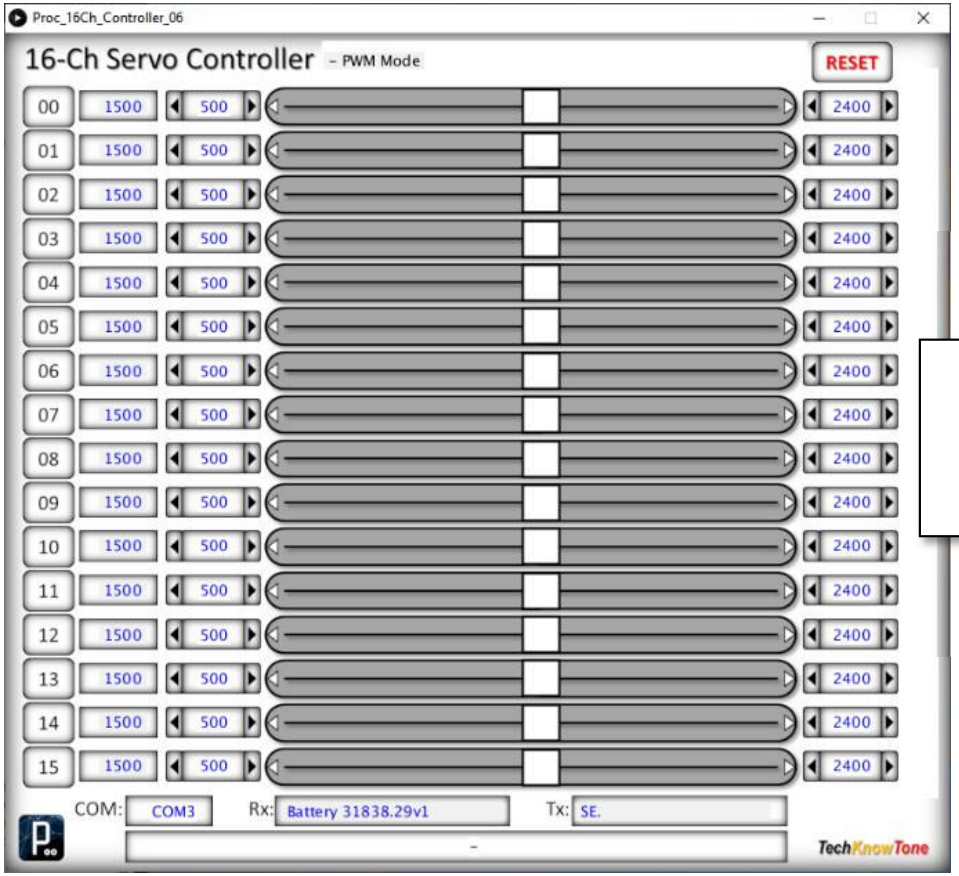
HexBot 2

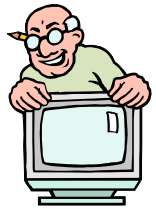
Servo Calibration



+ESP32

Issued: 08/12/2025 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.

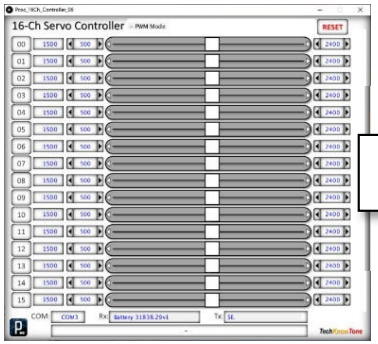
In case of fire: Get out and stay out. If a fire starts, leave immediately, and call the fire brigade. For low voltage Lithium batteries, water is a safe extinguisher.

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.

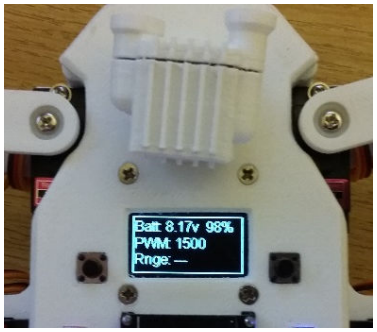
Overview

There are two servo calibration phases in creating a robot that walks and performs really well. ‘Course’ calibration is done during the initial assembly stage, which aims to locate the servo leavers at a good approximate starting point, making the second ‘Fine’ phase of calibration possible. We will use built in features of the code and a special Windows app to control the servos. You will need to print, assemble and use the test stands and gauges too, in order to follow the steps in this guide.



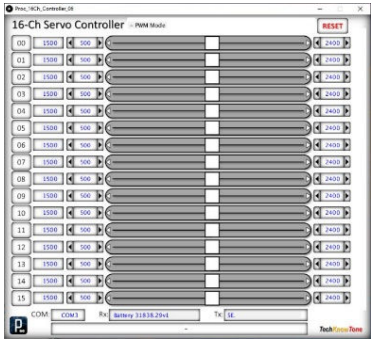
Course Calibration

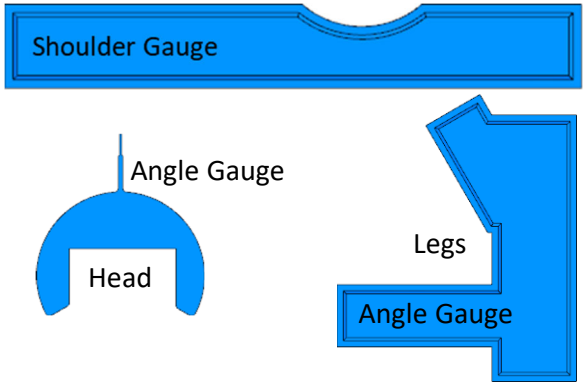
The first part of this document focuses on getting the servo leavers attached at a good approximate starting point. This assumes that the micro plate and body plate are fully wired, and connected together, and the ESP32 is loaded with code, but none of the leg servos have yet to be fitted. By default a TEST flag in the code is set to true, which forces the robot to display basic information, generate a simple coloured LED animation, and output 1500µs PWM to all 13 servos. A short press on the left-hand button switch SW0 will toggle the PWM between being fixed at 1500µs and having a small triangular waveform variation.



Fine Calibration

This applies to the second part of this document and aims to accurately determine critical PWM values, which need to be inserted into the code. To makes this process quite straight forward you will need to print off the custom angle gauges provided as .stl files. Once this is done, and the PWM values are inserted into the code, we can then use the map() function to convert angles of choice into calibrated PWM drive values.

