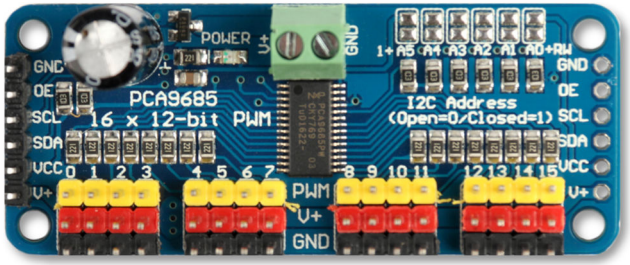
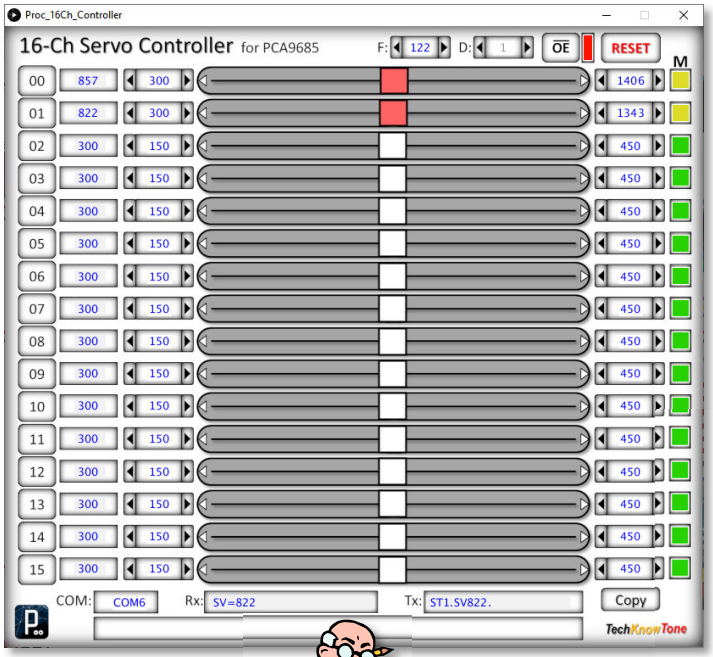
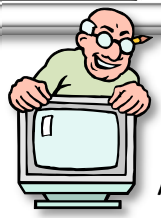
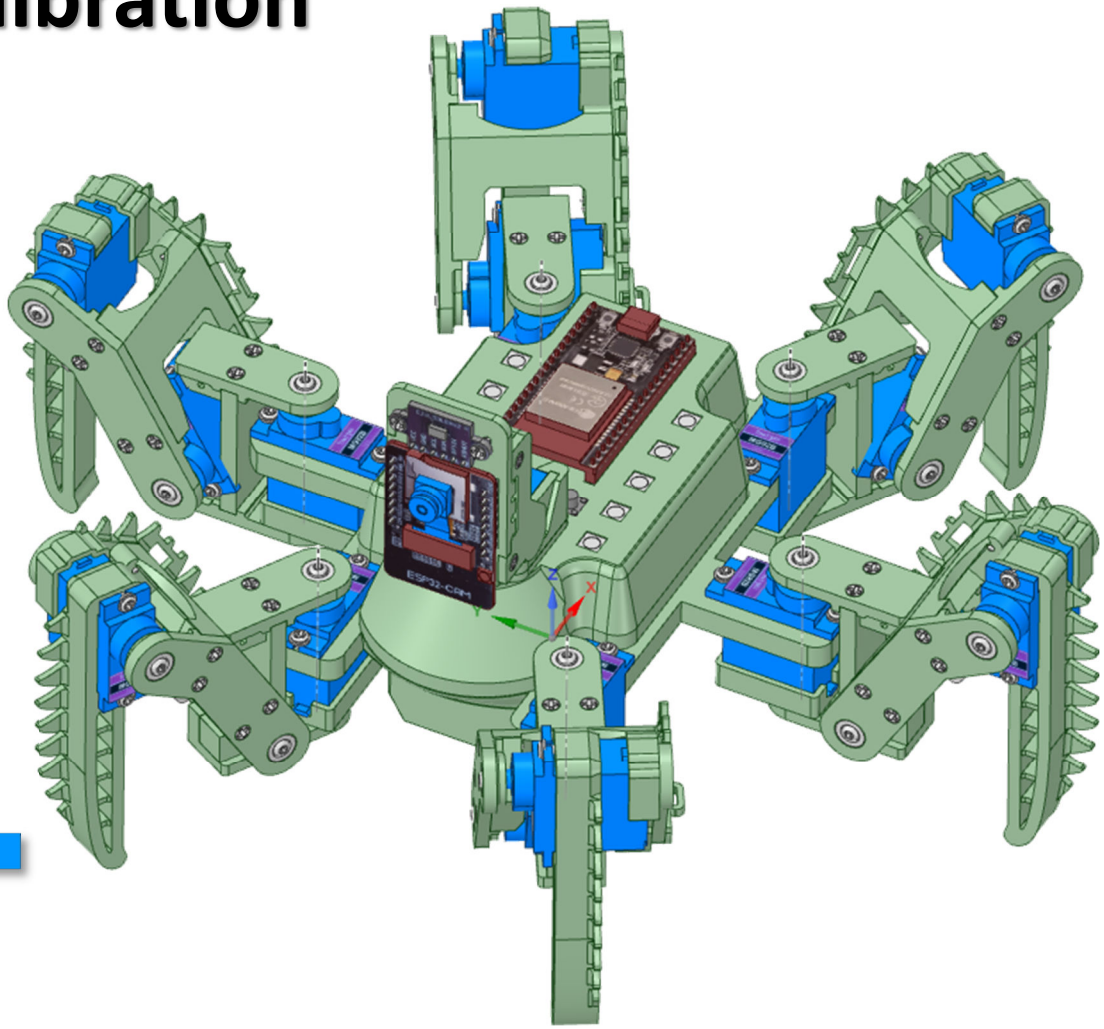
SpidaBot

Servo Calibration









+ ESP32

An important process, that will govern your robots performance.

Issue: 1.2 Released: 10/05/2025 TechKnowTone

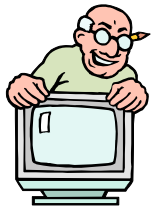
Why do we need to calibrate the servos?

- No two servos are the same.
- Servos can be damaged if not setup correctly.
- Course calibration must be performed during the assembly process.
- This sets approximate positions for the leaver arms.
- A servo drive shaft has 20 splined teeth == 18° (best fit +/-9°)
- Course calibration ensures servos are within mechanical range/limits.
- Fine calibration determines min/max robot physical limits.
- The ESP32 C++ code needs limit values in order to work accurately.
- Hence, all SpidaBots have a unique set of calibrated PWM code values.
- No two SpidaBots would ever be exactly the same.
- Code limits and default values would be different for each one.
- Once calibrated we can use angles, as common values, not PWM.

Servo calibration is performed in three stages:

- Course ensures mechanical parts are assembled correctly.
- Fine calibration, performed during testing, for accurate movement.

- Repeat this process for a given servo if it is ever replaced.

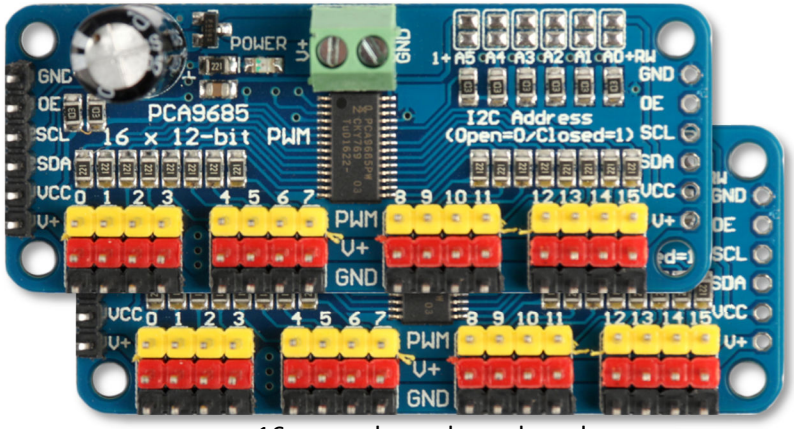


Only use genuine Tower Pro servos, for best performance.

Issue: 1.2 Released: 10/05/2025 TechKnowTone

Servo Testing

The SpidaBot employs 20 servos, therefore needing two PCA9685 controllers in tandem to provide the PWM control signals. The Adafruit_PWMServoDriver.h library, used in this project, enables you to set the base clock from which all of the PWM signals are derived. The base clock is passed into a hardware counter, which has a range of 4096 counts. By setting a start and end count values, you define when a given PWM signal will be driven HIGH and LOW. The higher the base clock frequency, the greater the control we have over the PWM pulse width. In this project we will use a PWM frequency of 125 Hz.



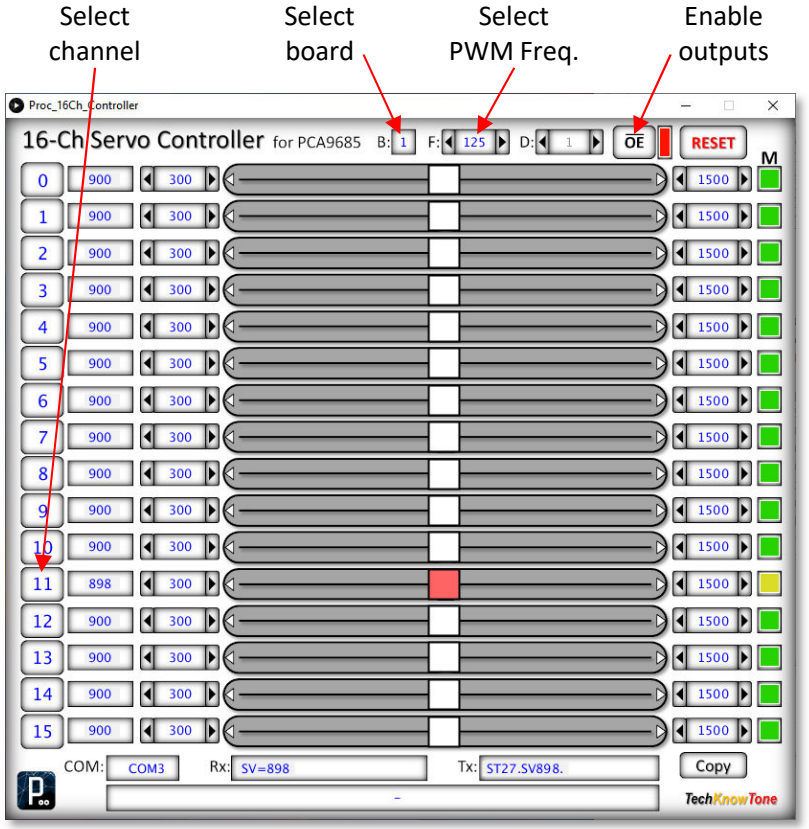
16 servo channels per board

I have created a Windows app which, when used with an ESP32 micro and the SpidaBot code, enables you to control up to four PCA9685 boards. Selected servos will respond to the slider settings, so you can determine what PWM values are required to set the servo control arms to a particular position.

This app is used for both course settings, during the build process, and the final fine adjustment settings, once all of the legs have been attached to the SpidaBot robot.

You can either wait until the robot build process has reached the point where servos are being attached, or create a prototype lash-up circuit to try out the app and PCA9685 boards on their own.

The app can be used with other projects employing servos, provided that you use the code in SpidaBot, which decodes and implements the serial commands.



