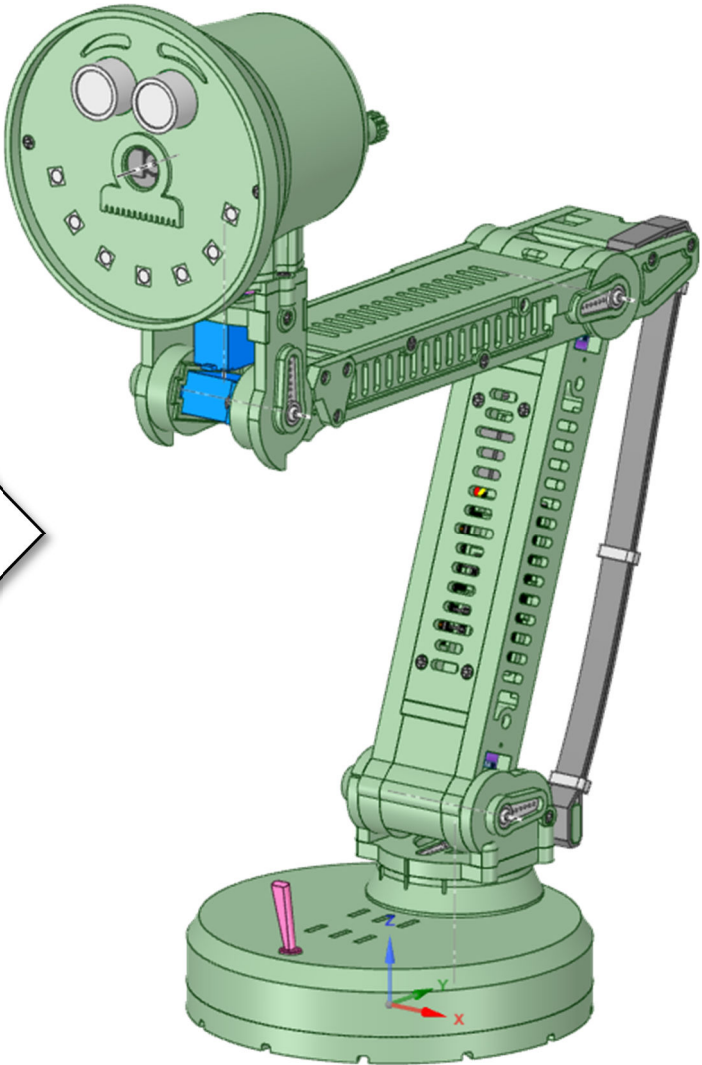
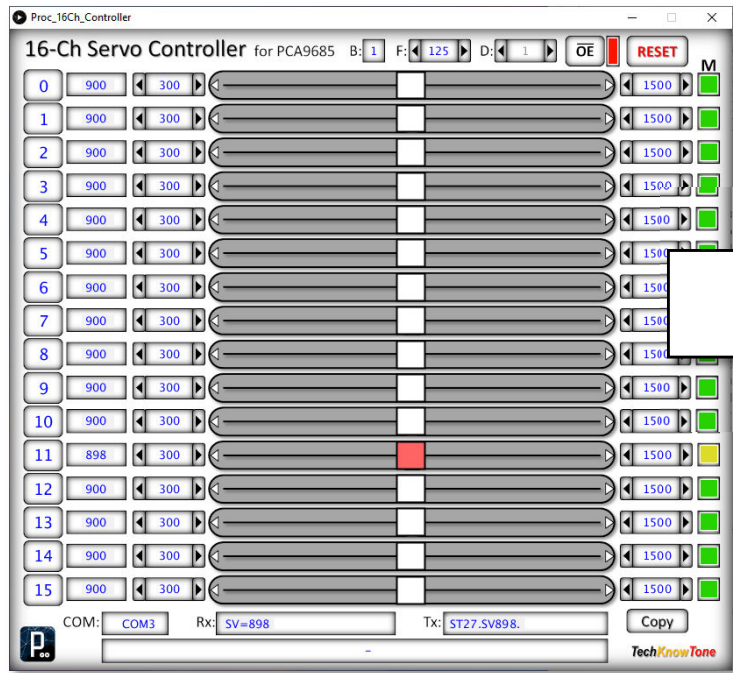


PIXAR Lamp Calibration



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.



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 $V_{BM} \approx 8.4v$ 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 x 3.3v)

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 $V_{ADC} \approx 4095$ 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 $10.1v \approx 4095$, 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: (100%) $V_M = 8.2v$, gave A0 = 3130 on V_{ADC} (2 x 4.1v)

HIGH: (80%) $V_H = 7.6v$, gave A0 = 2892 on V_{ADC} (2 x 3.8v)