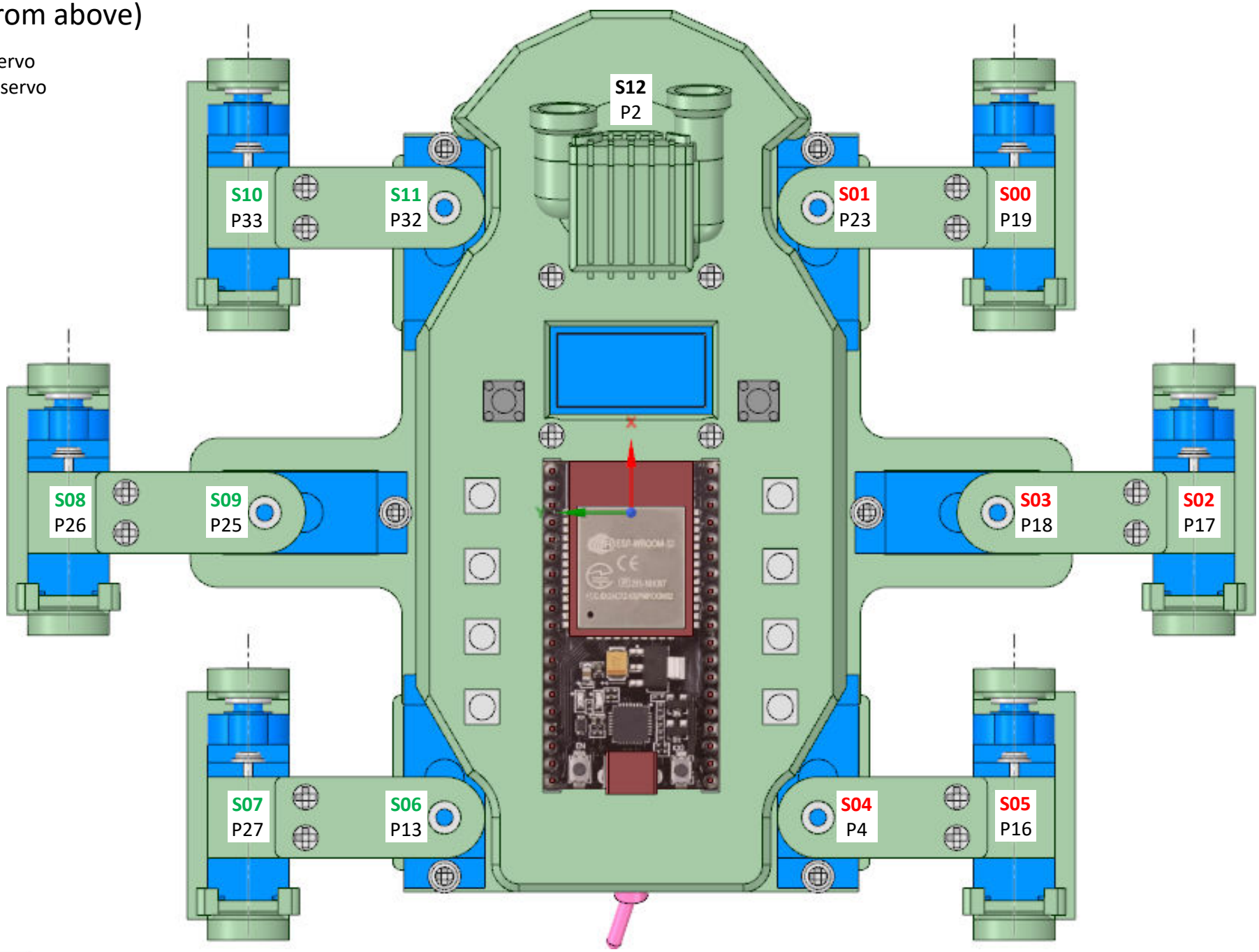




IMPORTANT: If servos are driven into stall conditions their motors draw excessive currents and heat up rapidly. Left in this condition for more than a few seconds can result in permanent damage to their motors, and even fire. Take particular care to avoid this.

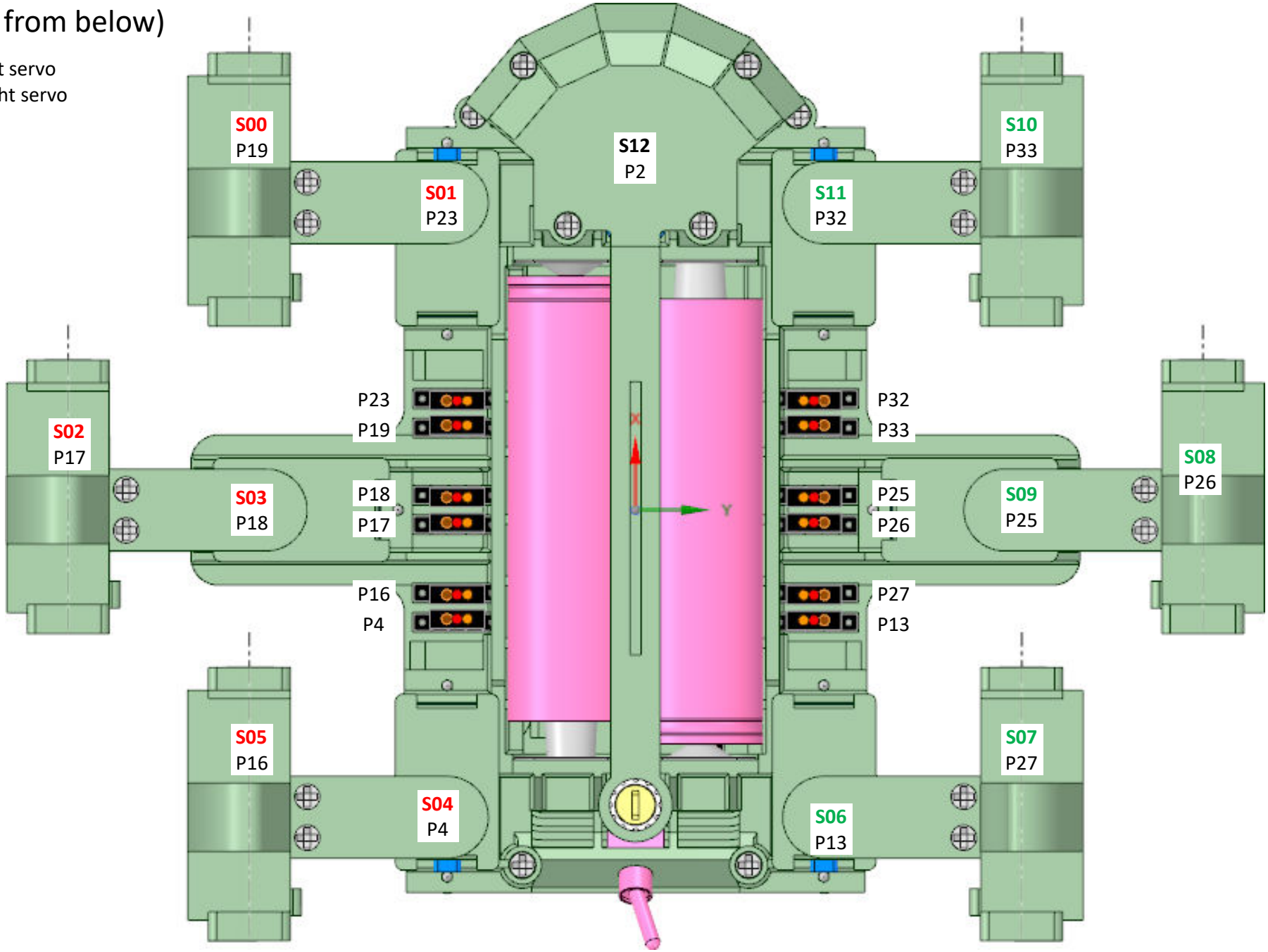
Servo to GPIO assignment
(viewed from above)

Snn Left servo
Snn Right servo



Servo to GPIO assignment
(viewed from below)