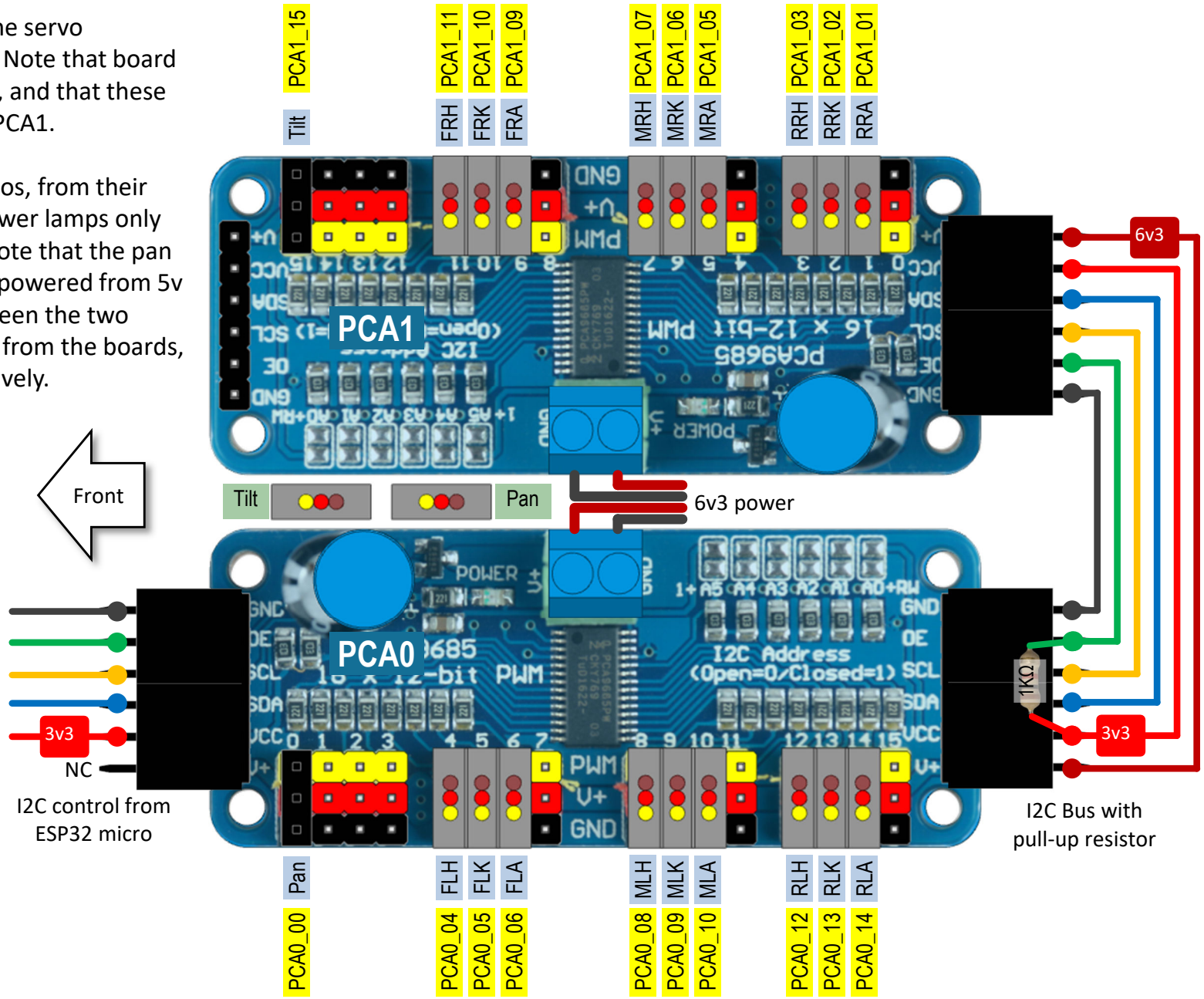
Issue: 1.2 Released: 10/05/2025 TechKnowTone

Servo Channel Assignments

Viewed from above, this diagram shows the servo assignments for the two PCA9685 boards. Note that board PCA0 is fed I2C bus signals from the micro, and that these signals are then passed through to board PCA1.

The boards receive 6v3 power for the servos, from their centre screw terminals. However, their power lamps only light when there is 3v3 on the VCC pins. Note that the pan and tilt micro servos for the ESP-CAM are powered from 5v using separate 3 pin strips, mounted between the two boards. But their PWM signals are still fed from the boards, from PCA0_00 and PCA1_15 pins, respectively.

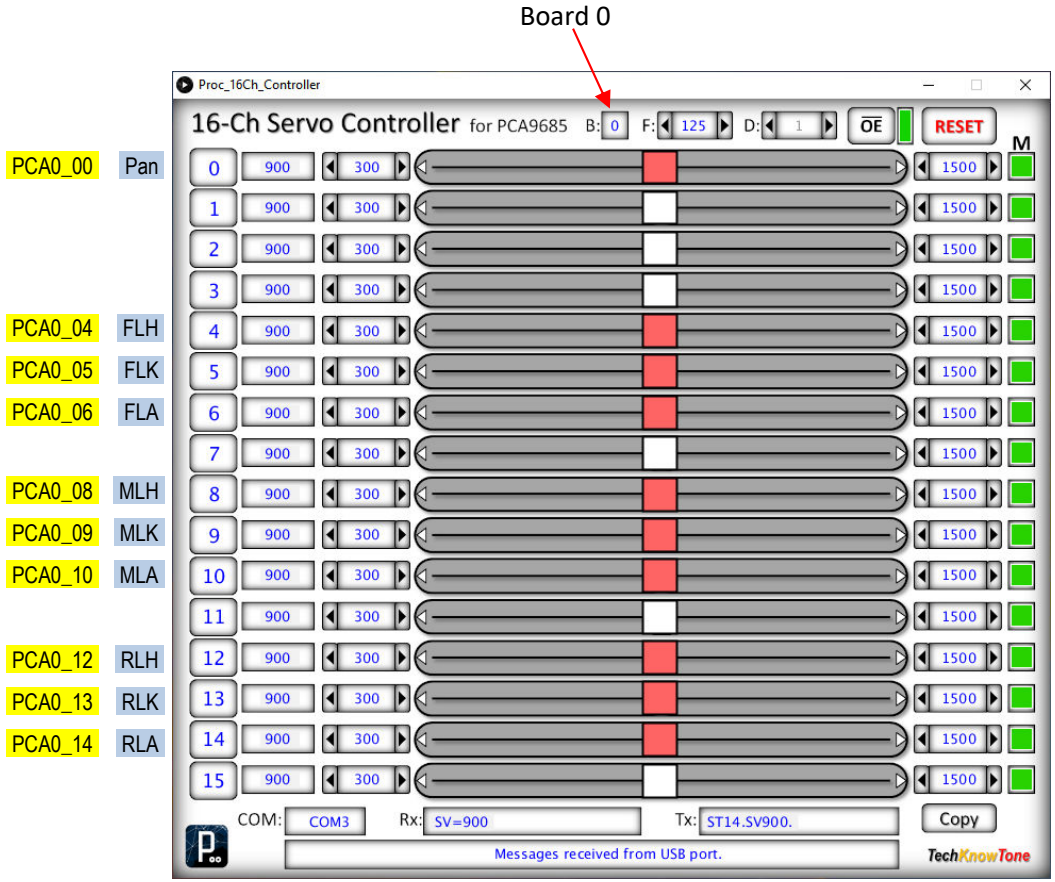
At boot up, or if reset, the digital pins of the ESP32 are set as high impedance inputs; and the boards have a 10k pull-down resistor on the OE input. This could lead to either board outputting PWM signals to the servos, and the robot jumping. To avoid this, a 1kΩ resistor is wired between the 3v3 pin and OE, to pull it up until the code in the micro takes control. Fit the 1kΩ resistor, as seen on the right of this diagram.

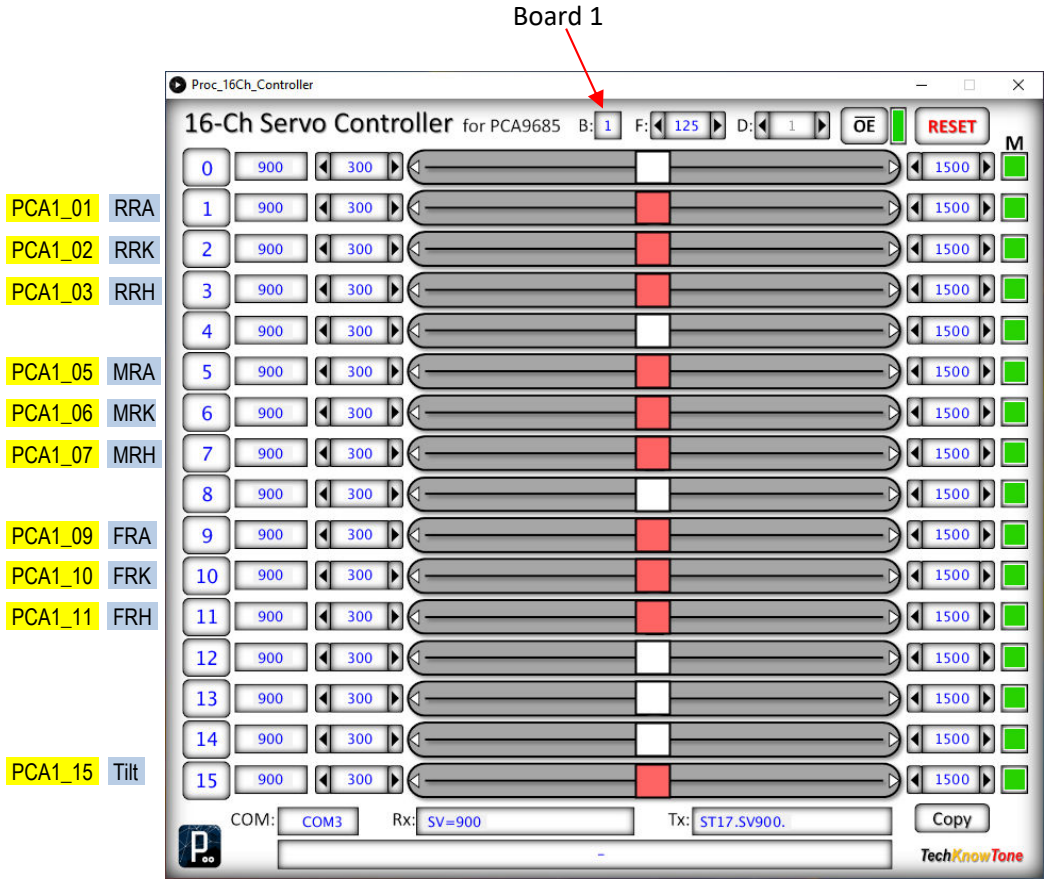


Issue: 1.2 Released: 10/05/2025 TechKnowTone

Servo Channel Control

The assignment of servo channels maps onto the 16-Channel controller app as follows. Note that we only use 20 of the available 32 outputs. The remaining 12 are ignored. The OE button acts as a global enable/disable switch. To make a channel active you simply click on its left-hand channel number, and the slider turns red. Click on the board field number to select boards 0 – 3. Note that the PCA references contain the board numbers, as in PCA**0**_05 and PCA**1**_05, which are channel 5's on boards 0 and 1. Leave the PWM frequency set at 125Hz. A dither signal can be added to a PWM channel to step the servo either side of its current value, to see what its true centre resting position is, given that servos have a deadband.



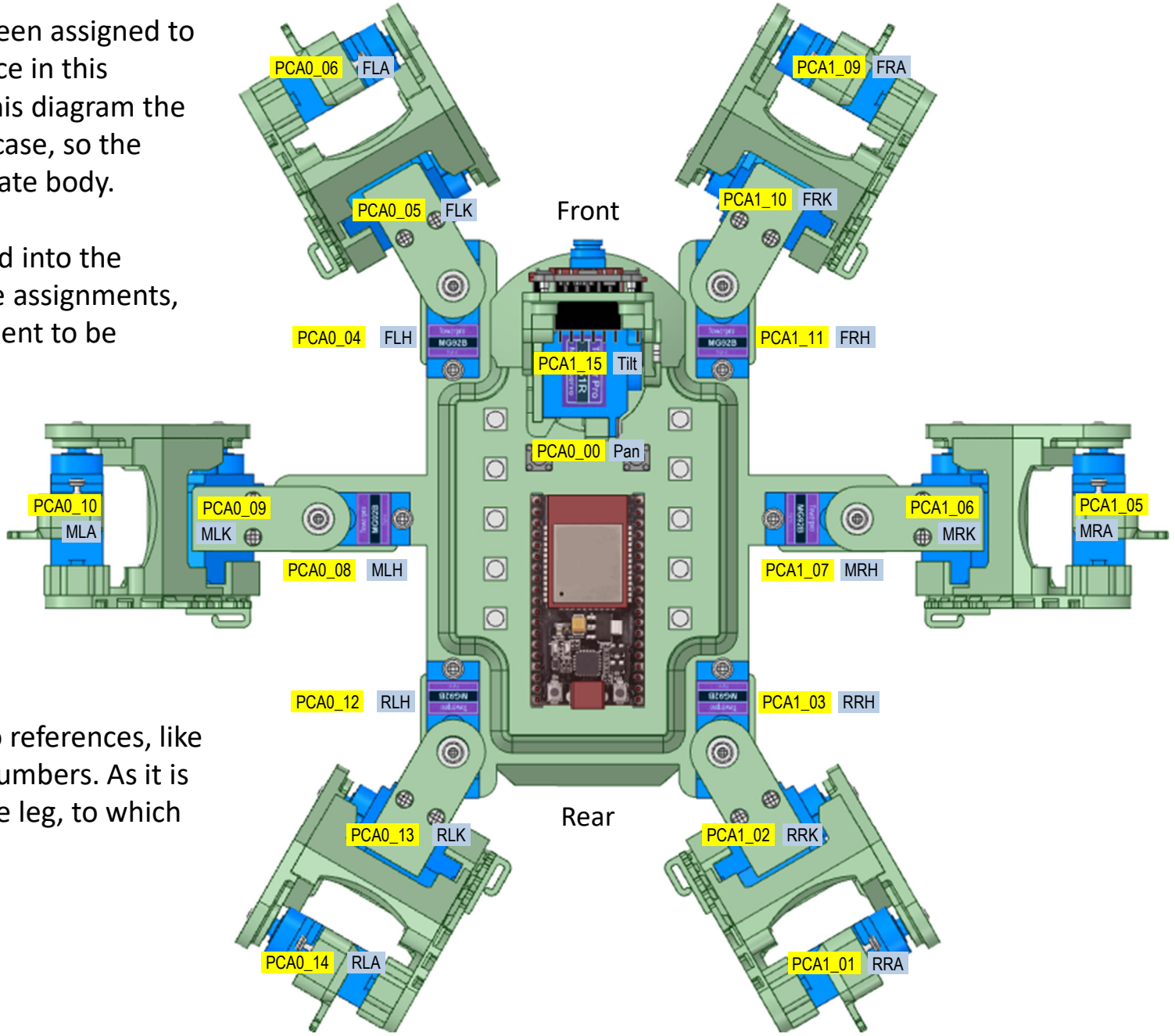


Issue: 1.2 Released: 10/05/2025

Servo Assignments – facing up

The following channel numbers have been assigned to the SpidaBot servos, for future reference in this document, and in the ESP32 code. In this diagram the Pan servo is not visible, it is inside the case, so the reference PCA0_00 sits on the micro plate body.

