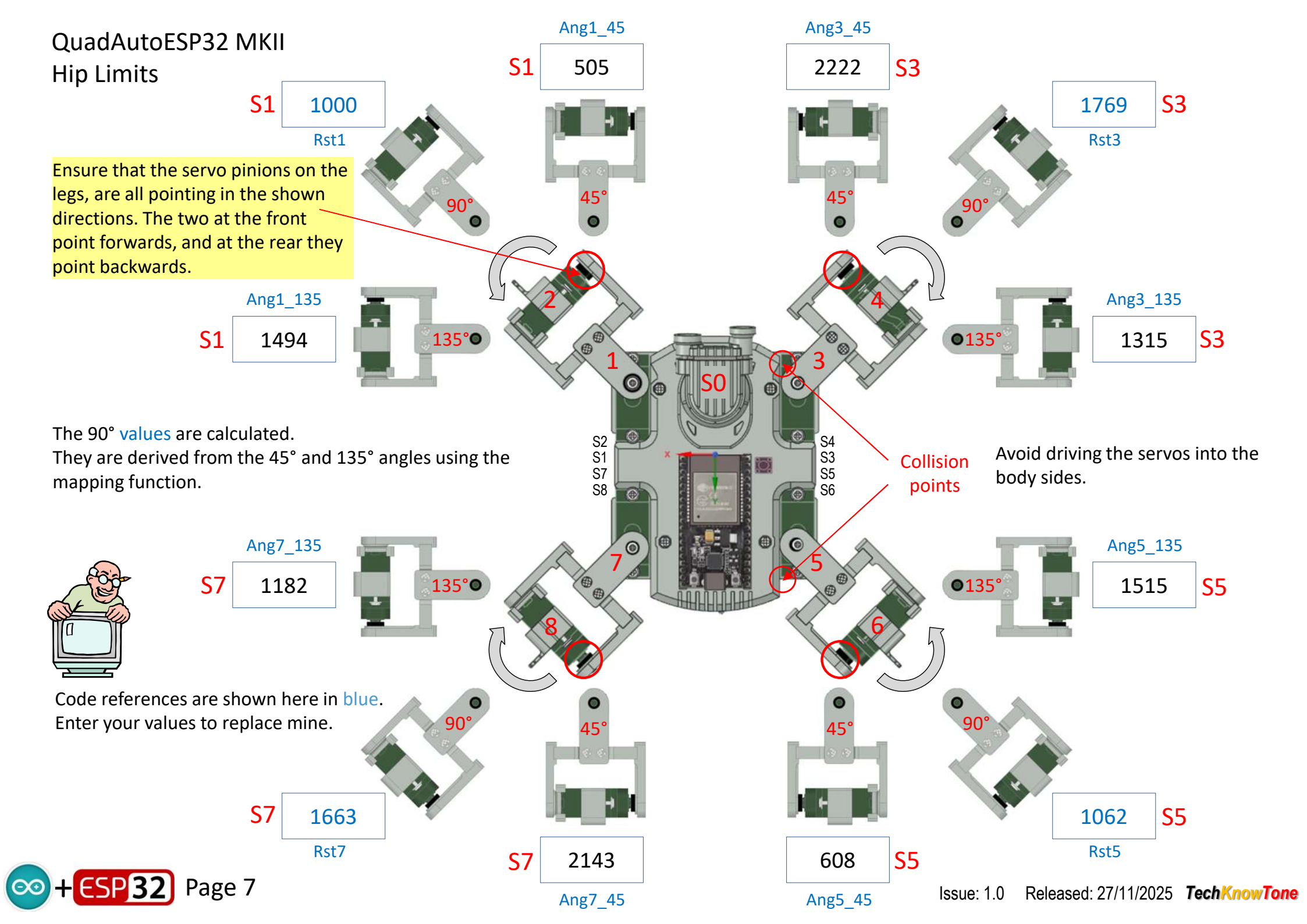
Leg servo Ang_135 gauge

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.