Snn Left servo
Snn Right servo



Head Angles

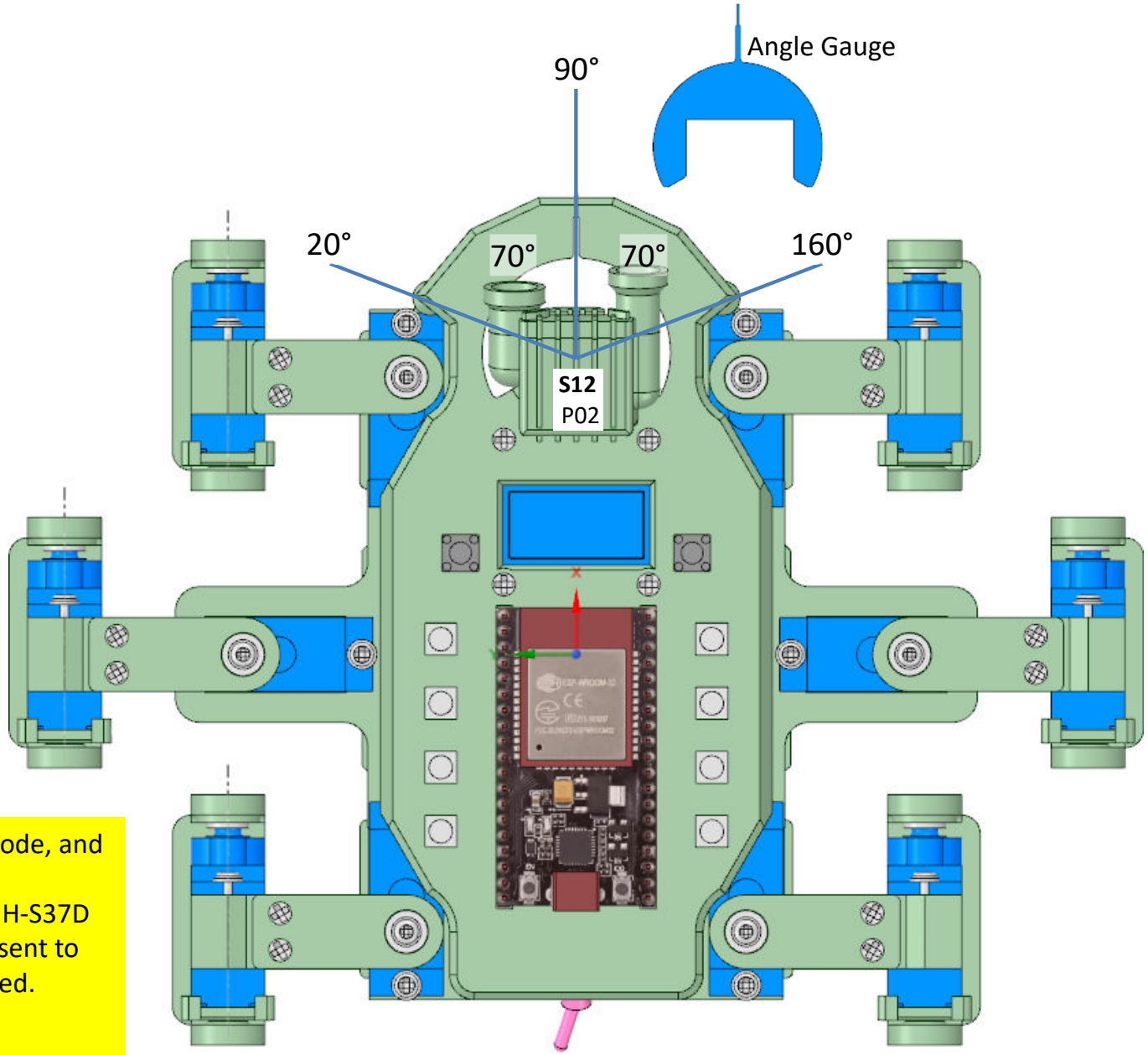
Place the robot on its stand. Use the PWM app to determine the values for the head servo S12 for each of the designated angles shown in the diagram. An angle pointer model is provided in the .STL files, which clips onto the base of the head.

These should be written down and entered into the code as servo constants:

S12	PWM	Code Ref.
20°	612	Head_20
90°	1398	Head_90
160°	2184	Head_160

Note that 90° represents the head facing forward. The calibration points gives the head +/-70° of movement.

The Angle Gauge clips onto the neck of the HexBot2



Note: Normally when you detach a servo motor in code, and remove its PWM signal, the servo will enter into an unpowered state, and can be turned by hand. The GH-S37D servo however appears to hold the last PWM value sent to it, and holds onto this position until power is removed.

So it can't be power-down in the usual manner.

Claw Crouch 69° Pinch Points
(all six legs)

Place the robot upside down on its stand. Use the PWM app to determine the values at which each of the six claws pinch against their respective link leaver. Use a small sliver of paper to determine this.

These should be written down and entered into the code as servo constants:

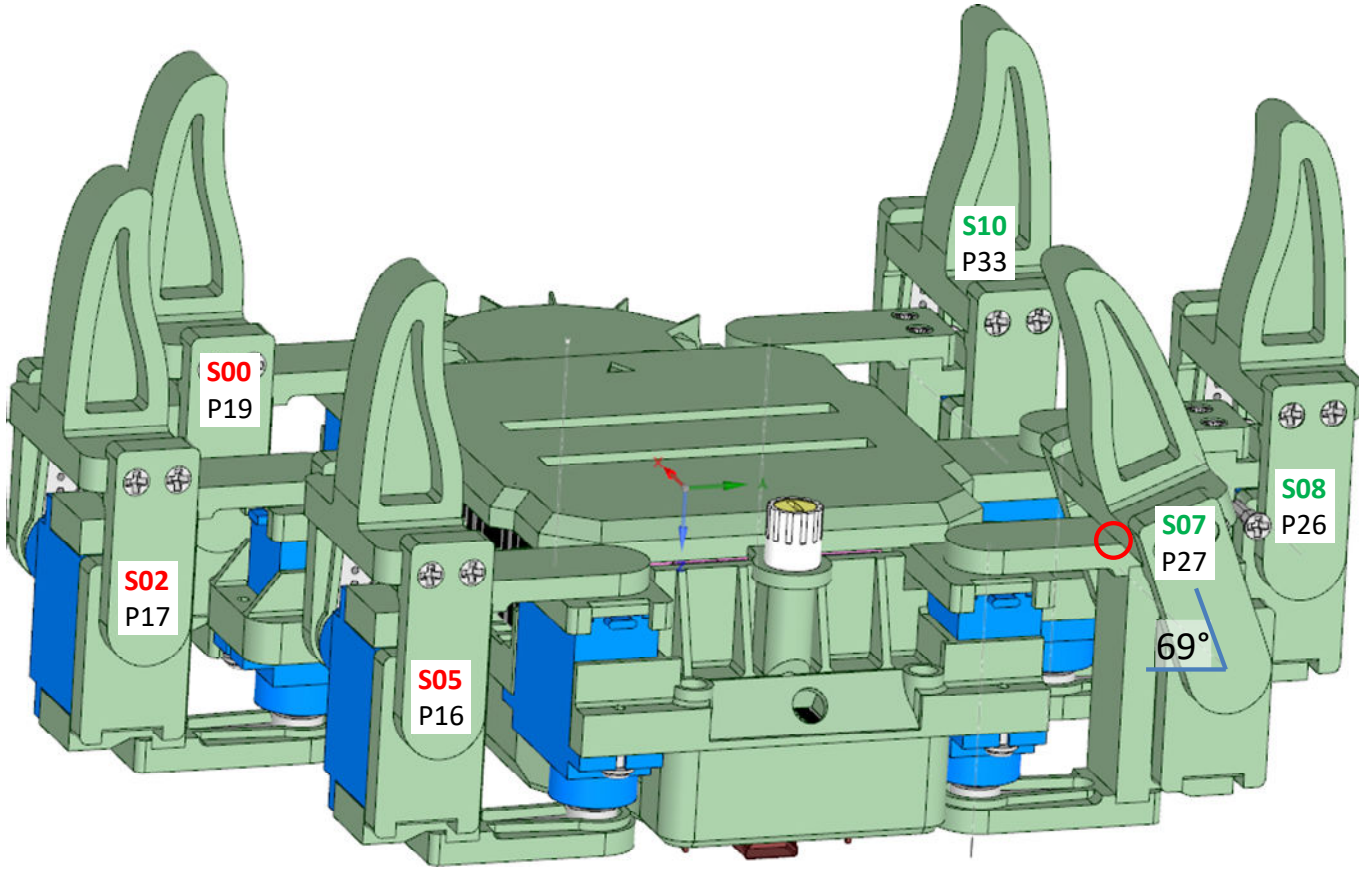
Left side:

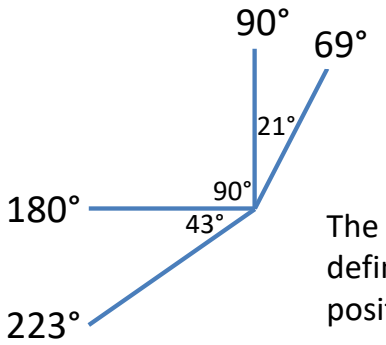
Servo	GPIO	PWM	Code Ref.
S07	P27	514	Ang7_69
S08	P26	607	Ang8_69
S10	P33	611	AngA_69

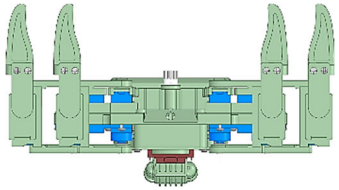
Right side:

Servo	GPIO	PWM	Code Ref.
S00	P19	2280	Ang0_69
S02	P17	2355	Ang2_69
S05	P16	2208	Ang5_69

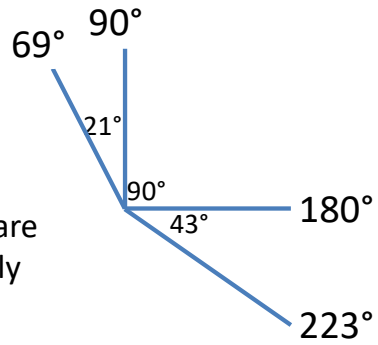
Note that the left-hand and right-hand servos rotate in different directions to cause this. Hence the values of the two sets are quite different.