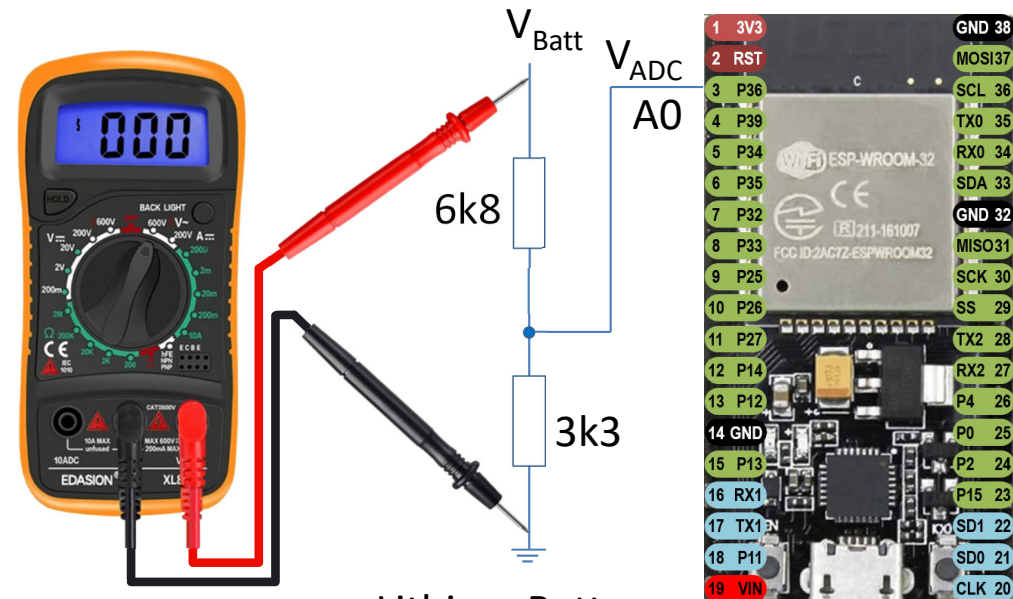
WARNING: (20%) $V_{BW} = 7.2v$, gives A0 = 2722 on V_{ADC} (2 x 3.6v)

CRITICAL: (0%) $V_{BC} = 6.6v$, gives A0 = 2485 on V_{ADC} (2 x 3.3v)

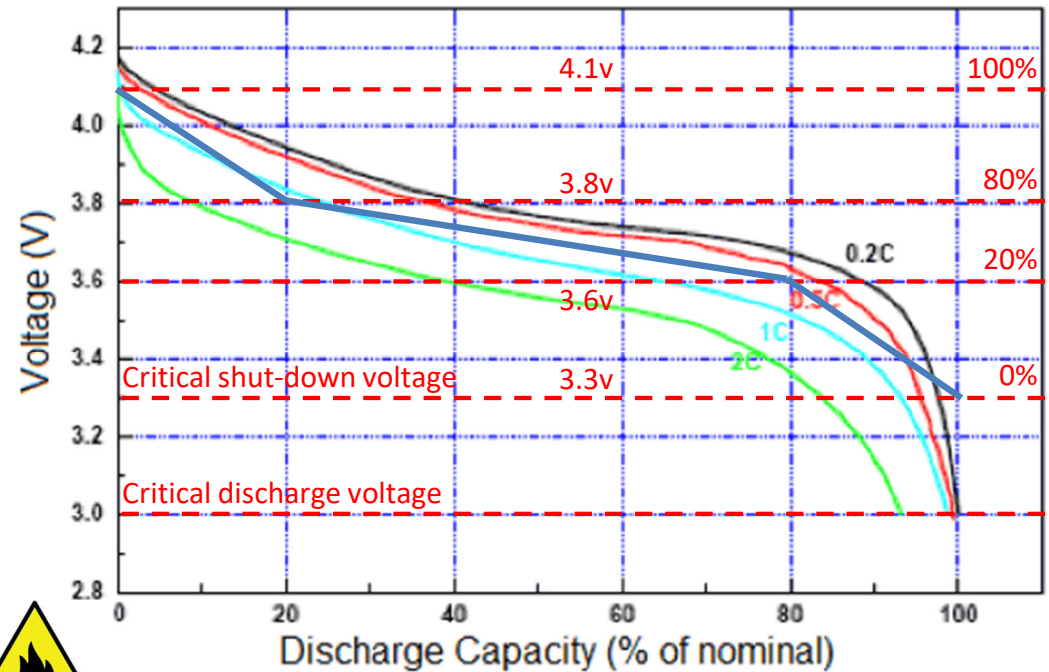
The code will sample the battery voltage on power-up to ensure it is sufficient, then at every 40ms interval, calculating an average (1/50) to remove noise. It also detects no battery as USB mode.

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.



Lithium Battery Discharge Profile



Discharge: 3.0V cutoff at room temperature.

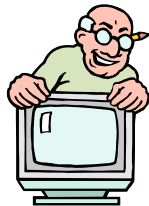


Why do we need to calibrate the servos?

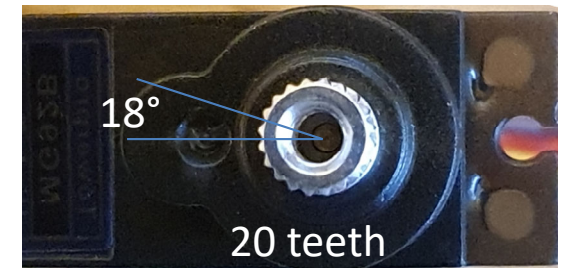
- No two servos are the same.
- Servos can overheat and be damaged if not setup correctly.
- Course calibration must be performed during the assembly process.
- This sets approximate positions for the lever arms.
- A servo drive shaft has 20 splined teeth == 18° (best fit $\pm 9^\circ$)
- Course calibration ensures servos are within mechanical range/limits.
- Fine calibration sets angles, and min/max robot physical limits.
- The ESP32 C++ code needs limit values in order to work accurately.
- Hence, all PIXAR robots have a unique set of calibrated PWM code values.
- No two PIXAR's 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 MG92B servos, for best performance.