Your wiring of the servos, when plugged into the PCA9685 controllers, must match these assignments, for the code and the rest of this document to be correct.



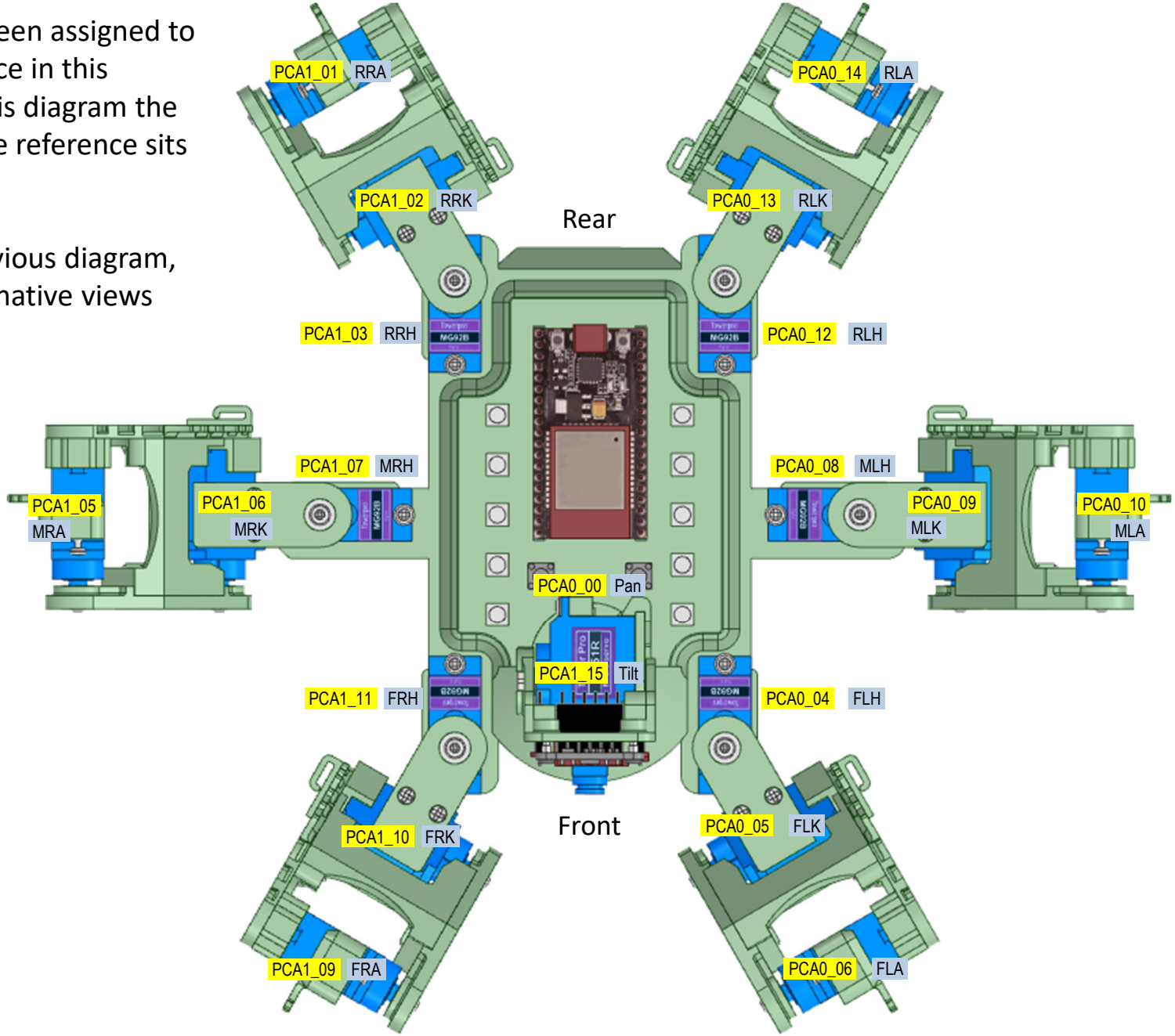
Within the main code we use the servo references, like SMLA and SFRH, rather than the PCA numbers. As it is easier to understand the position of the leg, to which the code is referring.

Issue: 1.2 Released: 10/05/2025 TechKnowTone

Servo Assignments – facing down

The following channel numbers have been assigned to the SpidaBot servos, for future reference in this document and in the ESP32 code. In this diagram the Pan servo PCA0_00 is not visible, so the reference sits on the micro plate body.

The assignment is the same as the previous diagram, it's just sometimes useful to have alternative views when calibrating servos.

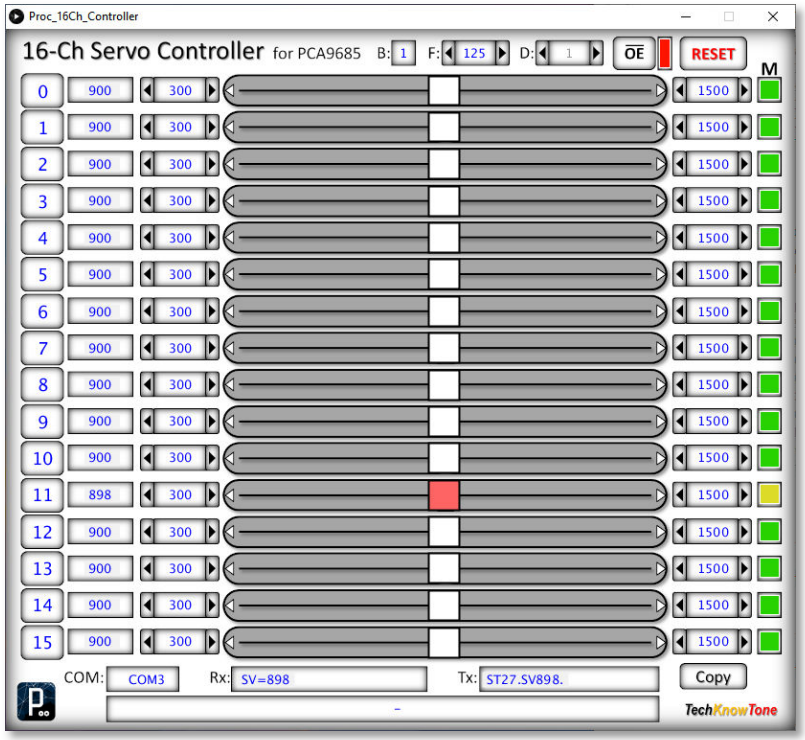


Issue: 1.2 Released: 10/05/2025 TechKnowTone

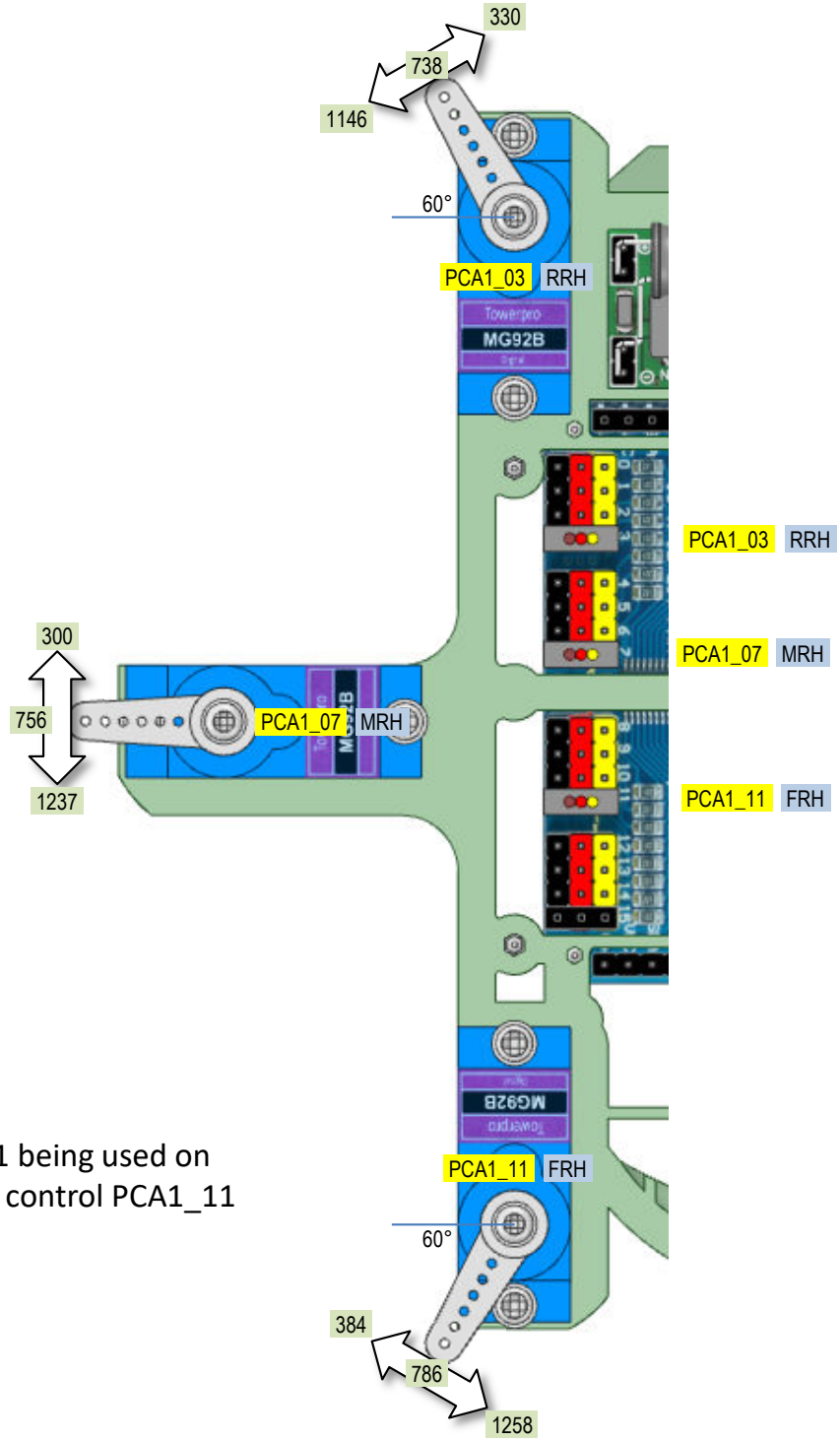
Course Calibration – Hip Servos

At this stage it is assumed that all six of the hip servos have been attached to the Hip frame and plugged into their respective PCA9685's. Here we will set up the right-hand side of the SpidaBot, with the left-hand process being the same, but with the front and rear servos being in effect reversed.

Using the 16-channel app, and the code flashed into the ESP32 micro, we select servo 7 on board 1, enable OE, and move the slider to 768, which will give a 1500µs PWM pulse train, setting the right-hand middle servos drive shaft to its nominal centre position. Then attach the leaver arm to the servo as shown, to best achieve the centre position. The splines on the drive shaft may compromise your ability to achieve this, so find the best position.