The angles for the claws are defined are defined like this. So that there are only positive angles within the range.



Claw Vertical 90° Points
(all six legs)

Place the robot upside down on its upper stand. Use the PWM app to determine the values at which each of the six claws are vertical when touching a set square.

These should be written down and entered into the code as servo constants:

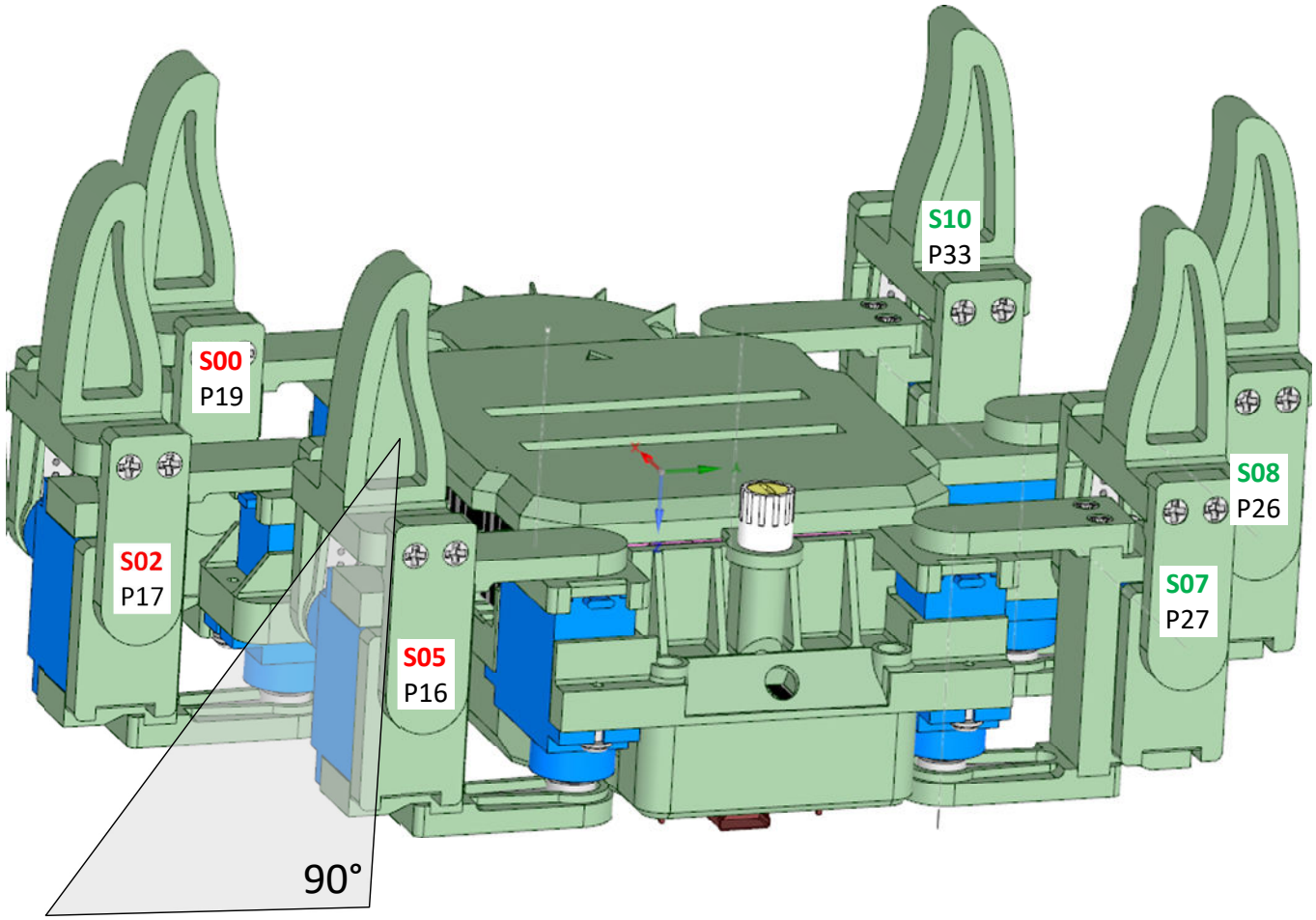
Left side:

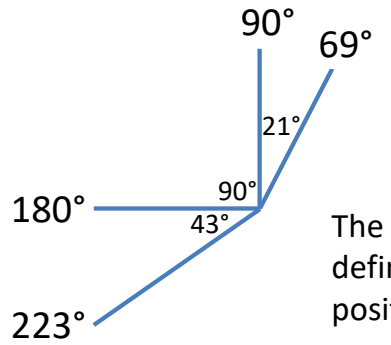
Servo	GPIO	PWM	Code Ref.
S07	P27	751	Ang7_90
S08	P26	784	Ang8_90
S10	P33	788	AngA_90

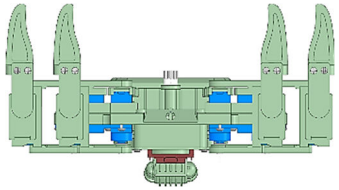
Right side:

Servo	GPIO	PWM	Code Ref.
S00	P19	2086	Ang0_90
S02	P17	2177	Ang2_90
S05	P16	2007	Ang5_90

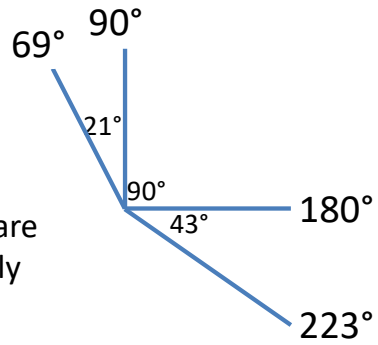
Note that the left-hand and right-hand servos rotate in different directions to cause this. Hence the values of the two sets are quite different.







The angles for the claws are defined are defined like this. So that there are only positive angles within the range.



Claw Raised 133° Pinch Points
(all six legs)

Place the robot the right way up on its stand. Use the PWM app to determine the values at which each of the six claws pinch against their respective bearing plates. Use the keyboard left and right arrow keys to achieve this.