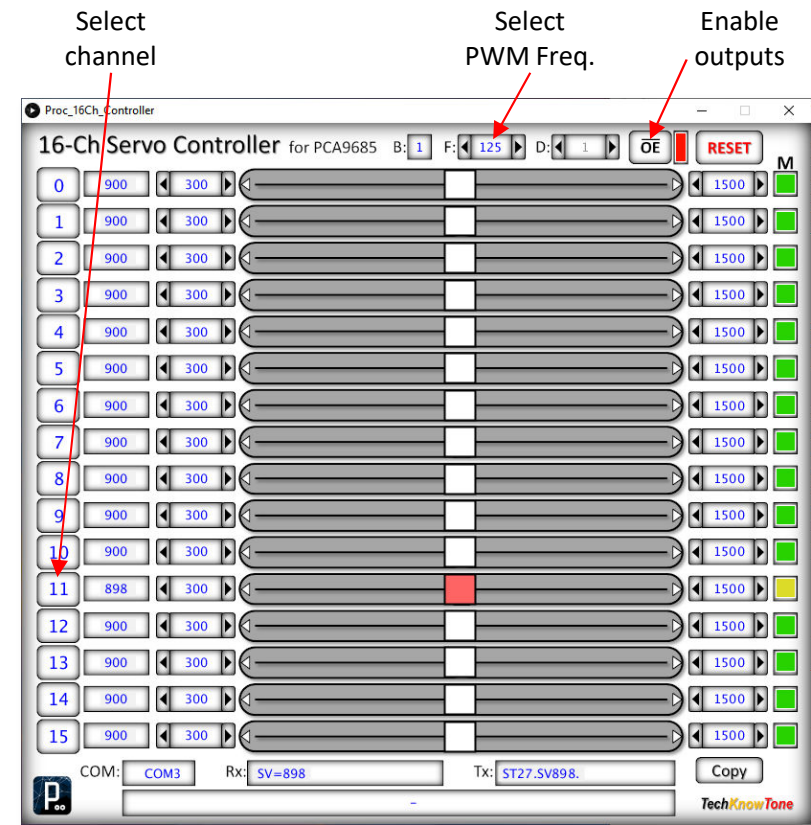
Servo Testing

The PIXAR lamp employs 5 powerful MG92B servo motors, providing the high torques needed to move it efficiently. Course calibration, during the construction phase, is most easily performed using a HJ servo tester. These relatively low cost devices, explained on the next page, provide an effective means of determining servo PWM values, and can be used on all of your projects that employ servos using PWM as the control signal.

I have created a Windows app which, when used with an ESP32 micro running the PIXAR code, enables you to control up to 16 servos. Selected servos will respond to the on-screen 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 fine calibration, after the build process. It is mainly used to determine limits of travel, at the extremes of movement. These limits also correspond to specific angles, which then allow the robot to be controlled by mapped angle values, making them independent of the servos PWM values.

This app can be used with other projects employing servos, provided that you use the C++ code in PIXAR lamp, which enables the micro to decode and implement the serial commands. It is designed to work with a range of servo types, not only conventional servos like the MG92B. See the page which provides an overview of this app.



HJ Servo Consistency Tester



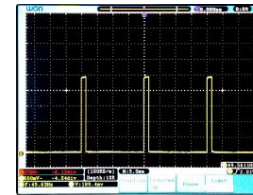
Select Button functions:

- Variable (Pot) pulse width 800 – 2200 μ s (default mode)
- Fixed constant pulse width 1500 μ s, centre position
- Sweep (Pot) pulse width from 800 -> 2200 -> 800 μ s

Pulse Width Button functions:

This is actually pulse frequency (Hz)

- 50H = 50 Hz - run MG92B at this frequency (default)
- 125H = 125 Hz
- 250H = 250 Hz