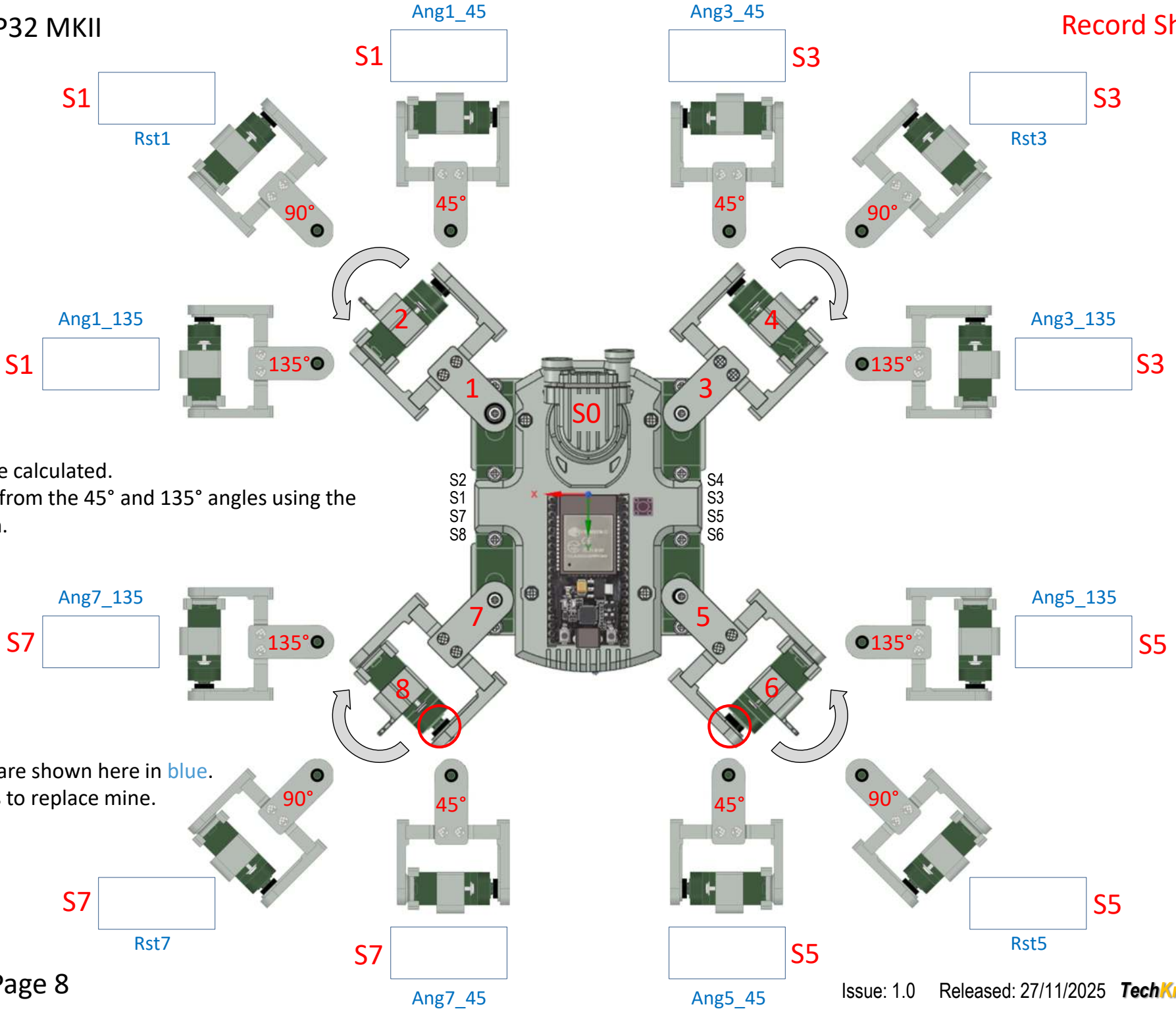
QuadAutoESP32 MKII
Hip Limits



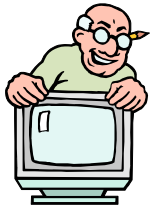
Issue: 1.0 Released: 27/11/2025 TechKnowTone

QuadAutoESP32 MKII
Hip Limits

Record Sheet



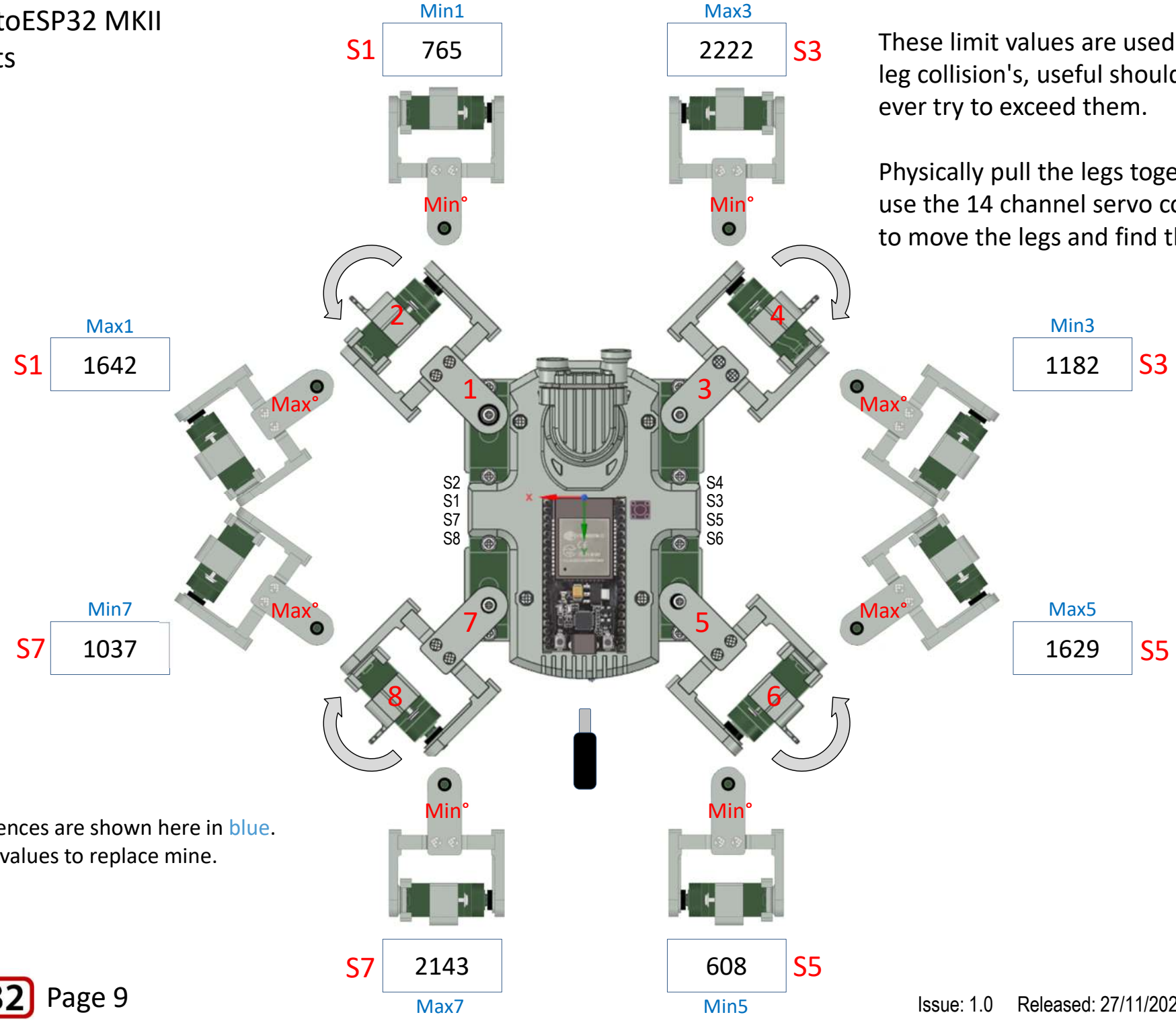
The 90° values are calculated.
They are derived from the 45° and 135° angles using the mapping function.