The PWM values should be recorded and entered into the code as servo constants:

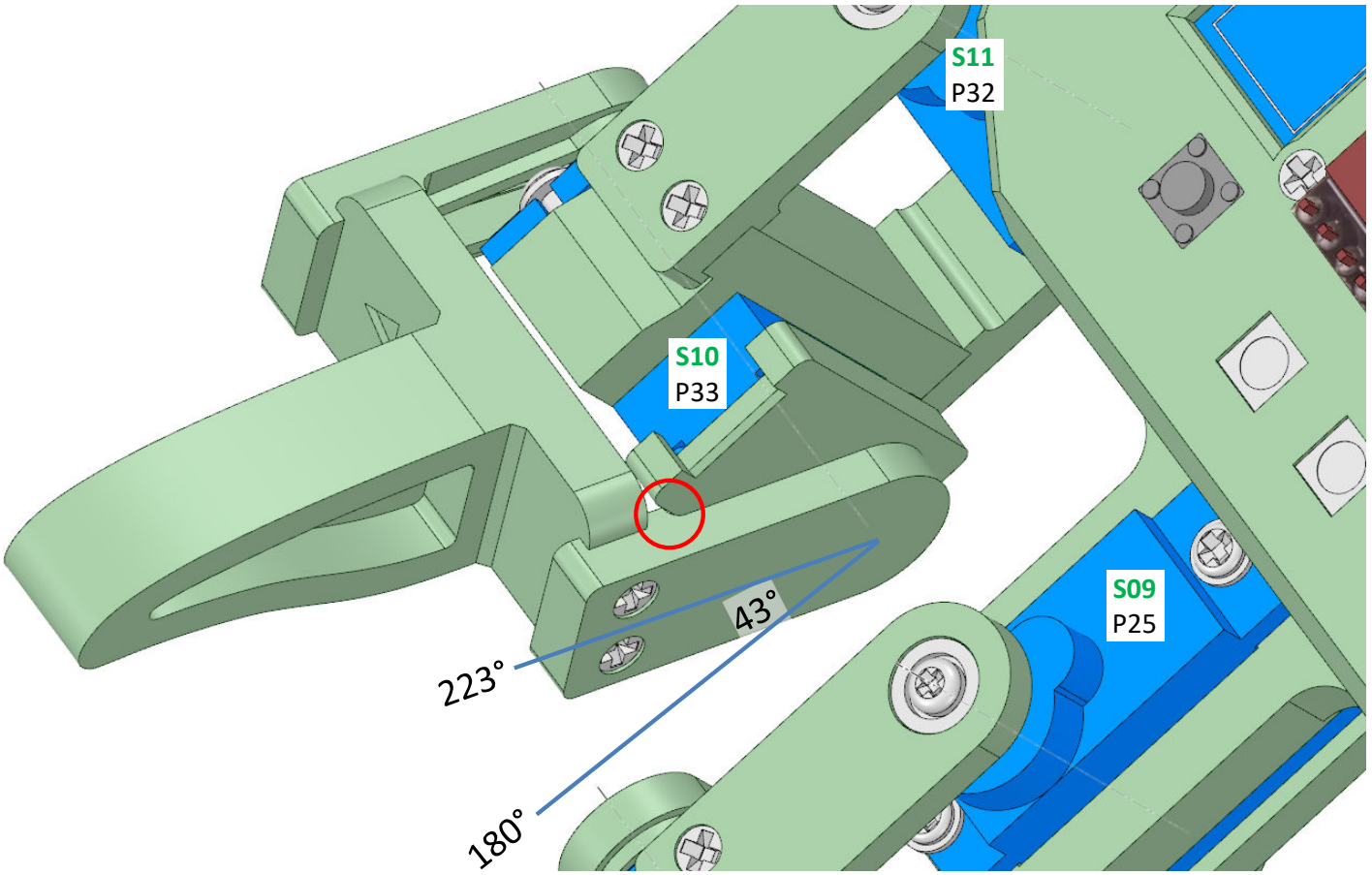
Left side:

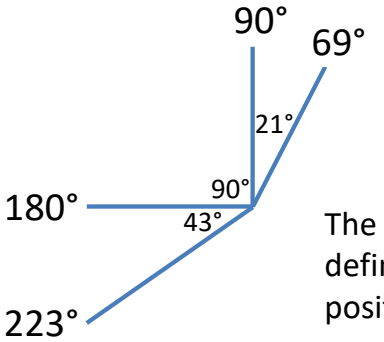
Servo	GPIO	PWM	Code Ref.
S07	P27	2128	Ang7_223
S08	P26	2150	Ang8_223
S10	P33	2153	AngA_223

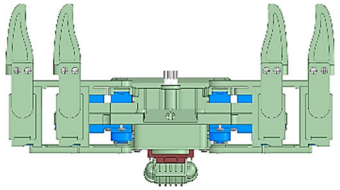
Right side:

Servo	GPIO	PWM	Code Ref.
S00	P19	713	Ang0_223
S02	P17	844	Ang2_223
S05	P16	708	Ang5_223

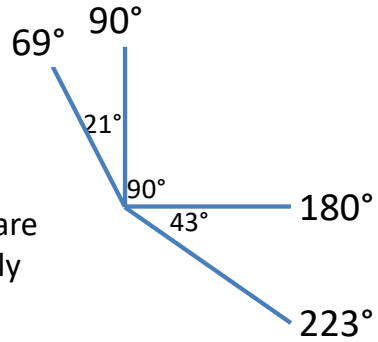
Note that the left-hand and right-hand servos rotate in different directions to cause this. Hence the values of the two sets are quite different.







The angles for the claws are defined are defined like this. So that there are only positive angles within the range.



Shoulder Angles
(front and rear, left/right)

Place the robot the right way up on its stand. Use the PWM app to determine the values at which each of the four front and rear shoulder servos are positioned as shown. To set the front leg 90 angles a shoulder gauge model is needed. Place it behind the neck of the head. Note some of these angles are pinch points.

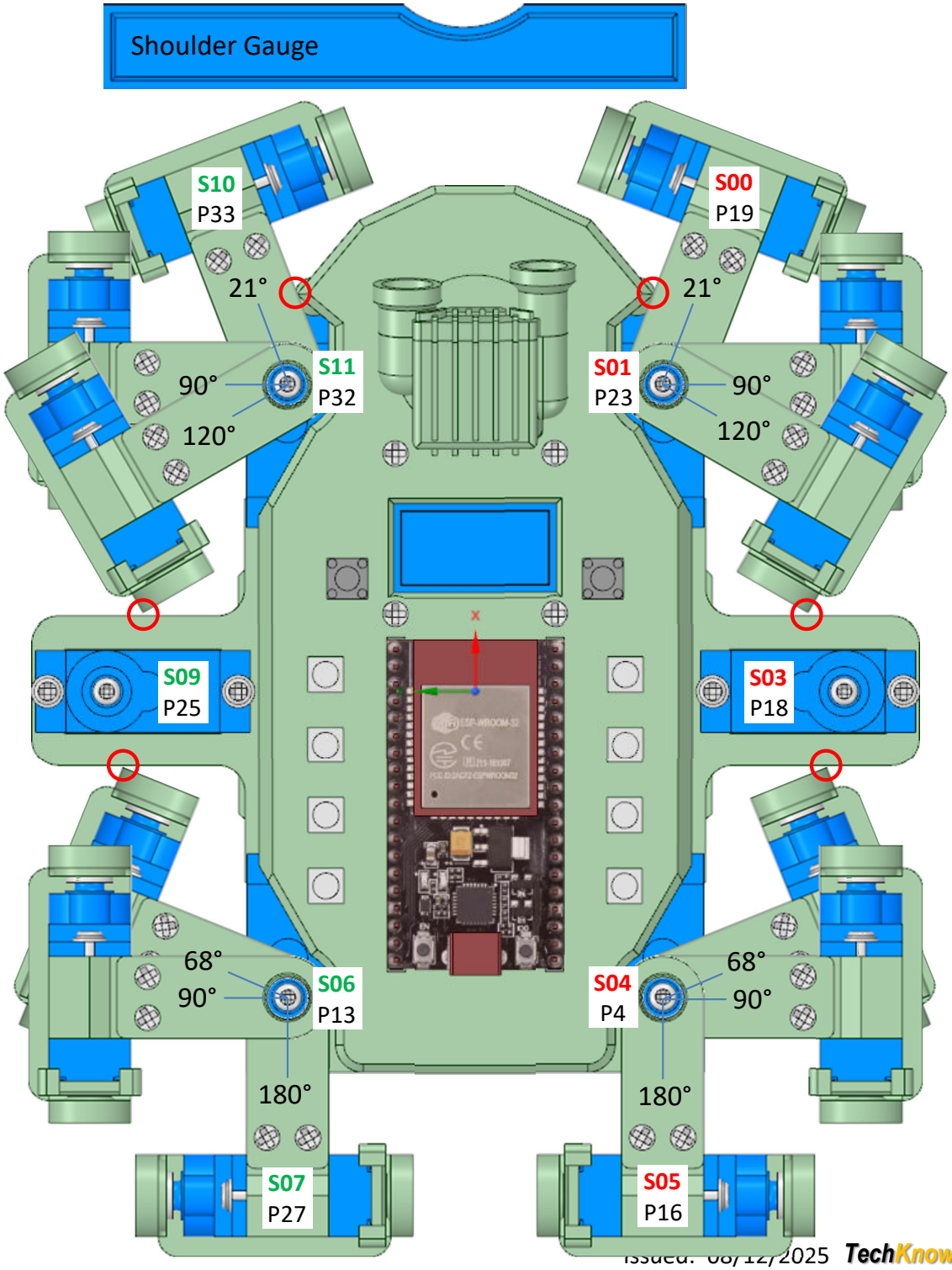
Left side:

Servo	GPIO	PWM	Code Ref.
S06	P13	1145	Ang6_68
		1394	Ang6_90
		2267	Ang6_180
S11	P32	858	AngB_21
		1546	AngB_90
		1917	AngB_120

Right side:

Servo	GPIO	PWM	Code Ref.
S01	P23	2066	Ang1_21
		1308	Ang1_90
		824	Ang1_120
S04	P04	2007	Ang4_68
		1734	Ang4_90
		830	Ang4_180

As the left-hand and right-hand servos rotate in different directions the values of the data sets are quite different.