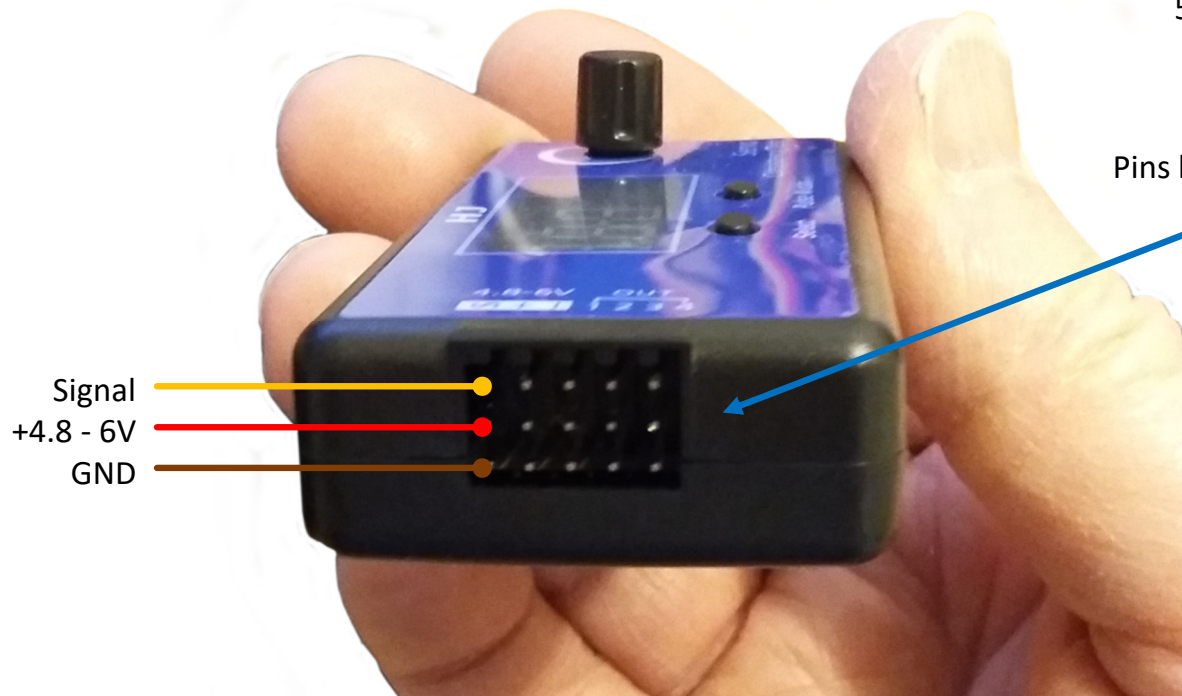
50Hz



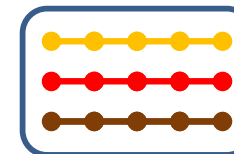
125Hz



250Hz



Pins have common connections



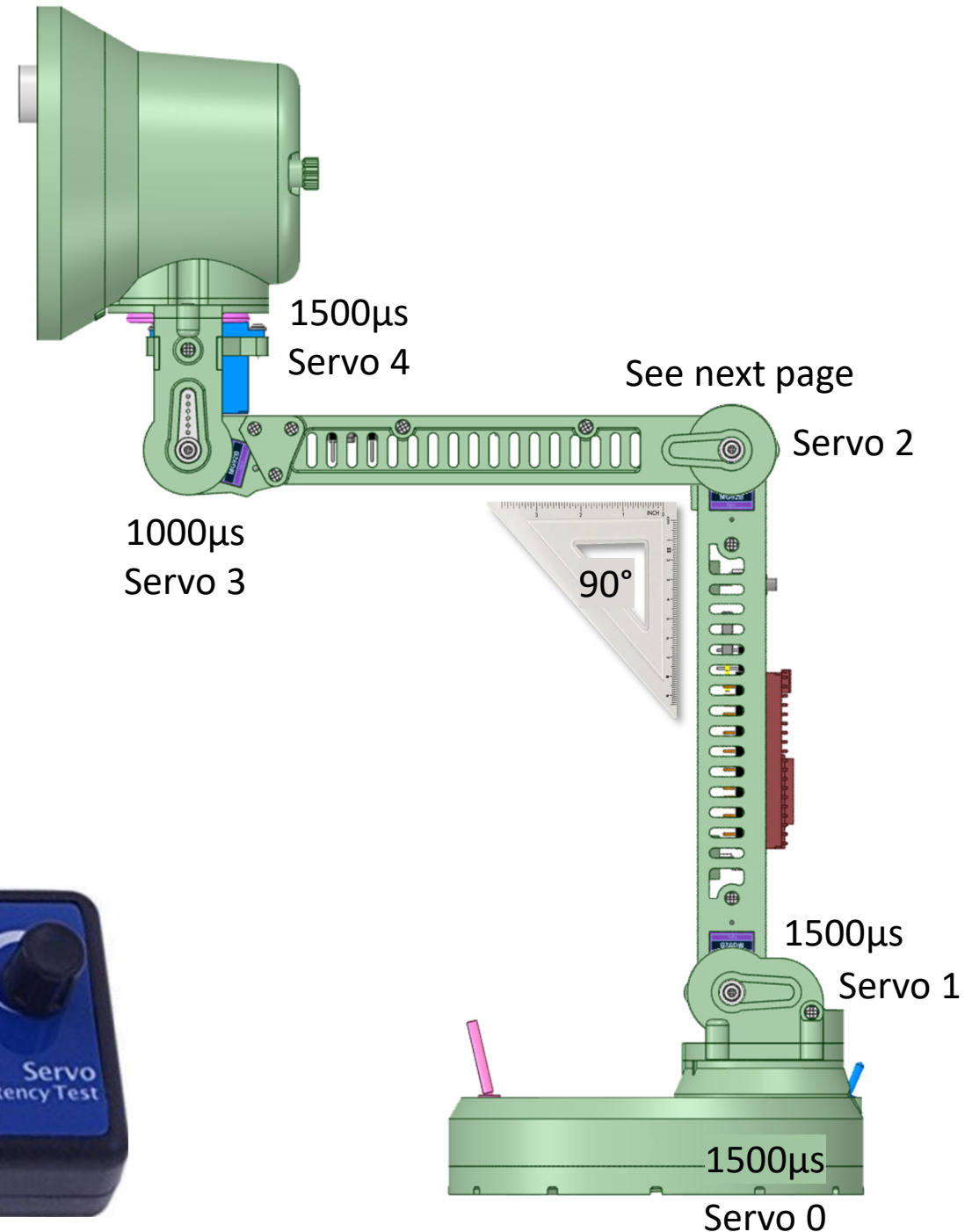
Signal
+4.8 - 6V
GND

Course Calibration

At this stage you are in the process of building the robot and attaching lever arms to the servos. These diagrams show you what PWM values to use, with a Servo Consistency Test box, to position each servo for the fitting of its lever.