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

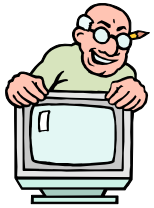
Issue: 1.0 Released: 27/11/2025 TechKnowTone

QuadAutoESP32 MKII
Hip Limits



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

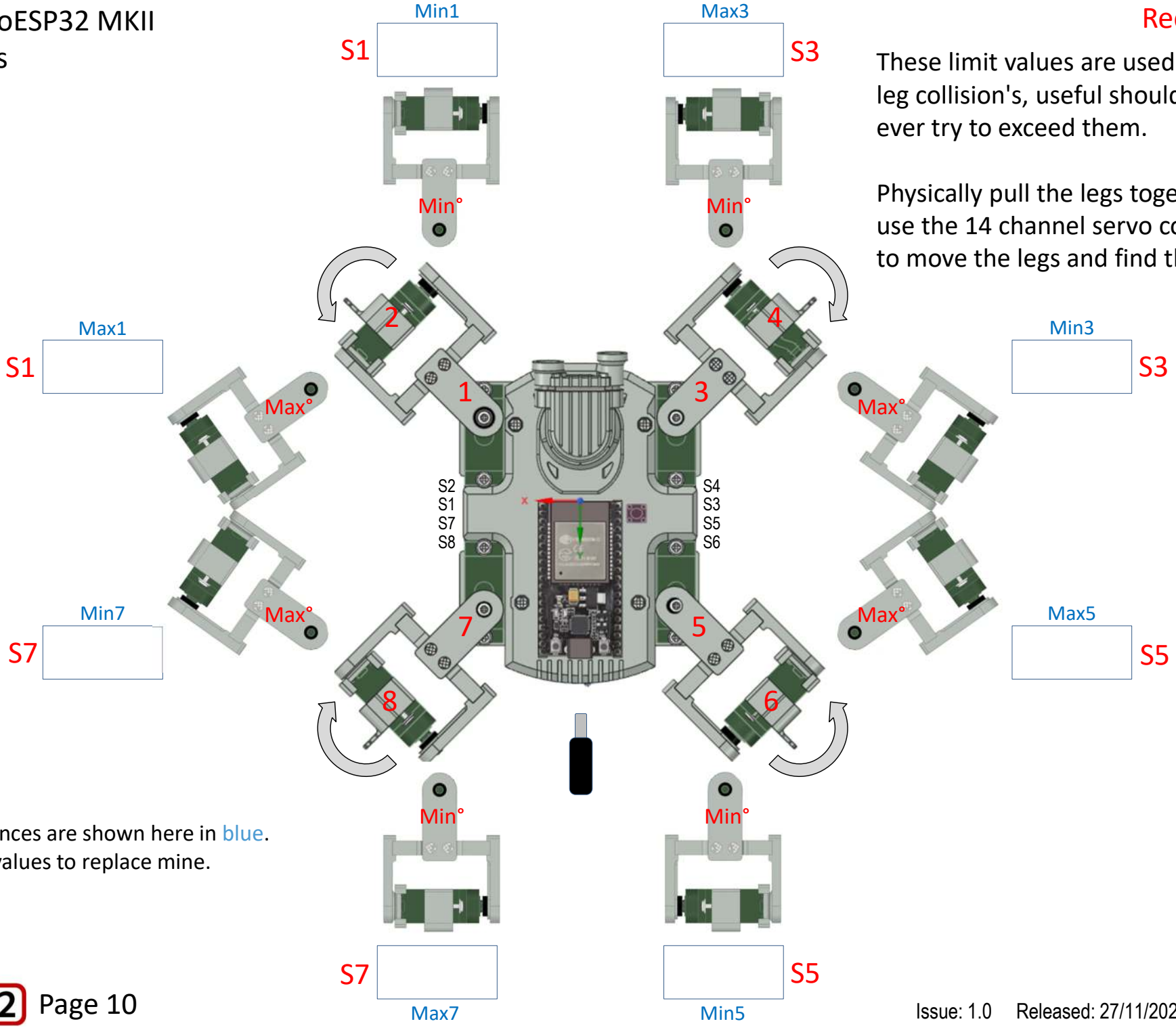
Issue: 1.0 Released: 27/11/2025 TechKnowTone

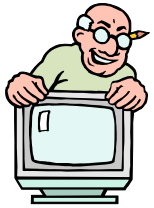
QuadAutoESP32 MKII
Hip Limits

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.