Shoulder Angles
(centre, left/right)

Place the robot the right way up on its stand. Use the PWM app to determine the centre servo values for each of the two centre shoulders as shown. Note you will need print off and use the angle gauge model to do this.

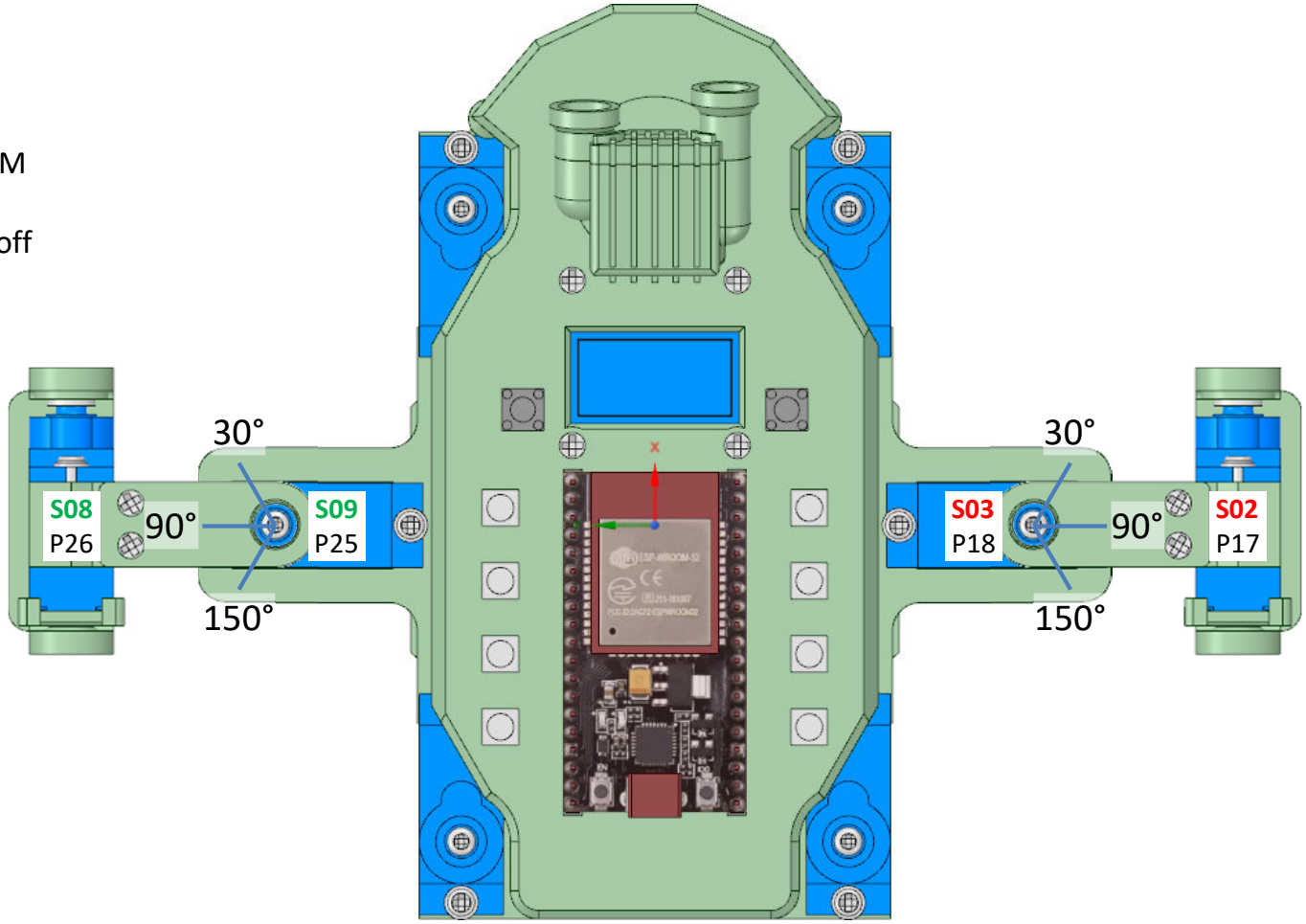
Left side:

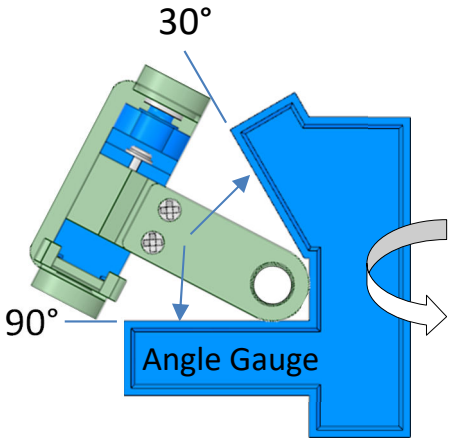
Servo	GPIO	PWM	Code Ref.
S09	P25	878	Ang9_30
		1543	Ang6_90
		2103	Ang6_150

Right side:

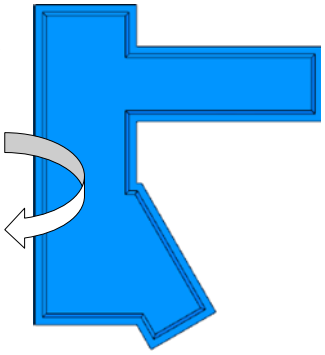
Servo	GPIO	PWM	Code Ref.
S03	P18	2084	Ang3_30
		1490	Ang3_90
		839	Ang3_150

As the left-hand and right-hand servos rotate in different directions the values of the data sets are quite different. This gives the leg a swing of +/-60° which is more than it needs for walking, but may prove useful in other manouvres.





Place the angle gauge against the side of the robot and move the leg shoulder until it is parallel with one of the gauges 30° edge. Flip the gauge over to set the other 150° angle.