Channel 11 being used on board 1 to control PCA1_11



Issue: 1.2 Released: 10/05/2025

Course Calibration – Knee Servos

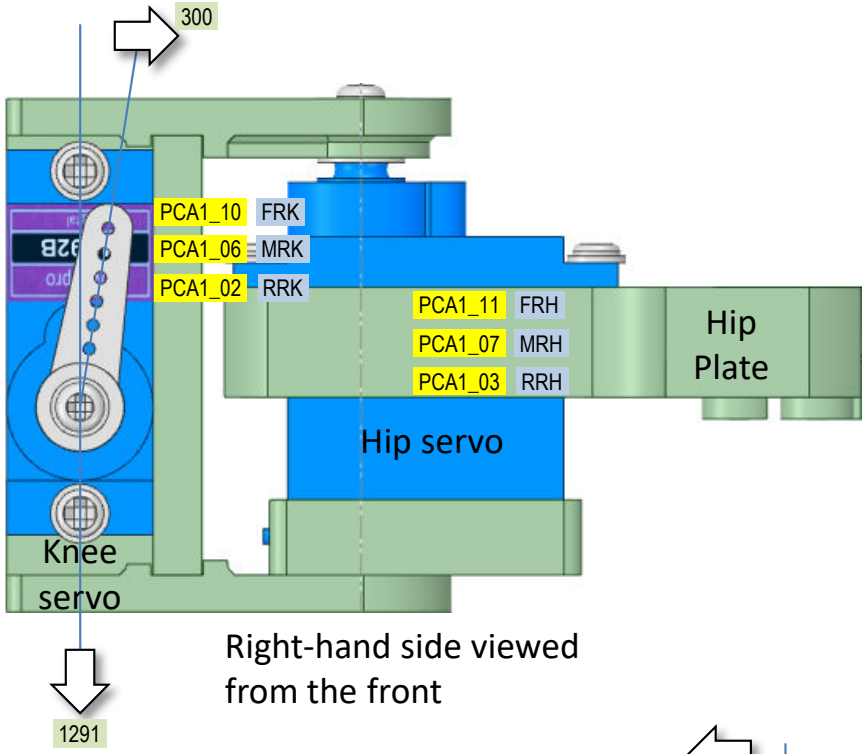
Now attach one of the right-hand knee servos to a hip hoint, and plug it into the corresponding PCA9685 connector. Ignore the excess wire at this stage, you will tidy that up once the servo arms have been attached and calibrated.

Select the correct channel number in the app, and enable it. Then move the slider to a PWM value of 300. Attach the servo arm onto the splined drive shaft, such that it is just past the vertical 12 o'clock position. Then check that by moving the slider towards 1500 the servo arm can reach the vertical down position, travelling in an anti-clockwise direction.

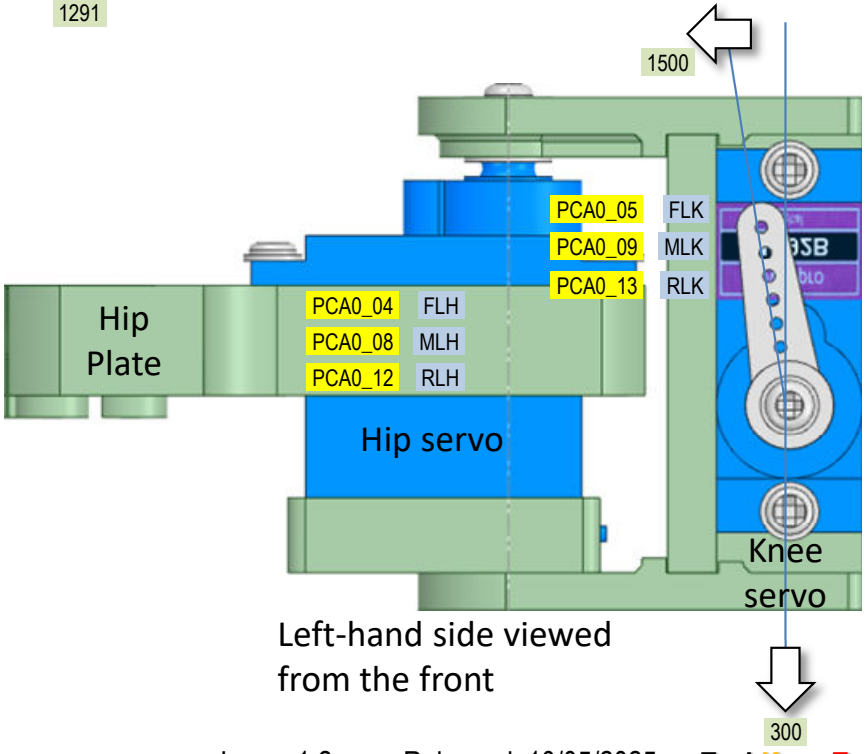
Once you are happy that the lever arm is in the optimum position, screw the arm onto the servo shaft, to retain that location.

Repeat this process for all 3 right-hand knee servos.

In a similar fashion, attach left-hand knee servos to the hip joints. Select the correct channel number in the app, and enable it. Then move the slider to a PWM value of 1500. Attach the servo arm onto the splined drive shaft, such that it is just before the vertical 12 o'clock position. Then check that by moving the slider towards 300, the servo arm can reach the vertical down position, travelling in a clockwise direction.



Right-hand side viewed from the front



Left-hand side viewed from the front

Issue: 1.2 Released: 10/05/2025 TechKnowTone

Course Calibration – Ankle Servos

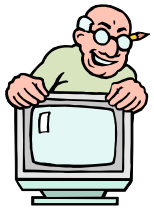
Attach a lower leg to an ankle servo, and plug it into the corresponding PCA9685 connector. Ignore the excess wire at this stage, you will tidy that up once the servo arms have been attached and calibrated.

Select the correct channel number in the app, and enable it. Then move the slider to a PWM value of 300. Attach the servo arm onto the splined drive shaft, such that it is pointing downwards and to the right. Then place the Link Double Leaver part over the servos lever arm as shown. The double leaver should partially overlap the lower leg as shown (see arrow). Then check that by moving the slider towards 1500 the link double leaver can reach the vertical upwards position, travelling in an anti-clockwise direction. This is to give the lower leg and ankle joint maximum movement (~180°).

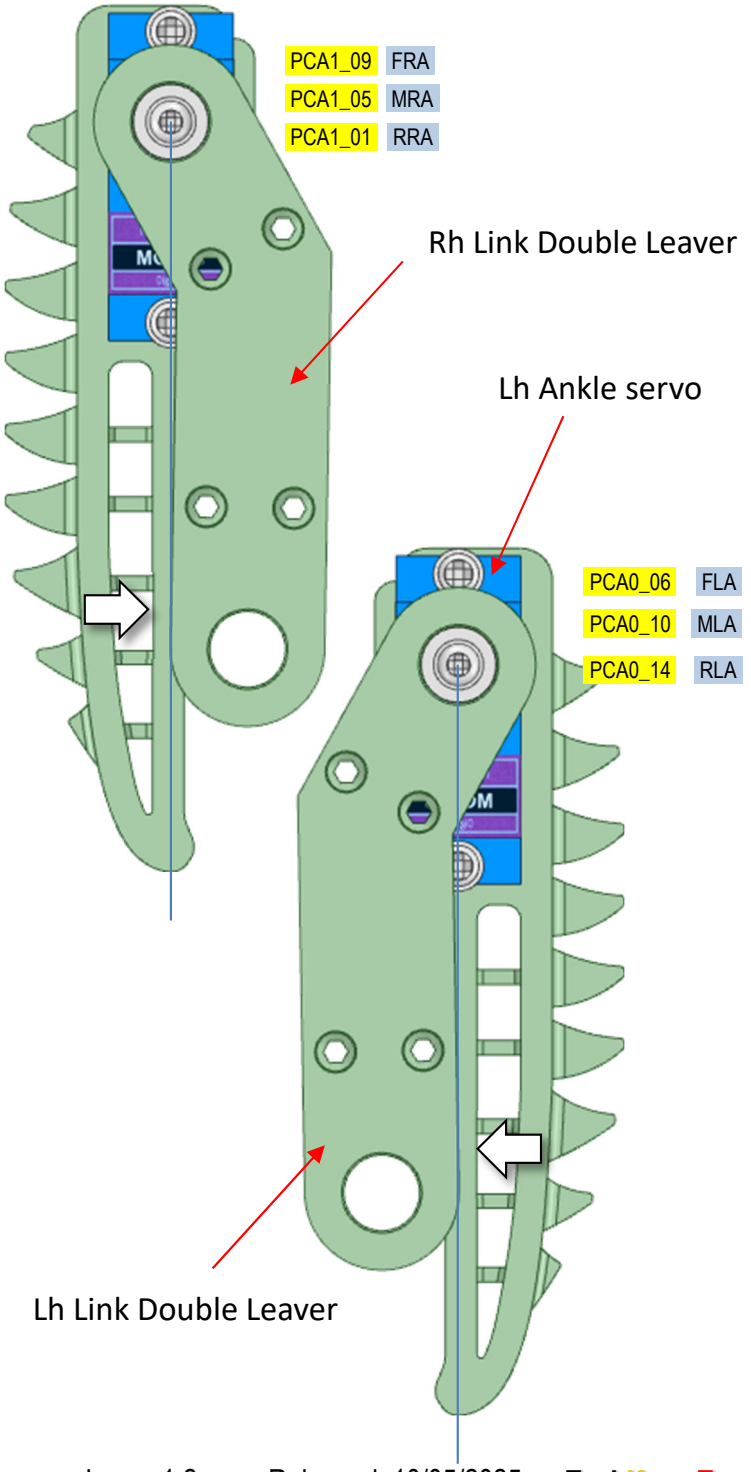
Once you are happy that the ankle servos lever arm is in the optimum position, screw the arm onto the servo shaft, to retain that location. Then repeat this process for the remaining 2 right-hand ankle servos.

Then repeat the process for the left-hand side, but this time with the start PWM value of 1250, rather than the 300 used in the above.

Then repeat this process for the remaining 2 left-hand ankle servos.



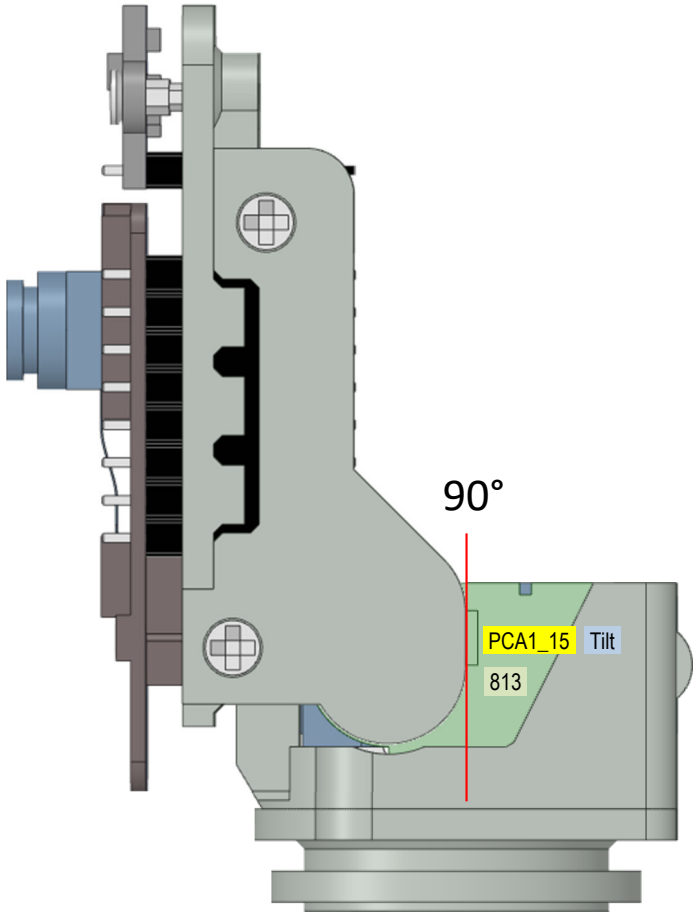
This procedure is all about attaching the servo arms, provided with the servos, in a position that will give them maximum movement.

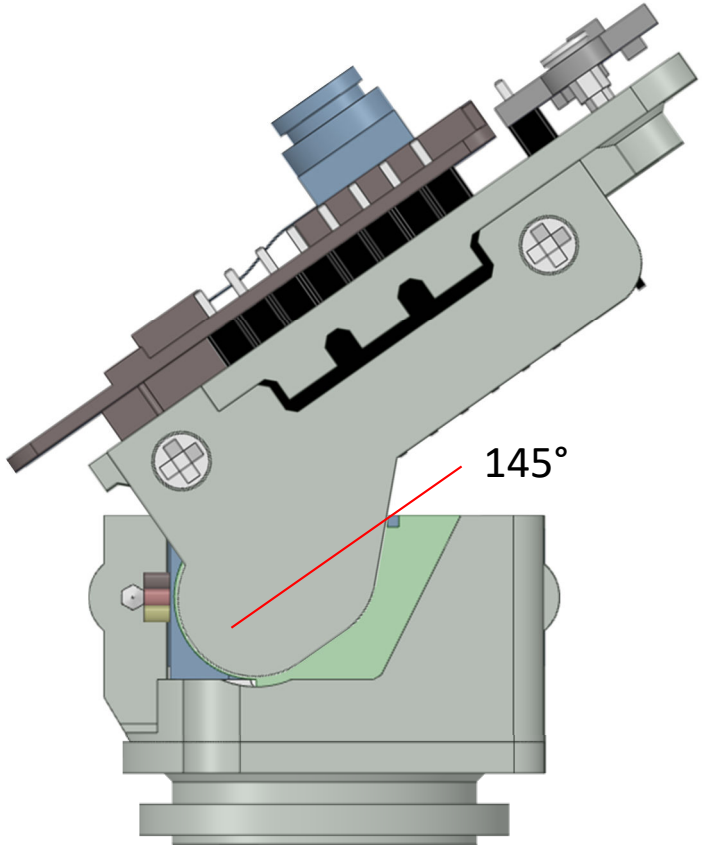


Issue: 1.2 Released: 10/05/2025 TechKnowTone

Course Calibration – Head Tilt Servo

The tilt servo PWM value is set to a value of 768, whilst attaching the leaver arm, placing the ESP-CAM in the vertical position. This course setting should then allow the head to be tilted forwards and backwards over a good range. There is an indent in the model to help determine the 90° vertical angle.