Note that all of the joints are at right-angles, when fitting the servo levers.

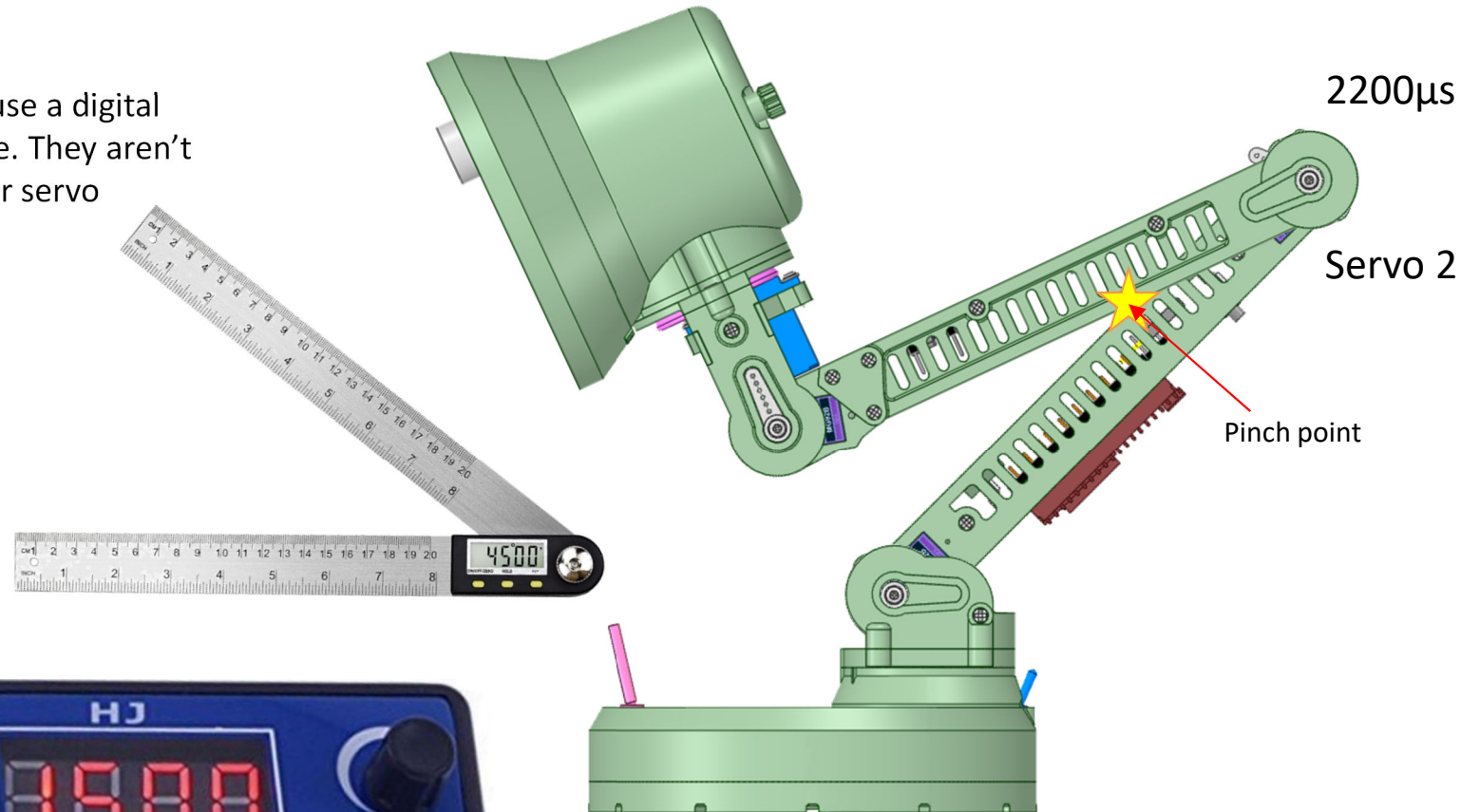
If you don't have a Servo Consistency Test box, then you can flash code into the ESP32 micro, and use the supplied Windows 16-channel app to set the PWM values. Then attach the lever arm to the servo as shown, to best achieve the centre position. The splines on the drive shaft may compromise your ability to achieve this exactly (remember the $\pm 9^\circ$), so find the best position.



Course Calibration

For servo 2 we adopt the same method; but in this case we close up the joint, so that it is at its minimum angle, resting on the cross brace. There is a pinch point where the cross member touches the lower arm. Use this position.

Out of interest, for angles you can use a digital protractor, if you have access to one. They aren't very dear, and can be very useful for servo projects like this.