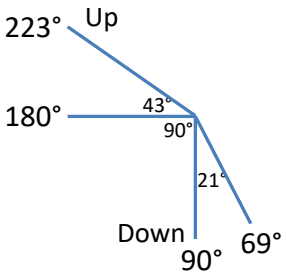


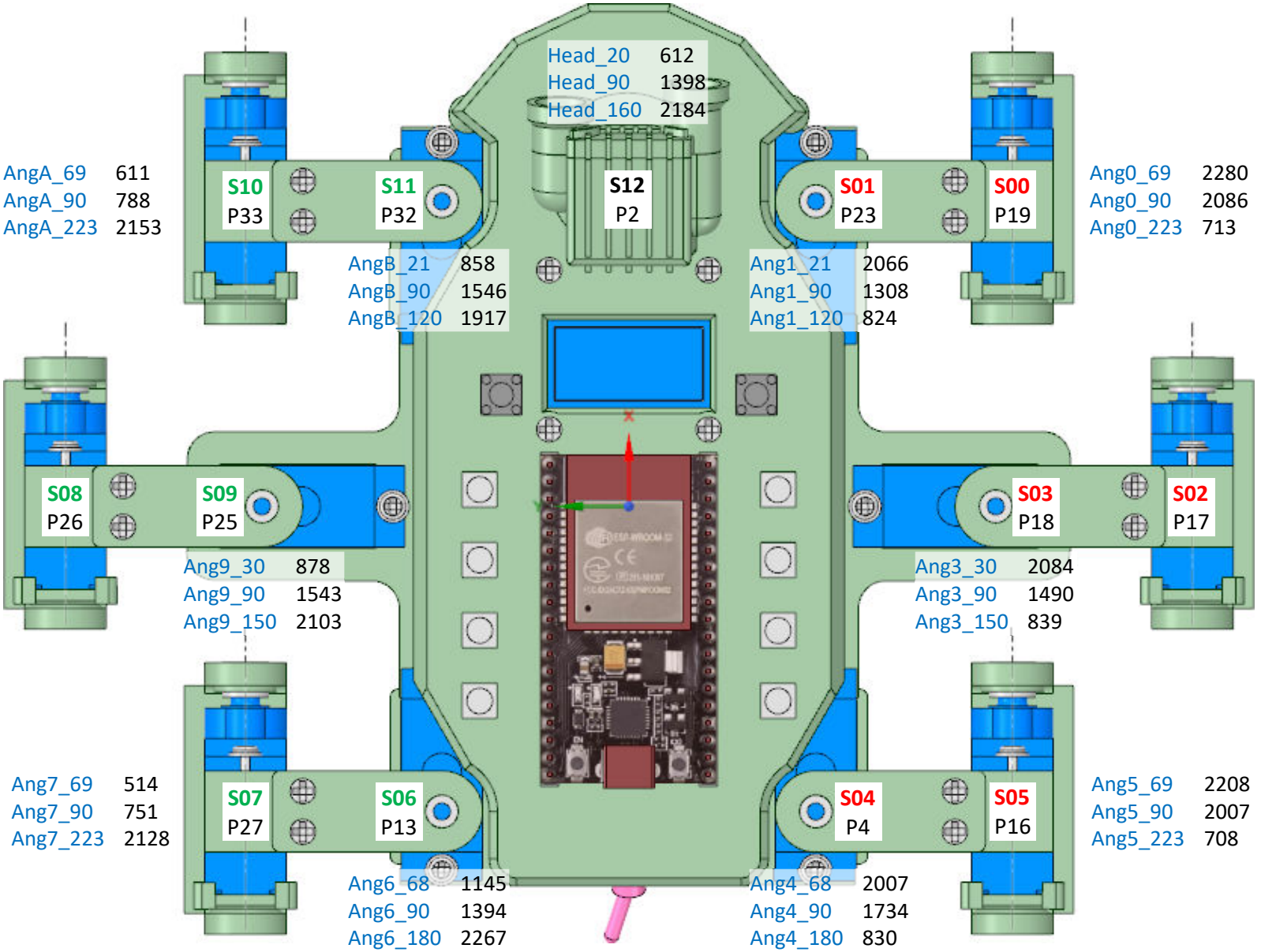
All servo to code assignment PWM values

(viewed from above)