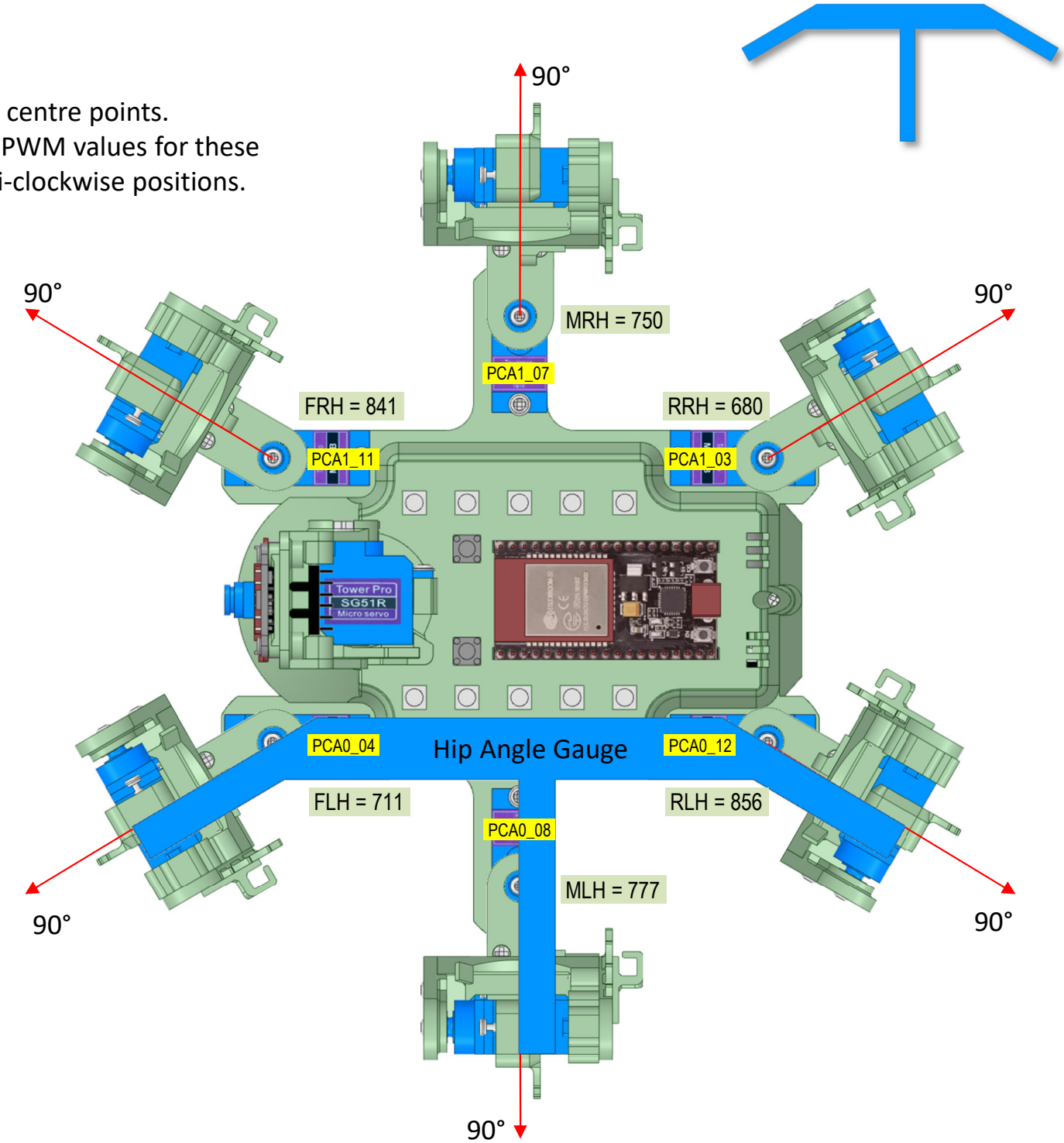
Fine Calibration – Hip Centre Points

This diagram shows the hip joints at their respective centre points.
To improve the linearity of the hip angles we record PWM values for these positions, and then their extreme clockwise and anti-clockwise positions.

To make the process of determining these PWM values easier, you can print off and use the bespoke angle gauge, shown here in blue. Place the SpidaBot on its stand, and pull out the legs, so that knee and angle joints don't interfere with the angle gauge measurements.

The angle gauge sits on the top edge of the micro plate, and its legs are at the target 60° and 90° angles. Use the 16-channel controller app to move the hip joints, such that the legs align with the gauge. Record the six PWM values, and enter them into your code.

Make the PWM readings as accurate as possible. The dither function, built into the app can usually be used to gain a more accurate reading. It effectively cycles the PWM value sent to the SpidaBot, around the set angle, to see where the servo actually settles.

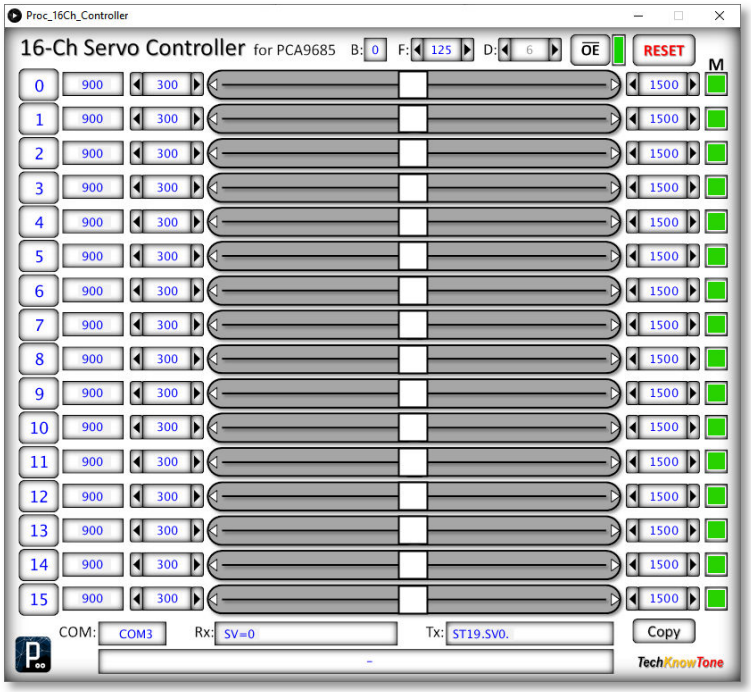


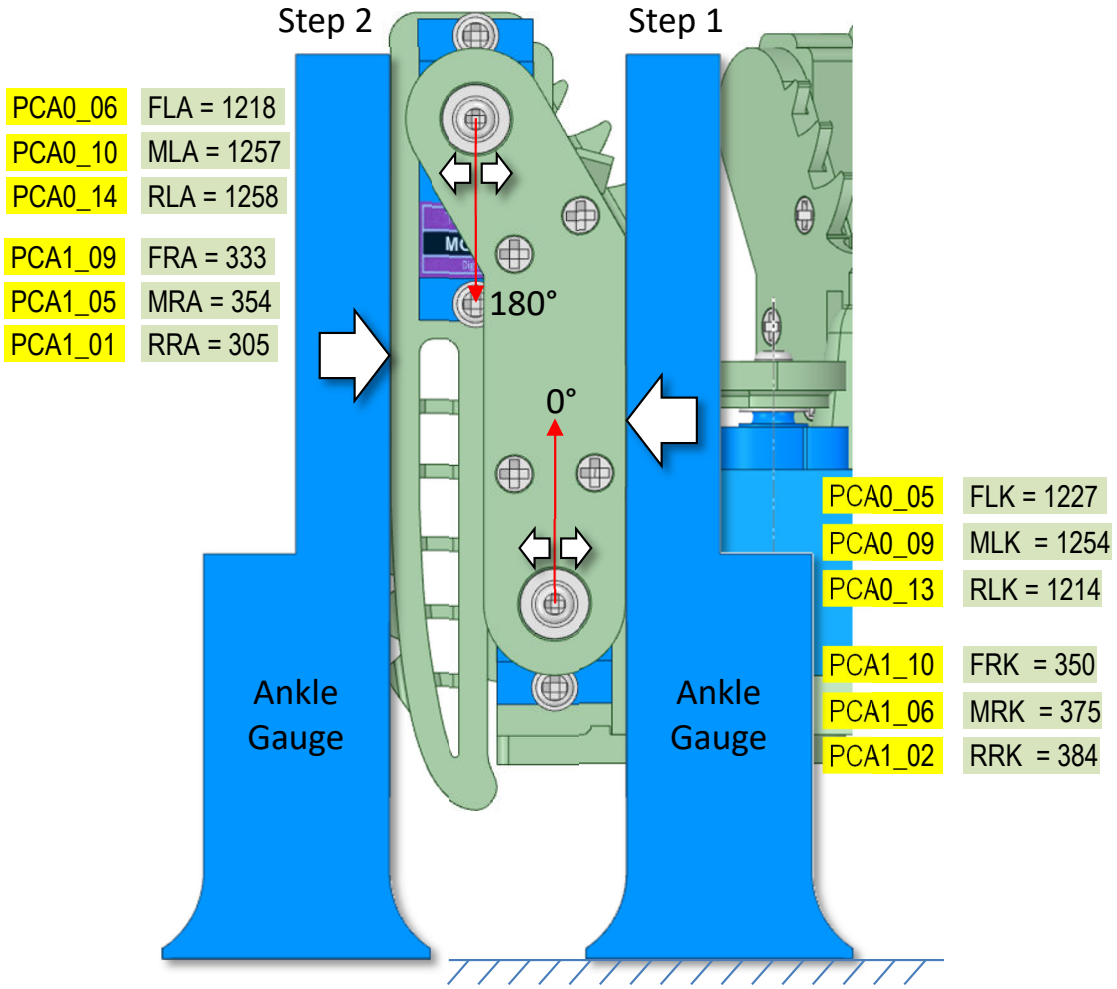
Issue: 1.2 Released: 10/05/2025 TechKnowTone

Fine Calibration – Knee Lever Plate

This diagram shows a right-hand knee joints, moved to its vertical position, and the leg facing down. Here, in step 1, we determine the PWM values needed to move each knee joint to a vertical position; defined as 0° in this design. Then in step 2 we use the same gauge to determine the PWM values needed to move each ankle joint to a vertical position; again defined as 180° in this design.

Use the ankle gauge as shown, rested on the surface that the SpidaBot stand is on, using its long vertical edge to act as a guide and reference. Adjust the servo PWM values to achieve this position and record their values, as seen here.





Issue: 1.2 Released: 10/05/2025 TechKnowTone

Fine Calibration – Hip Collision Points

This diagram shows the hip joints, moved from their centre 90° positions, to a point where they just touch the adjacent legs. Each leg is shown as being in two positions. The collision points are denoted by a star:☆

These angles are determined with the knee joints in their upright positions (set on the previous page), and were obtained from the 3-D model of the SpidaBot. Again use the 16-channel app to rotate the hip joints, to position the legs as shown, and record the PWM values needed for each.