2200µs

16-channel Controller App

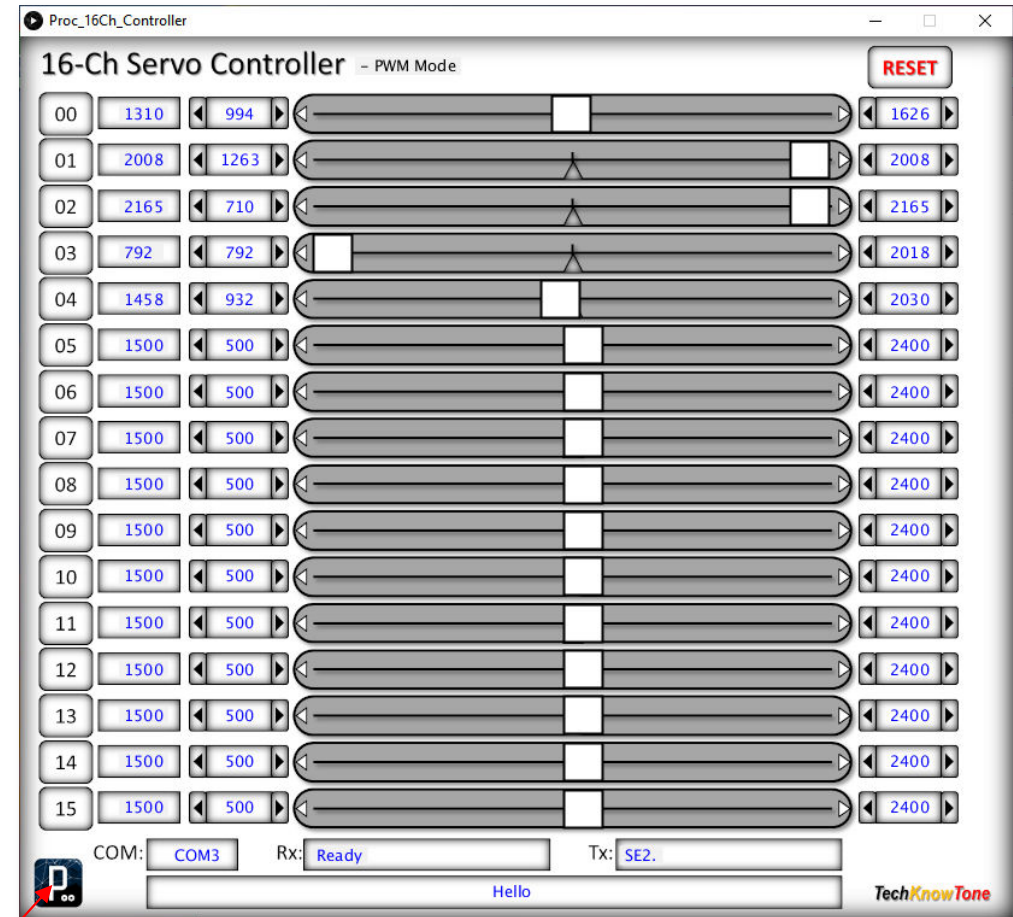
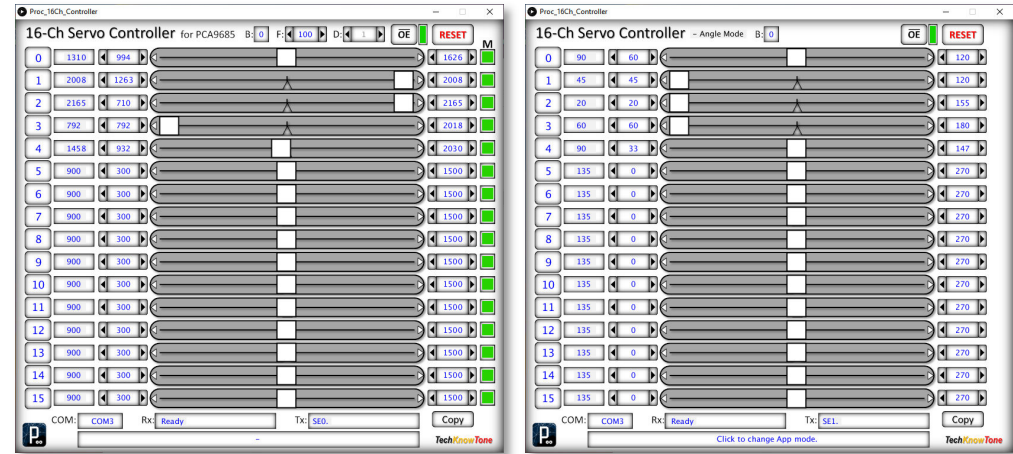
This Windows app communicates with your ESP32 micro, using the USB serial port, or over Wi-Fi. When you first connect it to the micro, it will request default values from it, which are then used to position the first five sliders (for PIX), to represent those default values.

The app can be used in several modes. By default it is set up for use with PCA9685 16-channel I2C controller boards; but for the PIX project you can switch to PWM mode by simply clicking on the bottom left 'Processing' logo. Initially you will want to use the app in 'PWM Mode', but later it may be useful to use 'Angle Mode'.