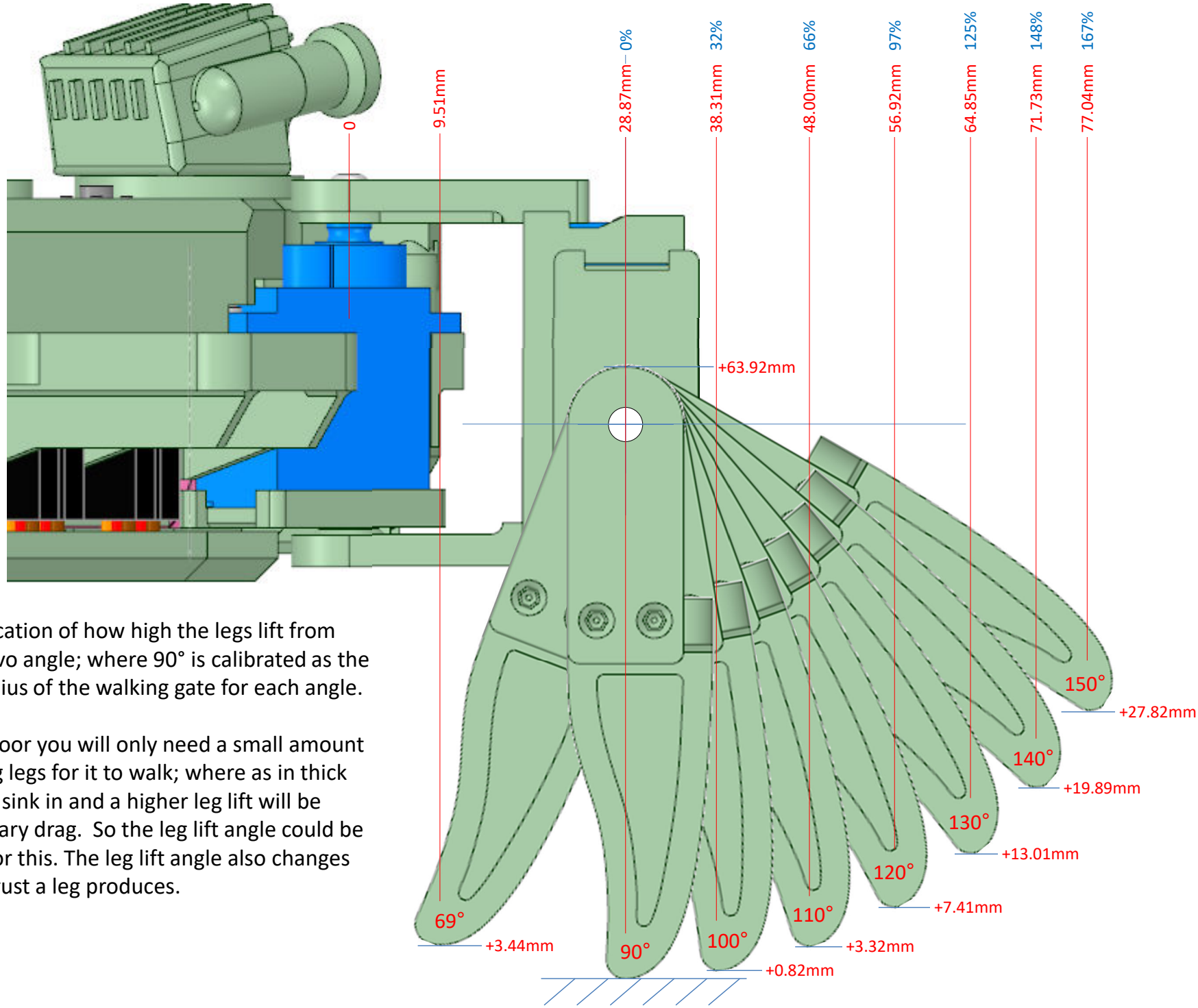
Snn Left servo

Snn Right servo





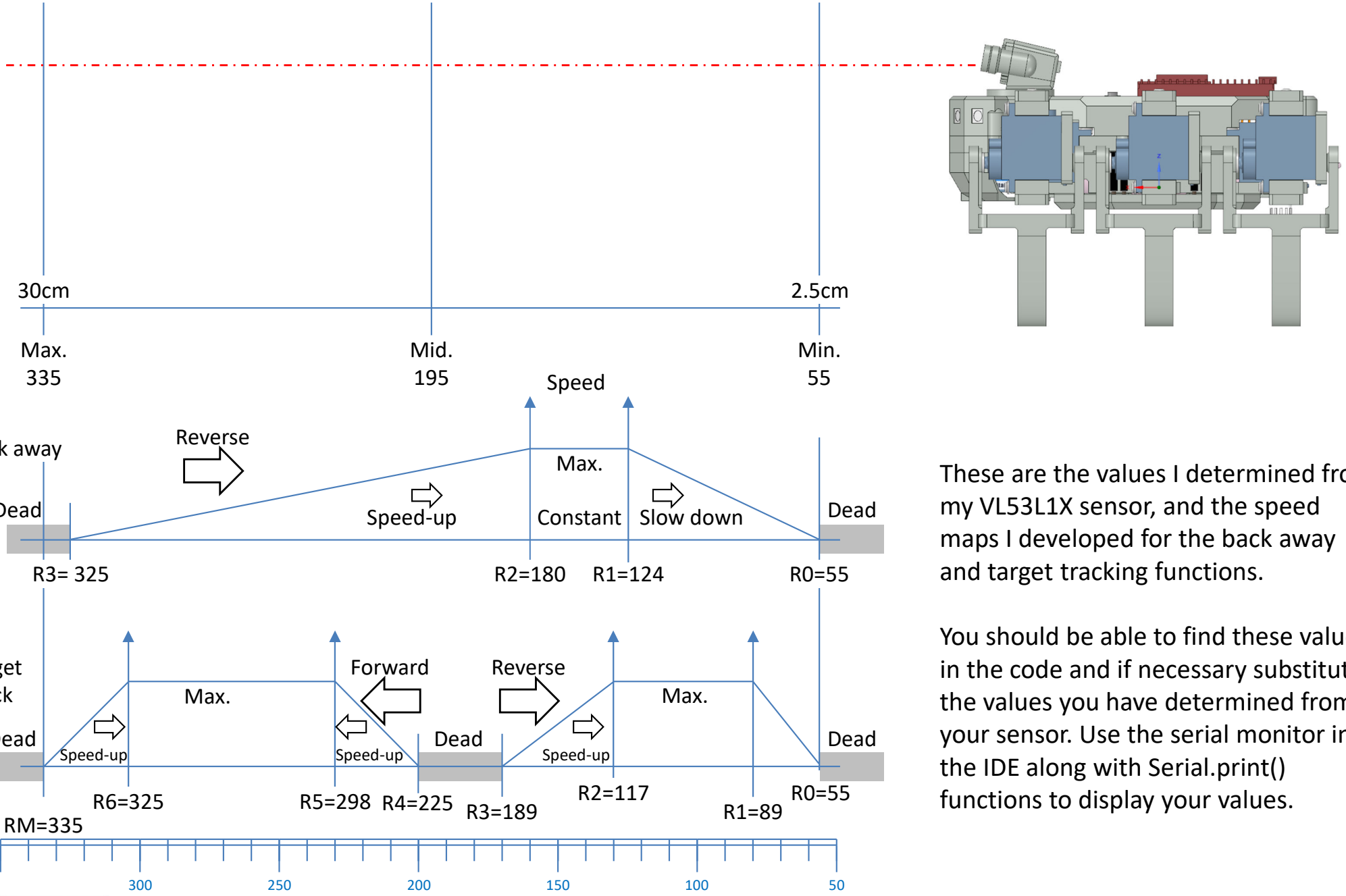
Leg Angles v Lift



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. Plus the radius of the walking gate for each 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. So the leg lift angle could be adjusted to compensate for this. The leg lift angle also changes the amount of forward thrust a leg produces.

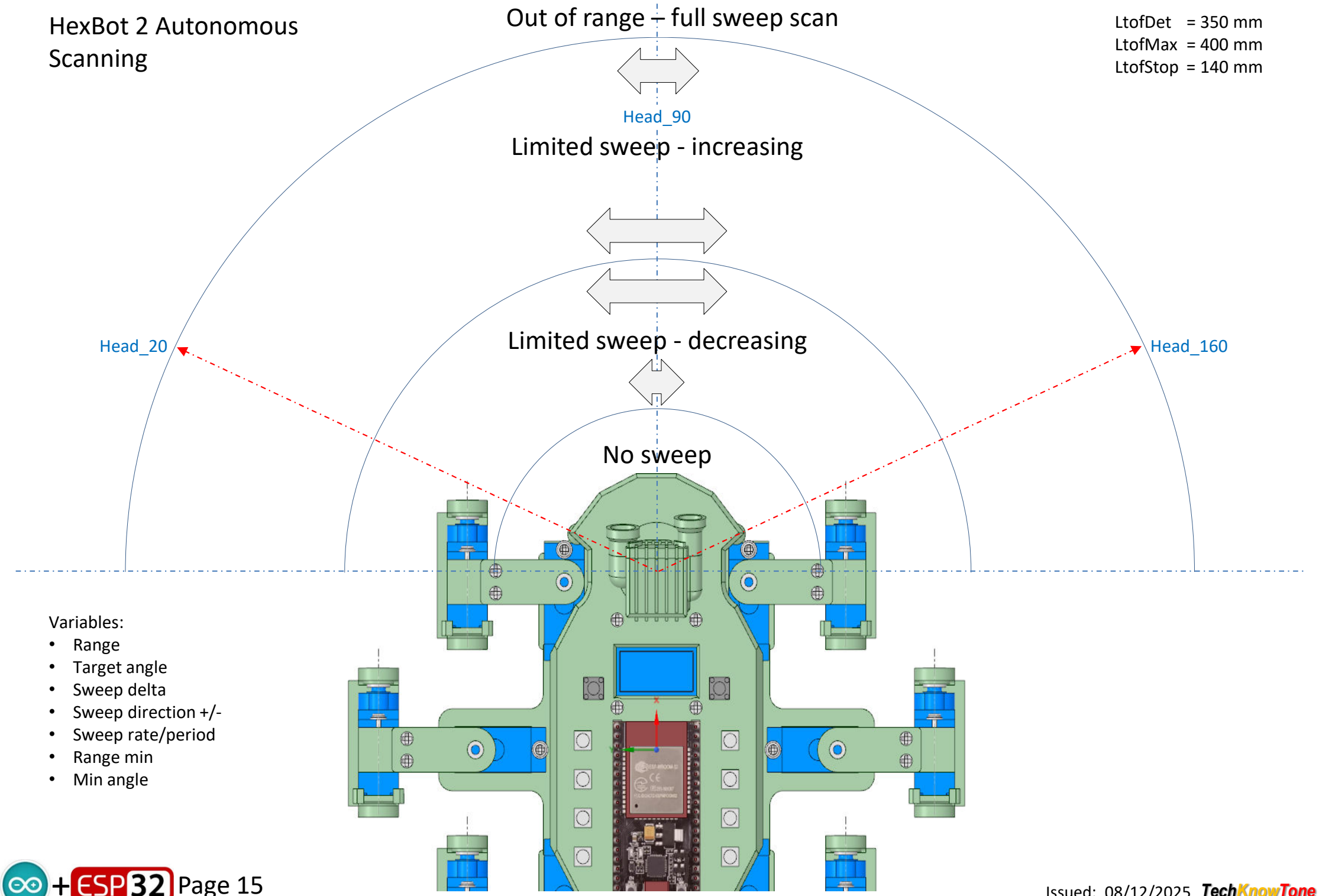
VL53L1X Fixed/Rotating Sensor



These are the values I determined from my VL53L1X 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.

HexBot 2 Autonomous Scanning



Variables:

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

Battery Voltage Health Monitoring

See 18650 discharge curve obtained from the internet. In this analysis both batteries are identical and connected in series,
Assume fully charged batteries max voltage is V_{BM} >= 8.2v max
I measured my rechargeable PP3 at 8.65v when connected and ON.
Set battery warning point at V_B = 7.00v
Set battery critical point at V_{BC} = 6.60v

ESP32 is powered from batteries connected to V_{in}.
3.3v at VADC == 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 404.85
Using a Multimeter I determined the conversion factor needed to be reduced to 383.9 to display voltage correctly.

MAX: V_M = 8.2v, gives A0 = 3148 on ADC (V_M* 383.9)

WARNING: V_B = 7.0v, gives A0 = 2687on ADC (V_B* 383.9)

CRITICAL: V_{BC} = 6.6v, gives A0 = 2534 on ADC (V_{BC}* 383.9)

The code will sample the battery voltage on power-up to ensure it is sufficient, then at every 40ms interval, calculating an average (1/20) 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 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.