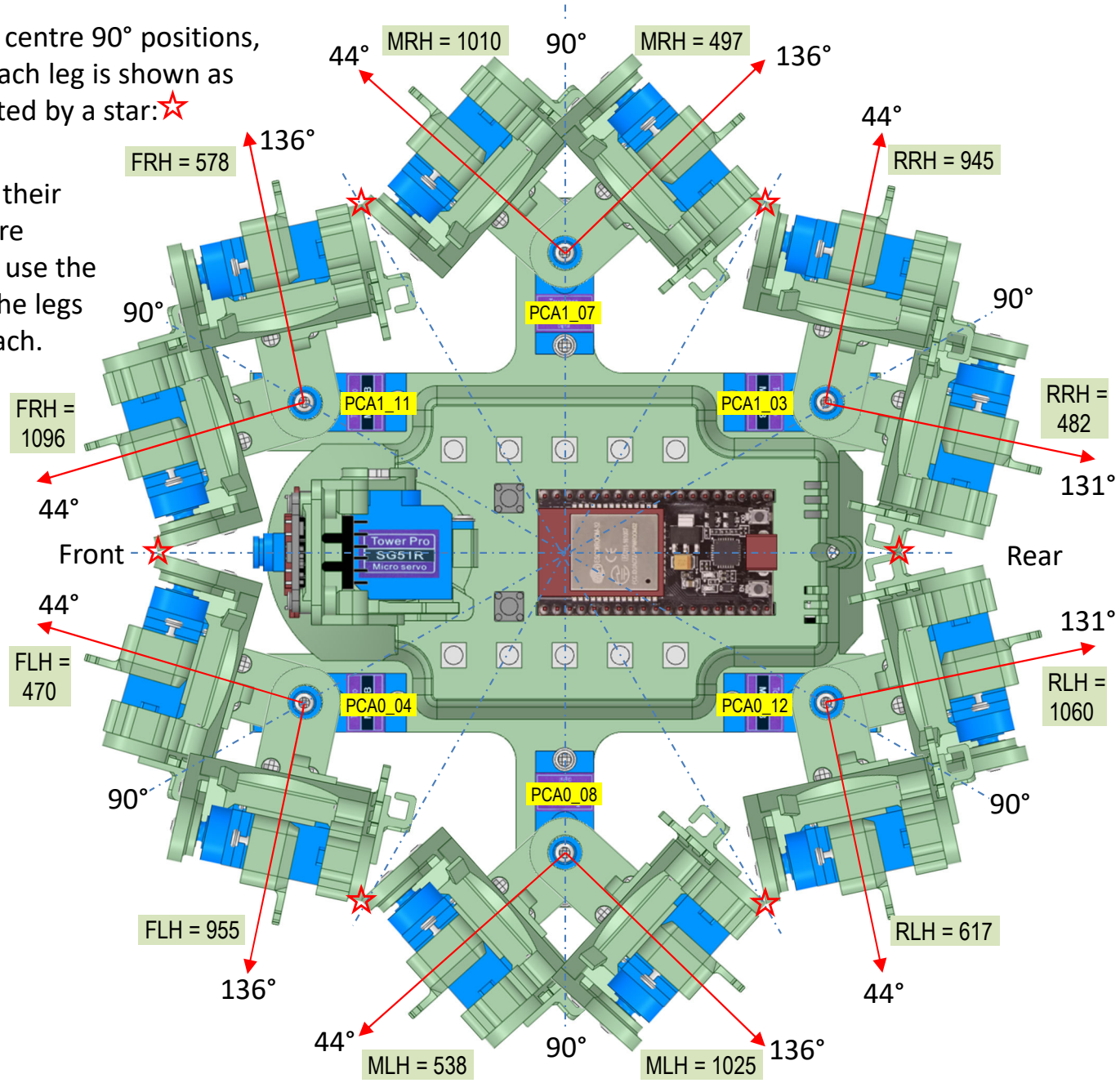
Note that the collision points at the rear, are associated with the servo wire clips, and not the top of the knee joints, like the others.

The true angles of these collision points, were determined from the 3D model. Both the relationship of PWM values to angles, and their collision points are stored in code as limits.

For example the front right-hand hip servo has the following definitions in code:

```
239   #define FRH44  1096   // front right hip 44°
240   #define FRH90   841   // front right hip 90°
241   #define FRH136  578   // front right hip 136°
```

This relationship between PWM value and arm angle is important, as we use angles to define movements, to which we then map PWM values.

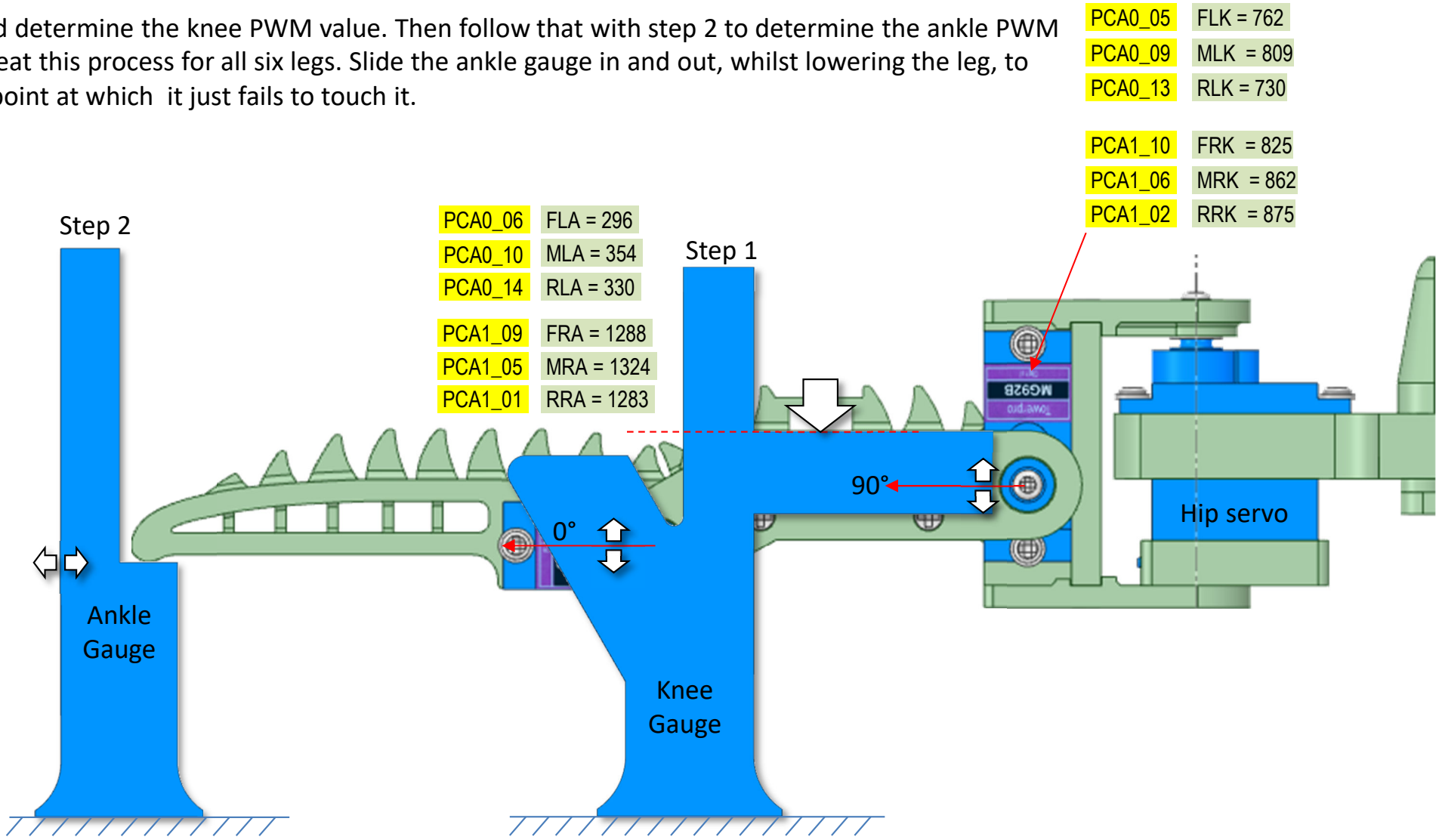


Issue: 1.2 Released: 10/05/2025 TechKnowTone

Fine Calibration – Knee & Ankle

This diagram shows the knee joints, moved to its horizontal 90° position, with the lower leg also positioned horizontally. For each leg we can record two PWM values, one for each servo, knee and ankle. The knee 90° angle is set first, to ensure it is horizontal, before determining the lower leg ankle 0° degree value. Since the ankle value is set, after the knee, it is critical that the knee adjustment is as accurate as possible.

So do step 1 and determine the knee PWM value. Then follow that with step 2 to determine the ankle PWM value. Then repeat this process for all six legs. Slide the ankle gauge in and out, whilst lowering the leg, to determine the point at which it just fails to touch it.



Issue: 1.2 Released: 10/05/2025 TechKnowTone

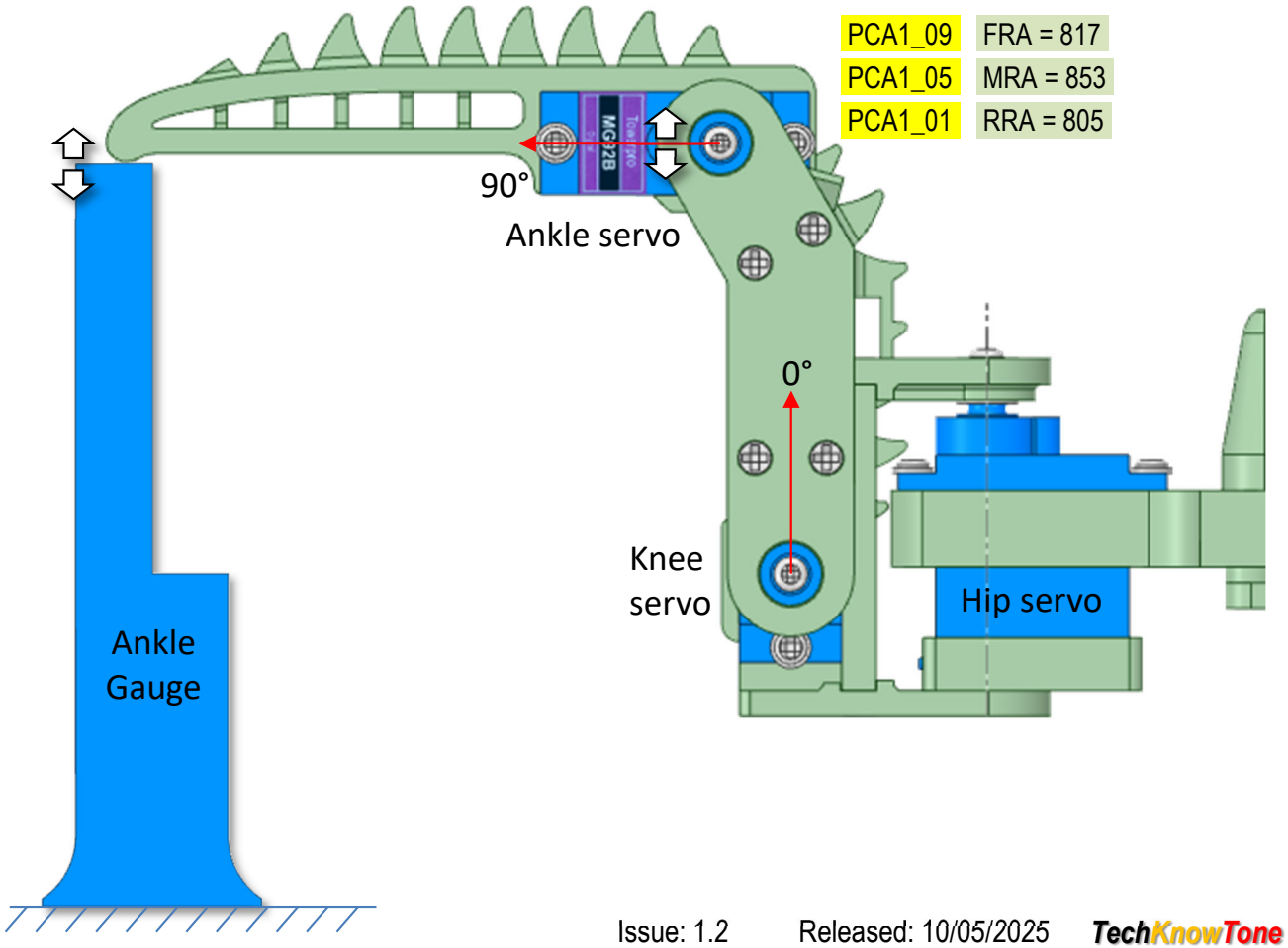
Fine Calibration – Ankle 90°

Return the knee joints to their vertical 0° positions, using the values recorded earlier. Then use the top edge of the ankle gauge to set the ankle servo, such that the leg is horizontal, in its 90° position, as shown in the diagram.

Repeat this for each leg, recording their PWM values. This concludes the leg calibration process, and now all of the PWM values can be defined in your code. View the first tab of the code to see how this is done.

For example, front left ankle servo value here, is defined as: `#define FLA90 759` within the IDE; where the associated angle of 90° is part of the naming convention.