From left to right, the first column is 16 ON/OFF buttons. When a channel is ON the slider will be coloured red, and values will then be sent to the micro as the slider is moved. The 2nd column represents the values being sent; which will change as you move the sliders. The 3rd column represents the lower limit of the slider. Clicking in the field will set this to the current value, and is a useful way of limiting the slider range. Then you have the click-n-drag sliders themselves. In the 5th column we have the slider upper limits. With the lower and upper limits set, you can then freely move the slider, to swing a servo through its desired range of movement.

At the bottom of the app you will see fields for the USB COM port, receive (Rx) and transmit (Tx) fields, and below them a useful Help field.

If a COM port connection has not been established, the COM field will be in red. To force a connection attempt, simply click-left in the COM field. If a connection is established, the text will turn blue and the COM port number will be displayed. Click-right to cancel a connection, as you may want to download code to your micro over the same USB link.



Fine Calibration – Servo 0

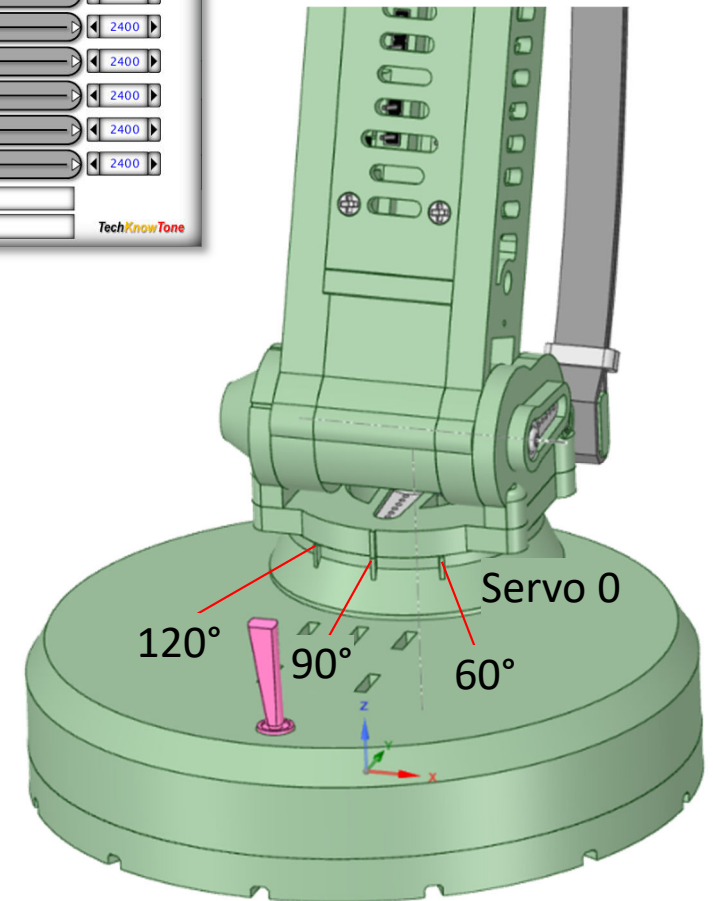
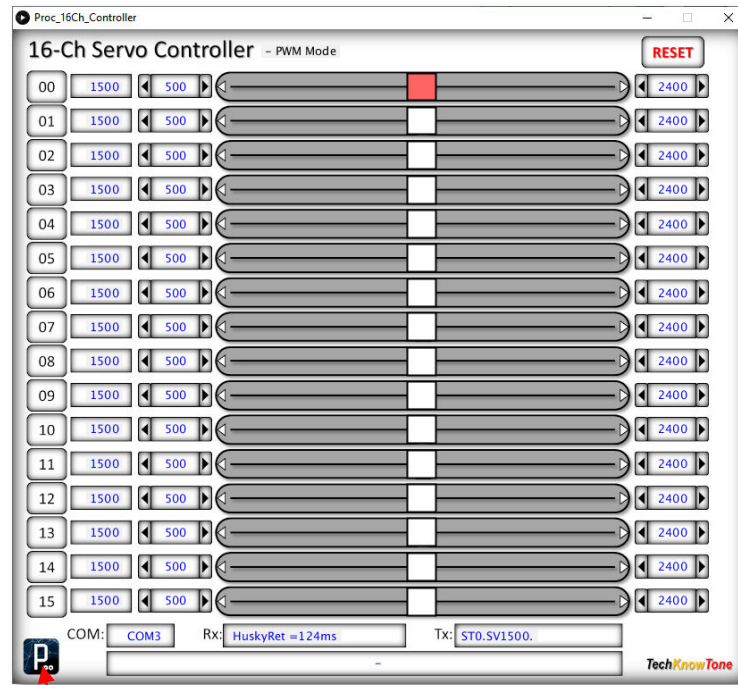
Servo 0, mounted in the base, swings the arm from side to side. The 3-D models include three calibration marks; one for forward (90°), one 30° to the left (60°), and one 30° to the right (120°).

By convention, in the PIXAR code their corresponding PWM values are defined as follows:

S0_60	1626	30° to the left
S0_90	1310	forward
S0_120	994	30° to the right

Use the 16-channel Servo Controller app, in PWM mode, to move channel 0 and determine these PWM values for your servos. I have included mine here for reference, and included them in the ESP32 code.