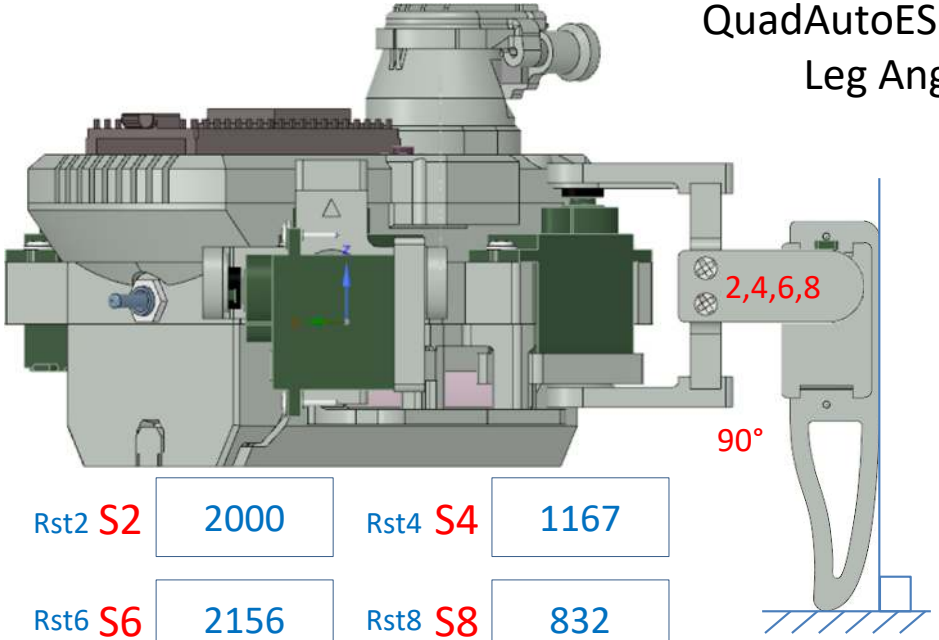




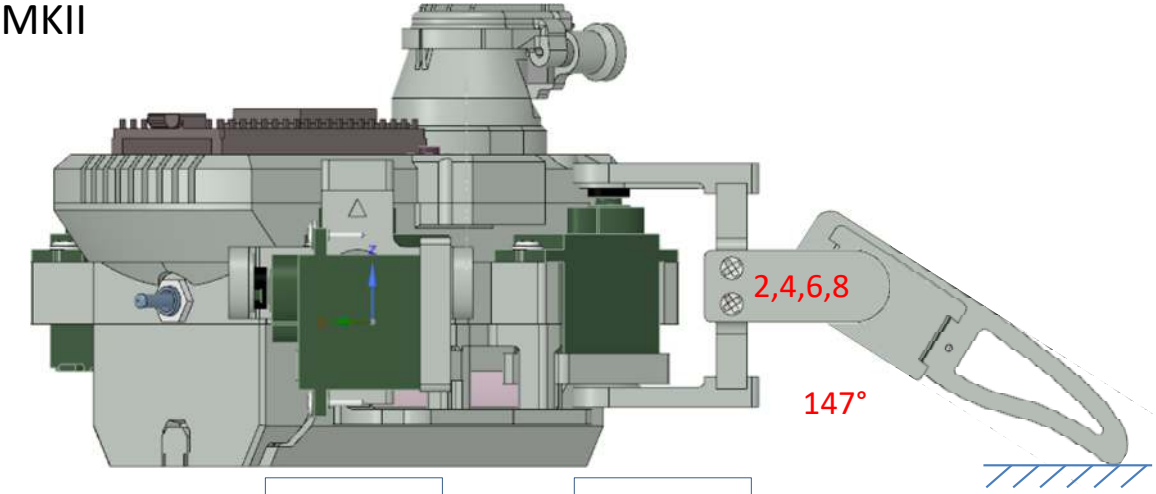
Code references are shown here in blue.
Enter your values to replace mine.

Issue: 1.0 Released: 27/11/2025 TechKnowTone

QuadAutoESP32 MKII
Leg Angles

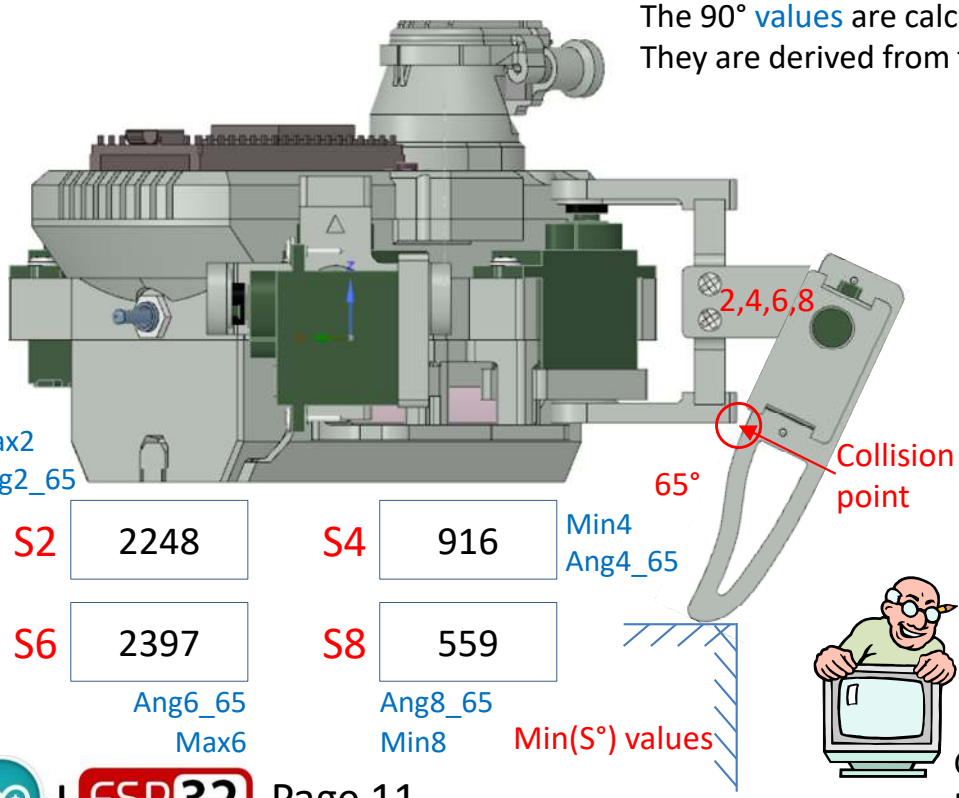


Rst2	S2	2000	Rst4 S4	1167	
Rst6	S6	2156	Rst8 S8	832	



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.