PCA0_06	FLA = 759
PCA0_10	MLA = 791
PCA0_14	RLA = 806
PCA1_09	FRA = 817
PCA1_05	MRA = 853
PCA1_01	RRA = 805



Issue: 1.2 Released: 10/05/2025 TechKnowTone

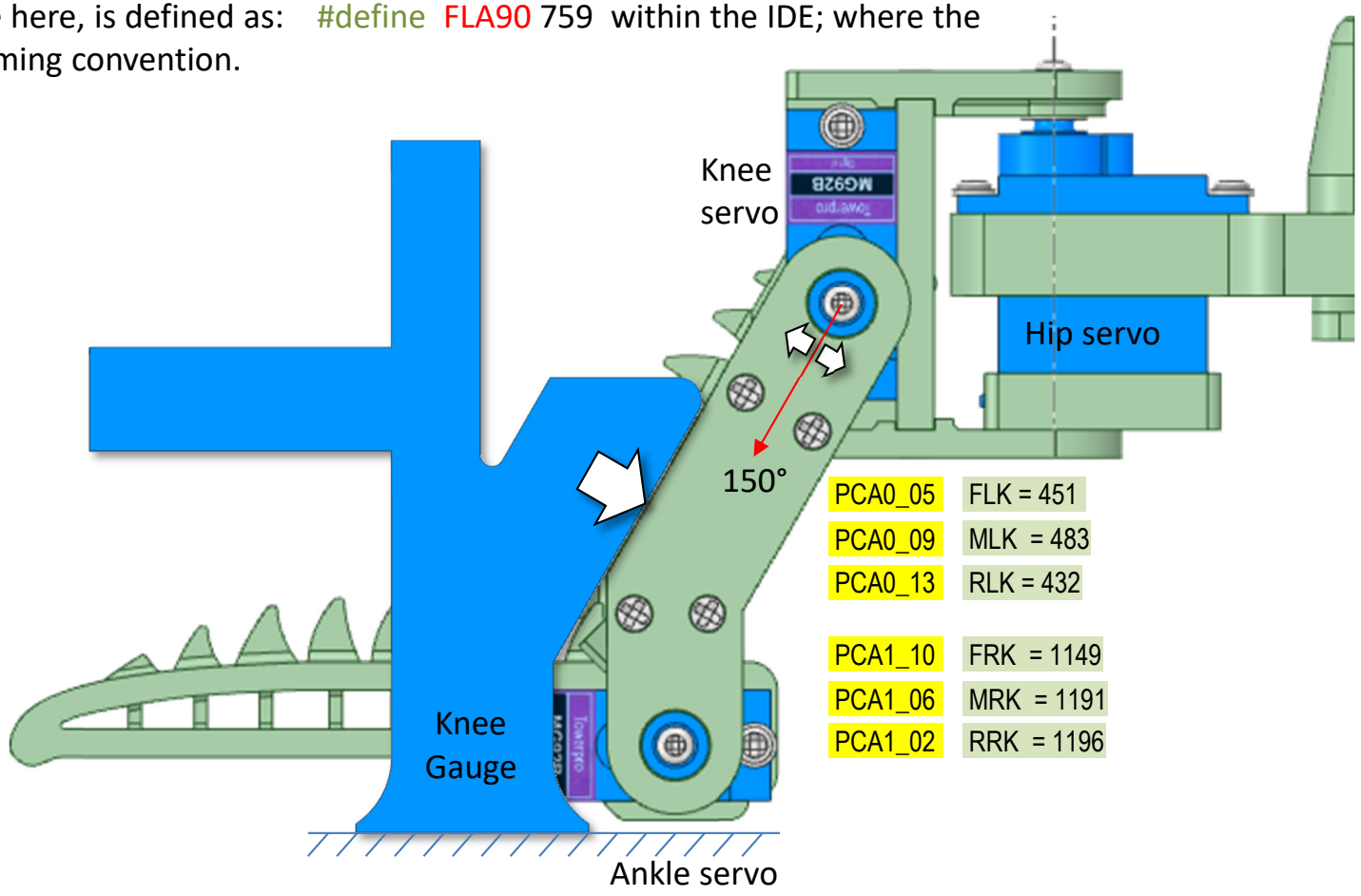
Fine Calibration – Knee 150°

Here we swing the knee joint down such that the lower part of the leg is vertical. This is defined as its 150° position, as shown in the diagram. Note that in order to do this the associated ankle servo needs to be turned off and it will need to be positioned as shown by hand, as it is not possible to drive the servo into this positon.

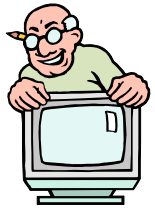
The sloping edge of the kneed gauge is used to check the knee angle.

Repeat this for each leg, recording their PWM values. This concludes the leg calibration process, and now all of the PWM values can be defined in your code. View the first tab of the code to see how this is done.

For example, front left ankle servo value here, is defined as: `#define FLA90 759` within the IDE; where the associated angle of 90° is part of the naming convention.

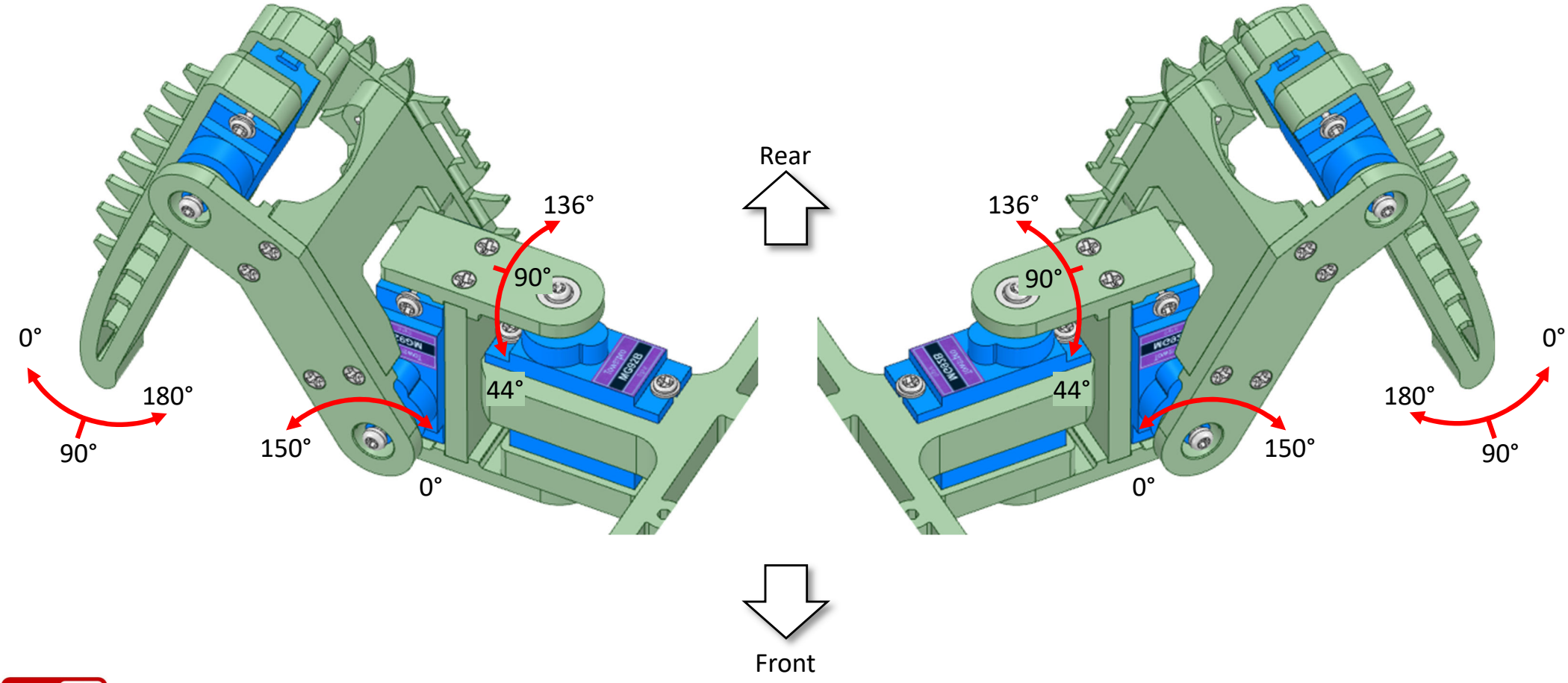


Leg Angles



This diagram summarises the leg angles, and their range of movements, for both right and left handed legs. Note that due to physical constraints of wiring clips, the rear hips can only swing to a maximum of 131°, not 136° as shown here.

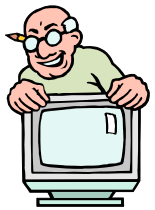
Also note that the angles on one side of the body are effectively the reverse of those on the other side. But consistent, in that to push the toe of a given leg outwards, we move the ankle angle towards 0°. And to swing the hips in a rearward direction we sing all hip angles towards 136°.



Issue: 1.2 Released: 10/05/2025 TechKnowTone

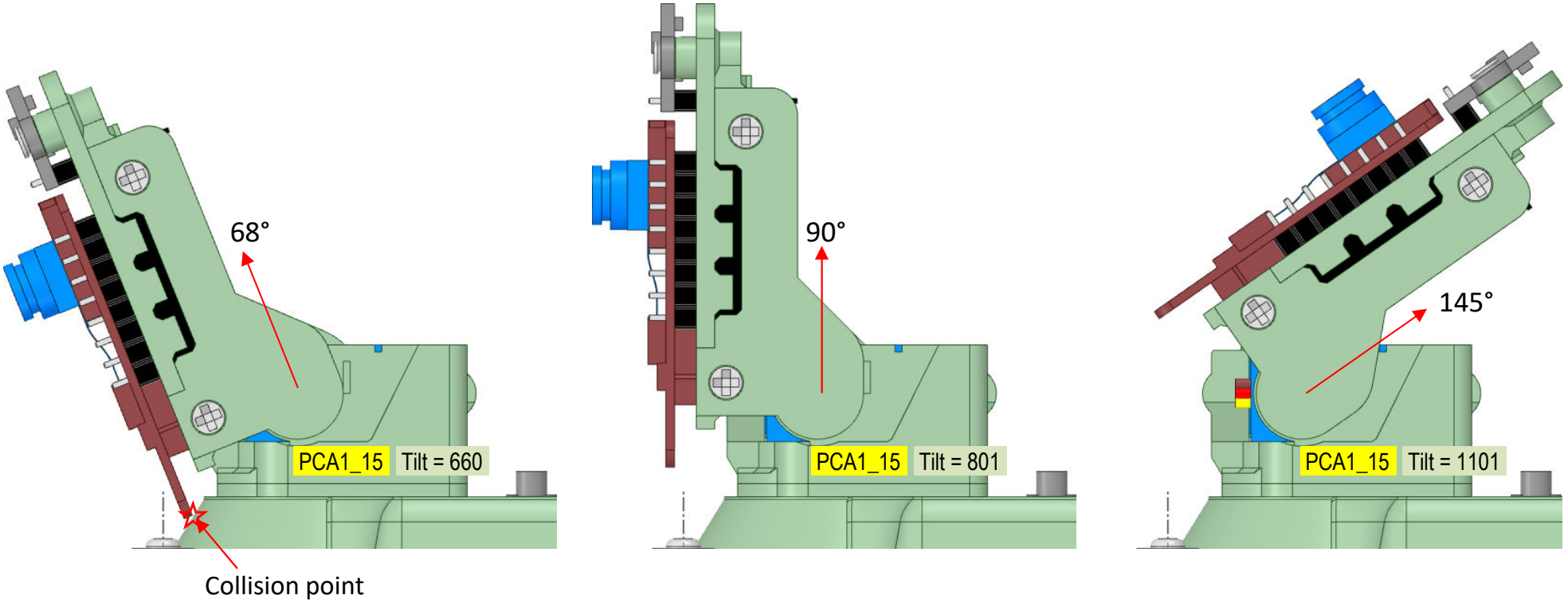
Fine Calibration – Head Tilt

The head pan and tilt servo values are less critical than the leg servos, as they have no effect on the movements of the SpidaBot. Nevertheless we need to calibrate and set limits for the head to work correctly. The diagrams below show the tilt angles for the head, for which you record PWM values; such as Tilt68 and Tilt90.



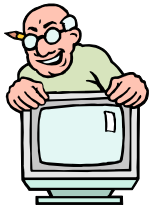
When tilting forward we want to avoid hitting the micro plate cover, so determine a value for Tilt68 which does not give a knocking sound when the head is moved quickly towards it. Otherwise unintentional collisions will have the effect of lifting the camera module out of its socket strip. In my code these values are recorded as:

```
289   #define T68     670   // tilt forwards at 22° from vertical
290   #define T90     801   // not tilt, vertical 90°
291   #define T145   1101   // tilt backwards at 55° from vertical
```



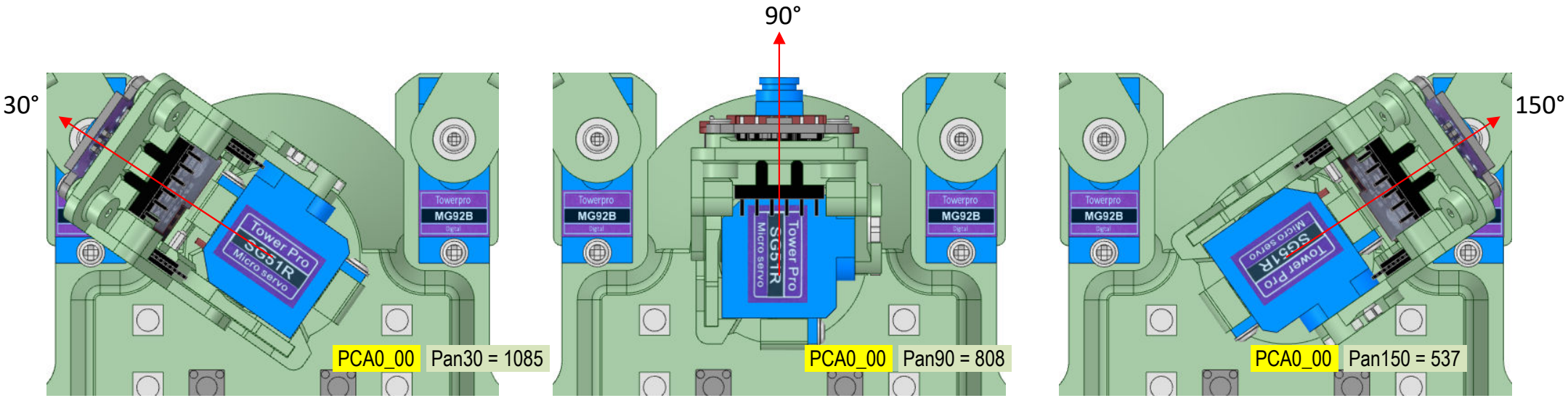
Fine Calibration – Head Pan

The head pan limits are derived from the 3D model as +/- 60°. We determine the PWM value for the head looking forward, and define this as 90°. Then swing the head to the left by 60° to the P30 value, and to the right by 60° to the P150 value. In my code I recorded the values as follows:



```
268   #define P30    1085   // pan turn left 30°
269   #define P90     808   // pan turn centre 90°
270   #define P150    537   // pan turn right 150°
```

Ensure that with the head tilted forward, to its T68 value, that the extremes of the pan angle do not cause collisions to occur.



Issue: 1.2 Released: 10/05/2025 TechKnowTone

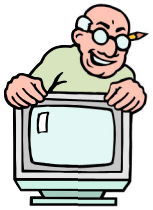
MPU6050 Calibration

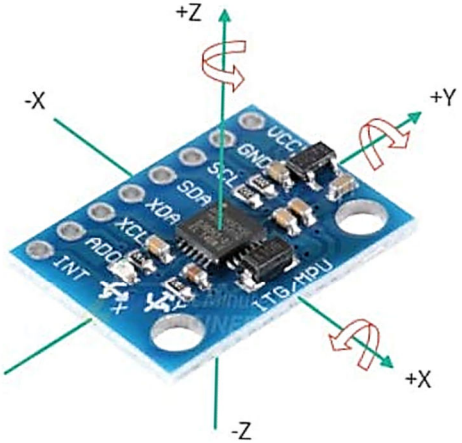
With the MPU6050 mounted on the roof of the micro plate as shown, the following principles apply

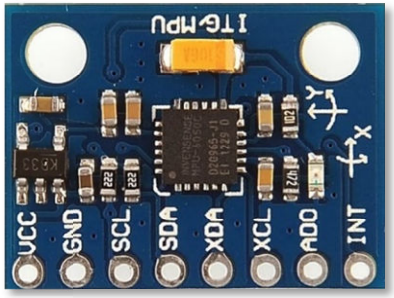
Pitch - X gyro, +ve tilt forwards, -ve tilt backwards
- Y accelerometer, -ve lean forwards, +ve lean backwards

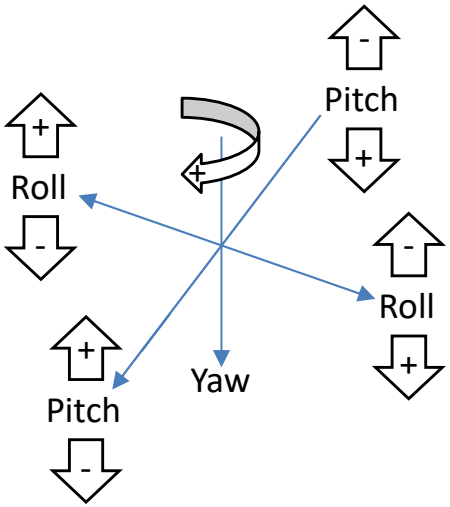
Roll - Y gyro, +ve tilt right down, -ve tilt left down
- X accelerometer, +ve lean right, -ve lean left

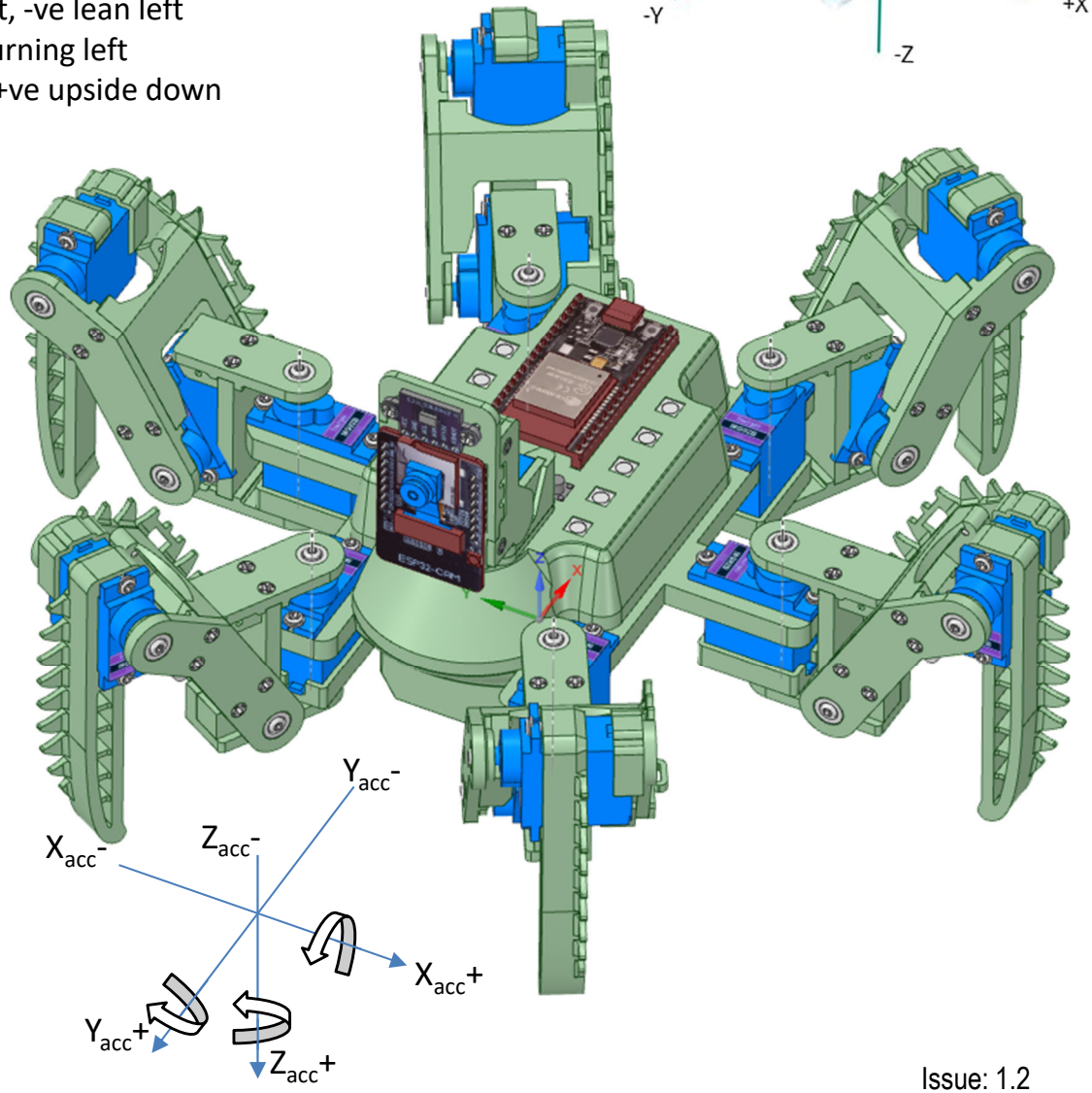
Yaw - Z gyro, +ve turning right ,-ve turning left
- Z accelerometer, -ve upright, +ve upside down











Use the Display Monitor+ app, and the MPU displays, to determine offsets for your MPU.

Set the offsets to zero initially, so that you can see there raw readings.

To improve the accuracy of the X and Y offsets, you can rotate the SpidaBot, about its vertical axis, to acquire min/max values.

It is unlikely that the surface upon which the SpidaBot is resting is truly level.

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 = 3110 On V_{ADC} (2 x 4.2v)

MAX: (100%) V_M = 8.2v, gave A0 = 3010 on V_{ADC} (2 x 4.1v)

HIGH: (80%) V_H = 7.6v, gave A0 = 2750 on V_{ADC} (2 x 3.8v)

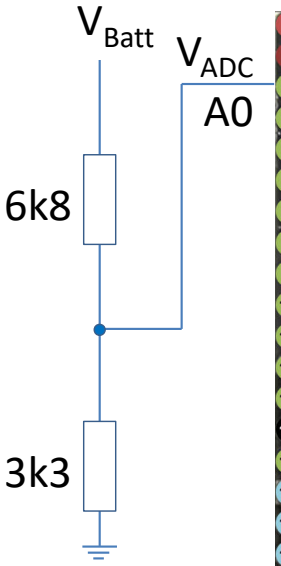
WARNING: (20%) V_{BW} = 7.2v, gives A0 = 2595 on V_{ADC} (2 x 3.6v)

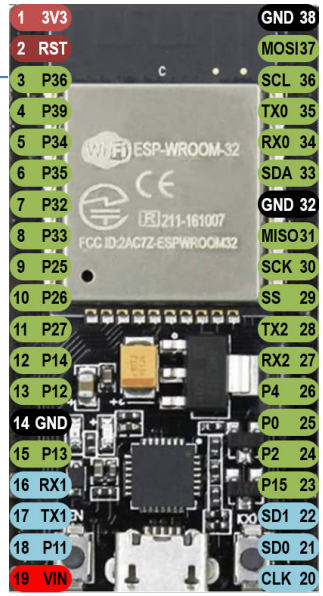
CRITICAL: (0%) V_{BC} = 6.6v, gives A0 = 2355 on V_{ADC} (2 x 3.3v)

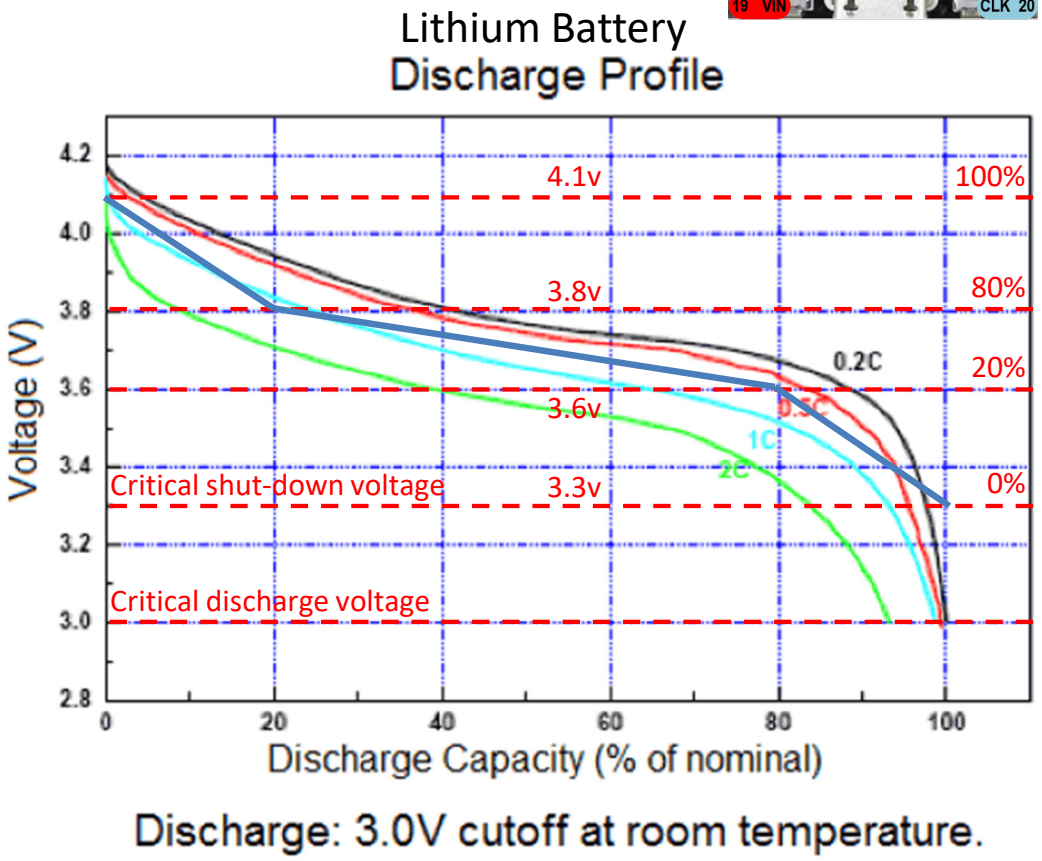
The code will sample the battery voltage on power-up to ensure it is sufficient, then at every 40ms interval, calculating an average (1/50) to remove noise. It also detects no battery as USB mode.

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.







Discharge: 3.0V cutoff at room temperature.

Issue: 1.2 Released: 10/05/2025 TechKnowTone