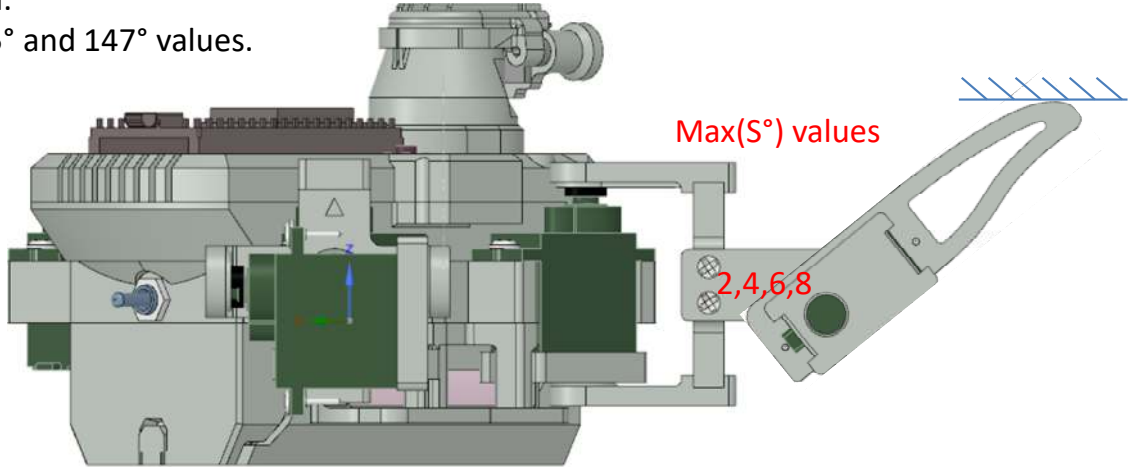


Max2
Ang2_65

S2	2248	S4	916	Min4 Ang4_65	
S6	2397	S8	559		

Ang6_65
Max6

Ang8_65
Min8

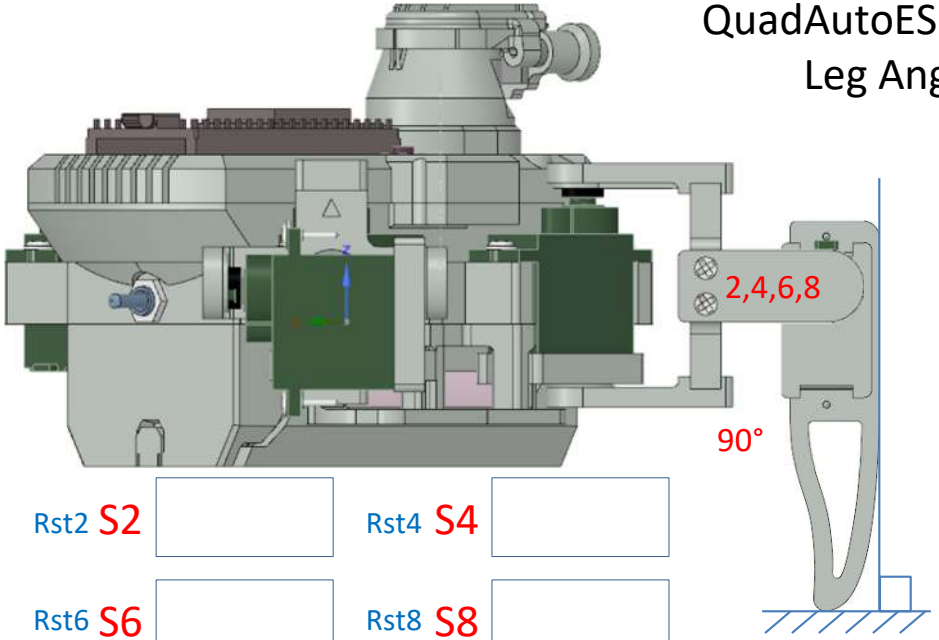


Min2	S2	508	S4	2400	Max4
Min6	S6	517	S8	2367	Max8

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

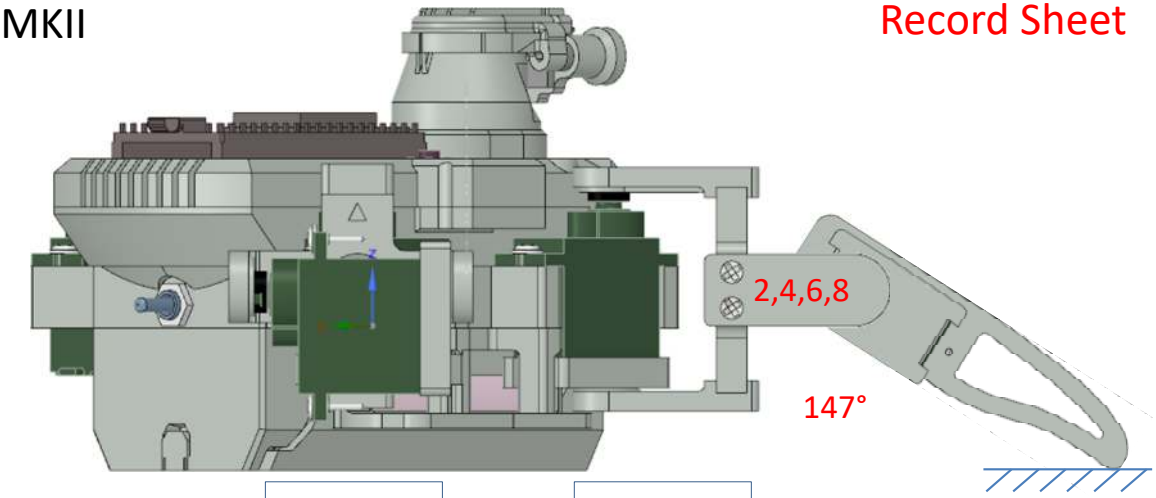
QuadAutoESP32 MKII
Leg Angles

Record Sheet



Rst2 S2 ____ Rst4 S4 ____

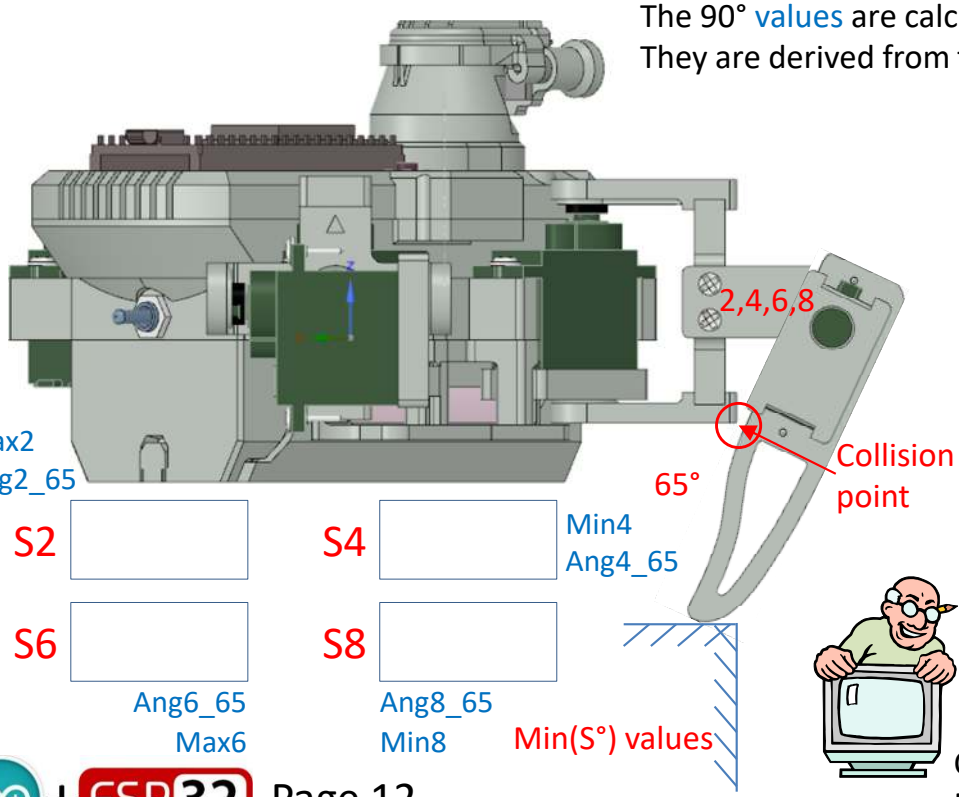
Rst6 S6 ____ Rst8 S8 ____



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.



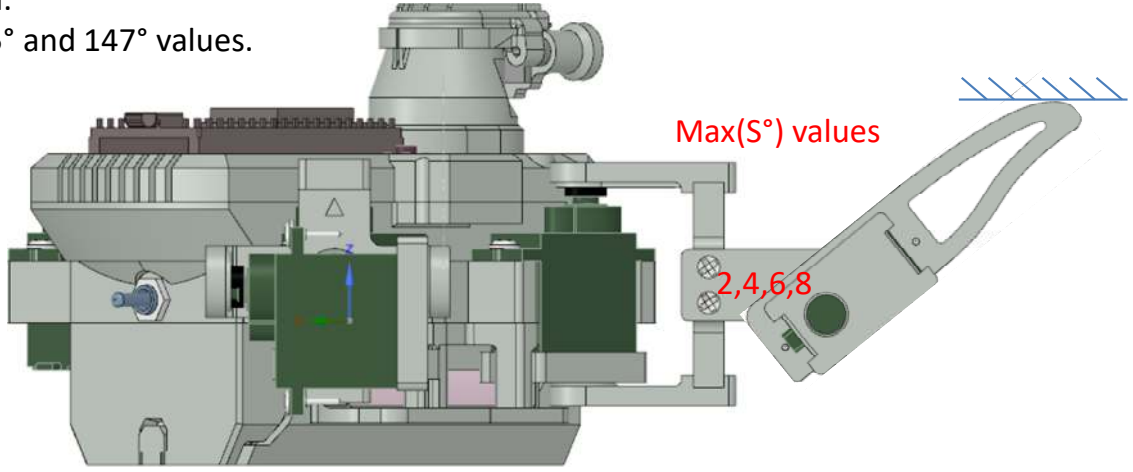
Max2
Ang2_65

S2 ____ S4 ____ Min4 Ang4_65

S6 ____ S8 ____

Ang6_65
Max6

Ang8_65
Min8

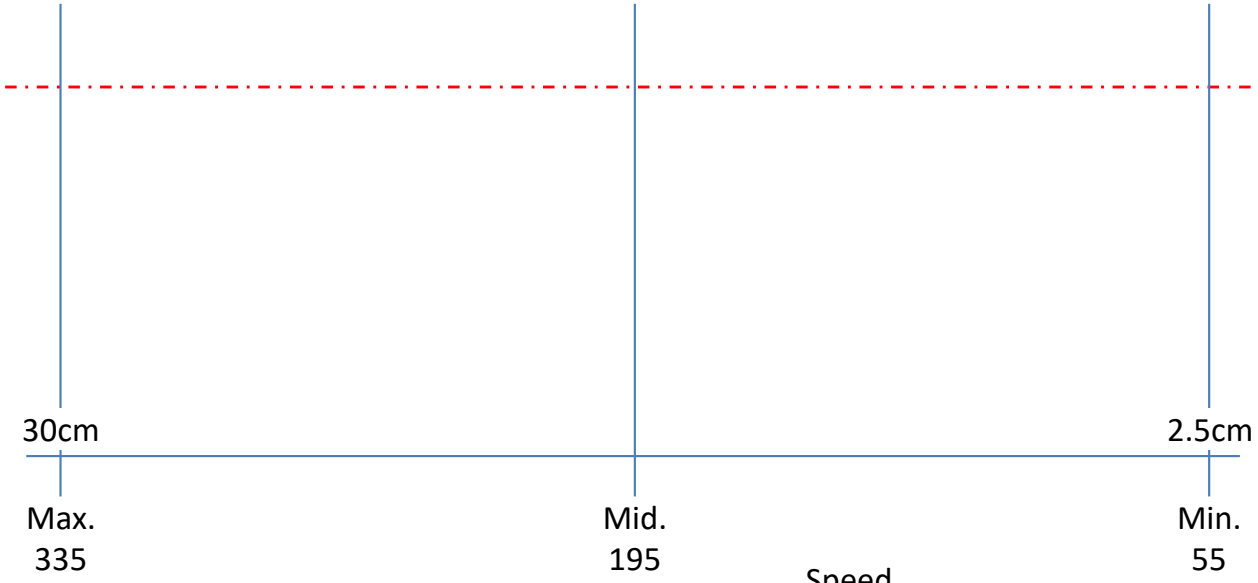


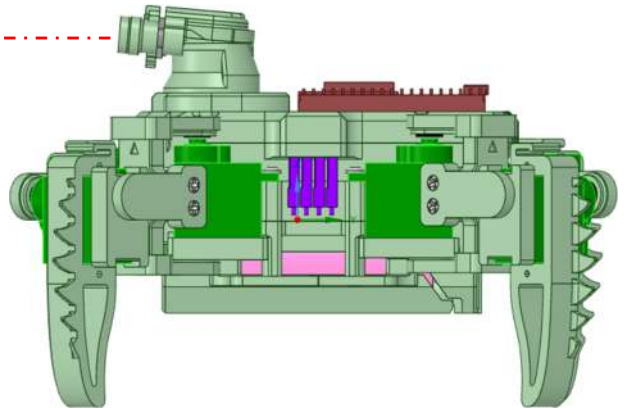
Min2 S2 ____ S4 ____ Max4

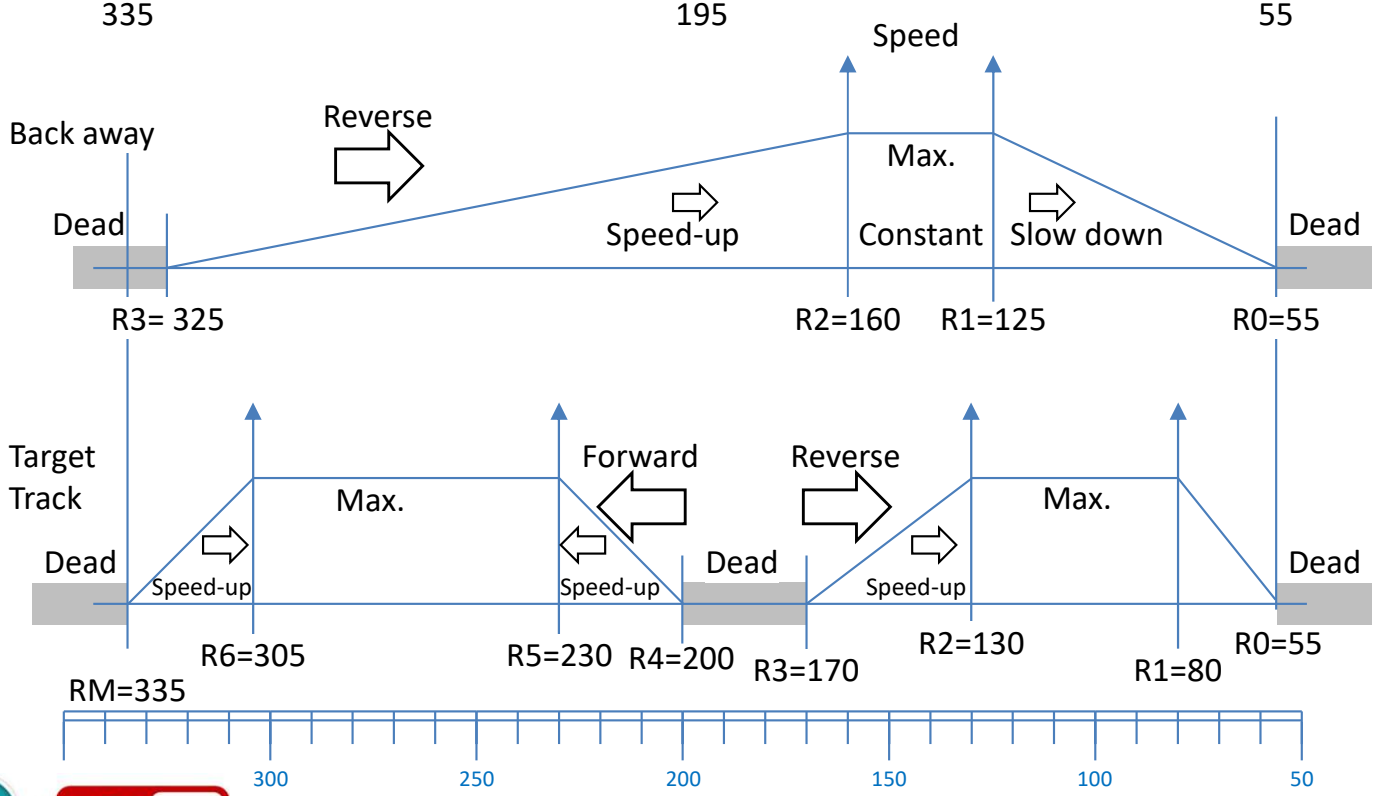
Min6 S6 ____ S8 ____ Max8

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

VL53L0X LTOF Sensor Ranges



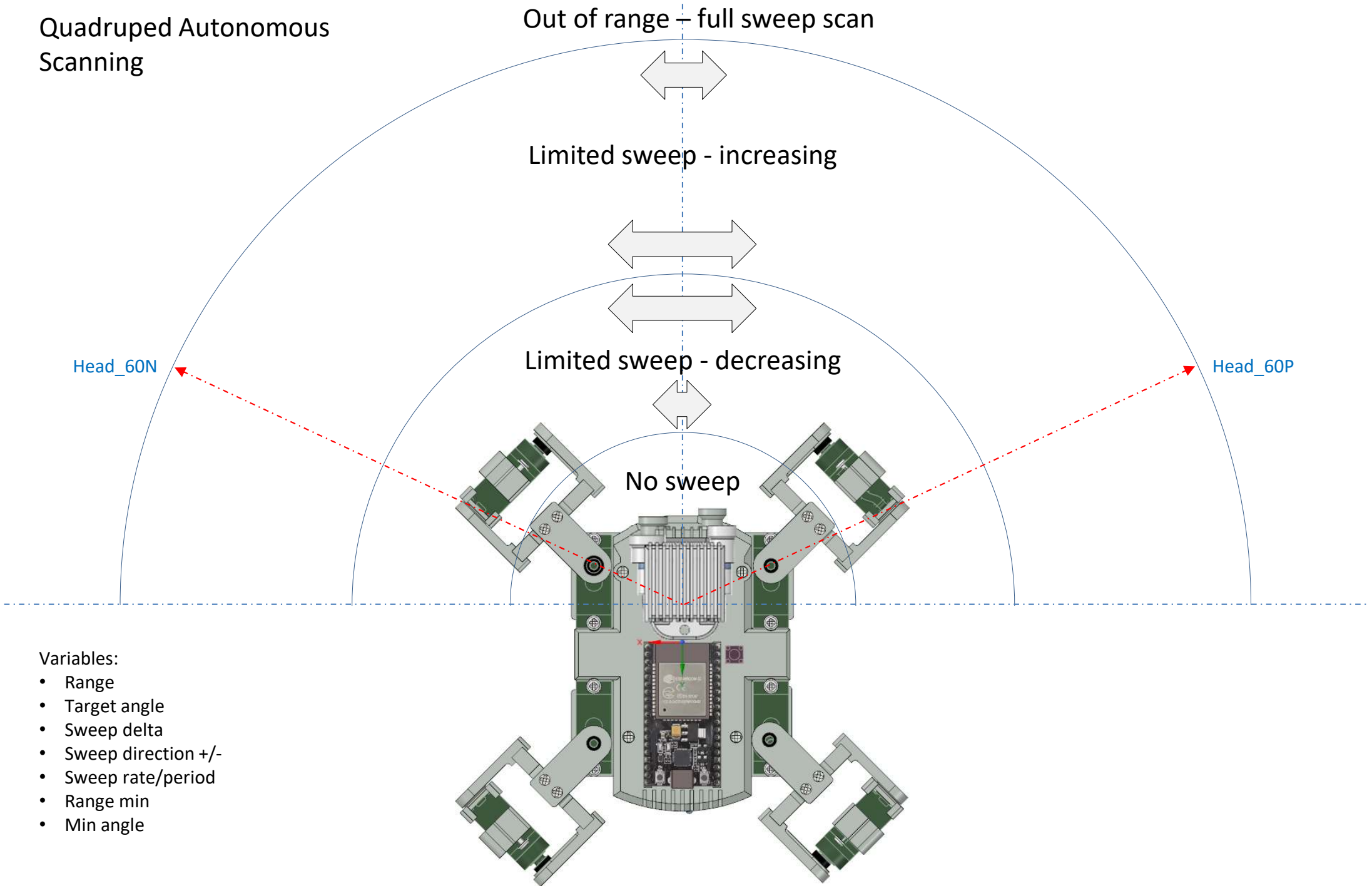




These are the values I determined from my LTOF sensor, and the speed maps I developed for the back away and target tracking functions.

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.

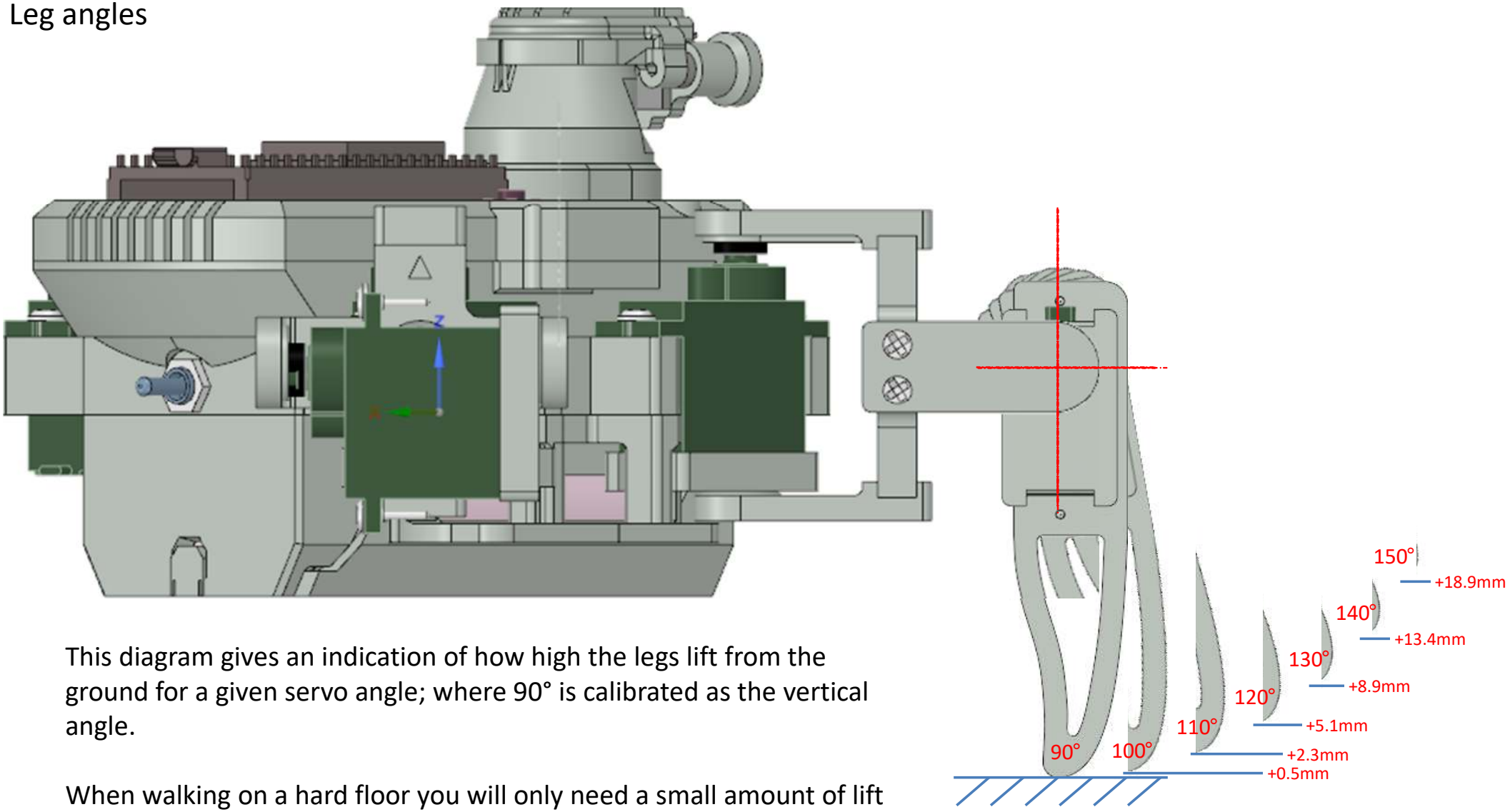
Quadruped Autonomous Scanning



Variables:

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

Leg angles



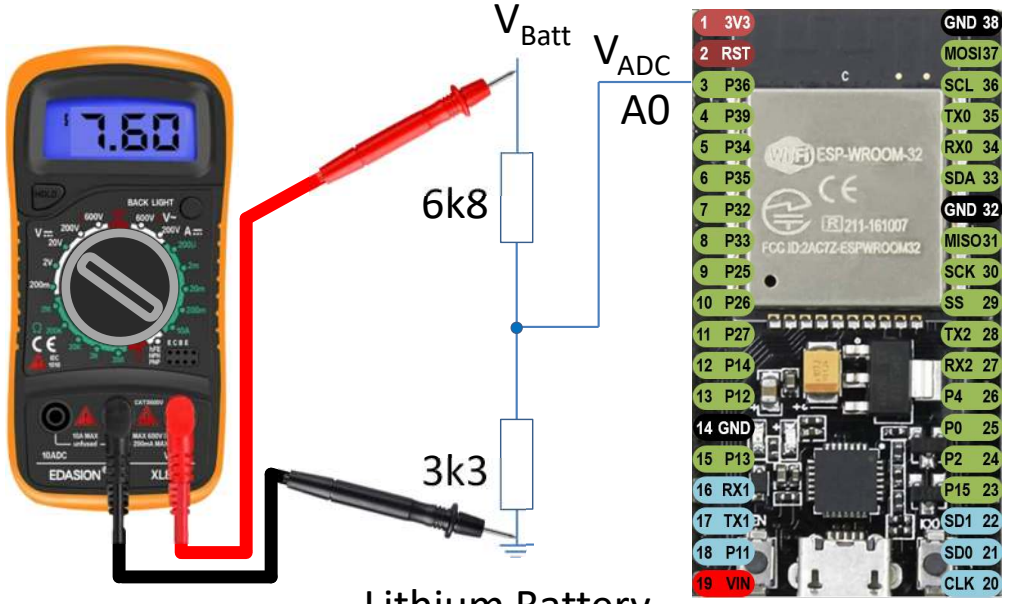
This diagram gives an indication of how high the legs lift from the ground for a given servo angle; where 90° is calibrated as the vertical angle.

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 from the infrared controller.

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.