Note that when you invoke the 16-channel app it will start up in PCA9685 mode, by default. You need to switch it to PWM mode by clicking on the Processing logo button, bottom left. Successive clicks on this logo will change the mode of the app; there are 4 modes in total; which can be used in other projects too.



Fine Calibration – Servo 1

Servo 1, mounted at the bottom of the lower arm, swings the arm forwards and backwards. The 3-D model includes physical stops, at the rear, to prevent the arm from going backwards more than 45°. This position can be used for one of the measurements. The other two angles are set and measured using a 45° set square, as shown.

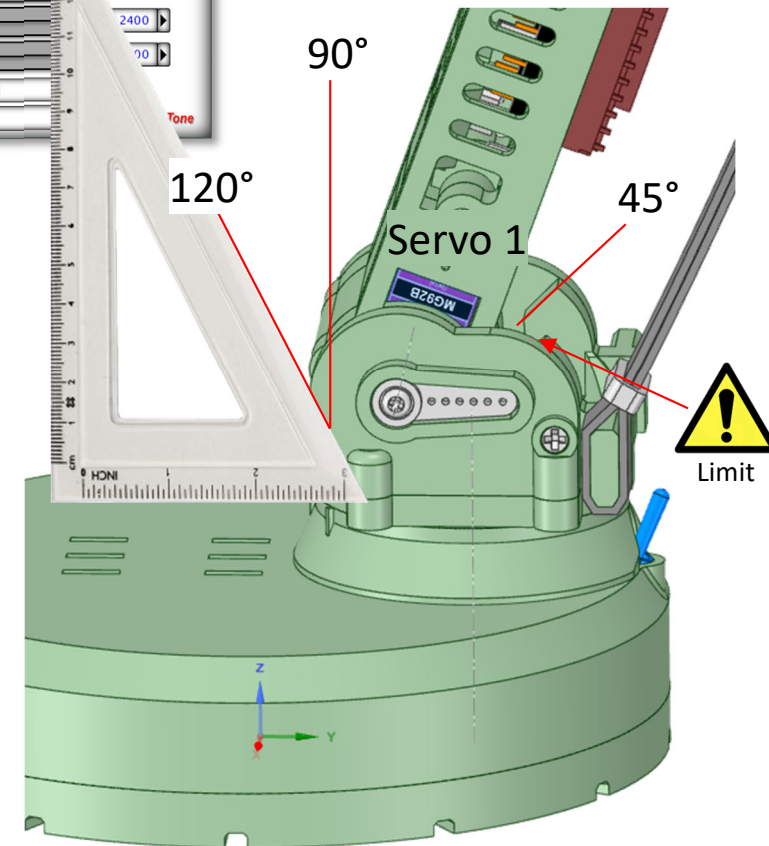
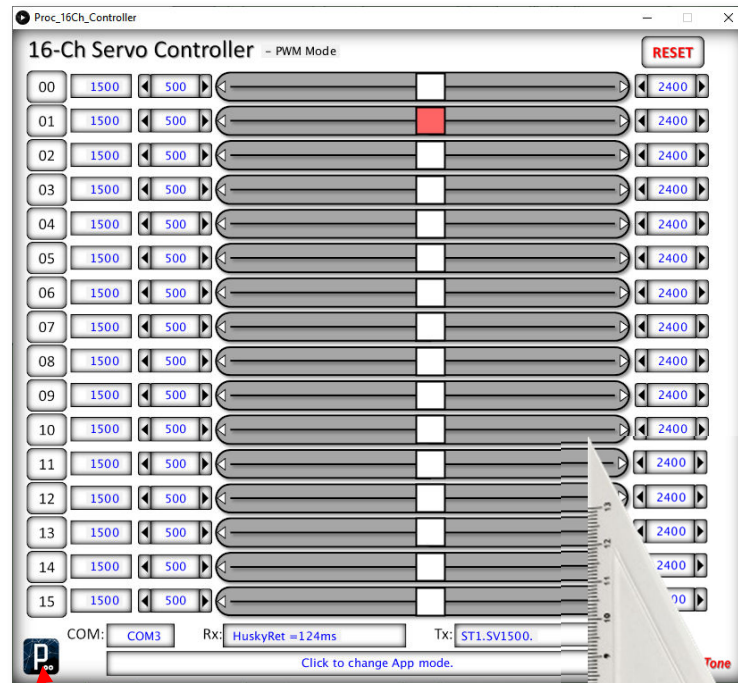
In the code their corresponding PWM values are defined for servo 1 as follows:

S1_45	2008	45° leaning backwards
S1_90	1592	upright, vertical
S1_120	1263	30° leaning forwards

Use the 16-channel Servo Controller app to move channel 1, to determine the PWM values for your servo. I have included mine here for reference, and included them in the code.

Note: that when you invoke the 16-channel app it will start up in PCA9685 mode, by default. You need to switch it to PWM mode by clicking on the Processing logo button, bottom left. Successive clicks on this logo will change the mode of the app; there are 4 modes in total. Which can be used in other projects.

At the 45° angle, the servo PWM must not be set so high a value that it is driving the arm into the physical stop limit; but just very close to it. By using an external power supply, with current sensing, you will see the current rise rapidly when the servo effectively hits the end stop, so you can back it off slightly from that position.



Fine Calibration – Servo 2

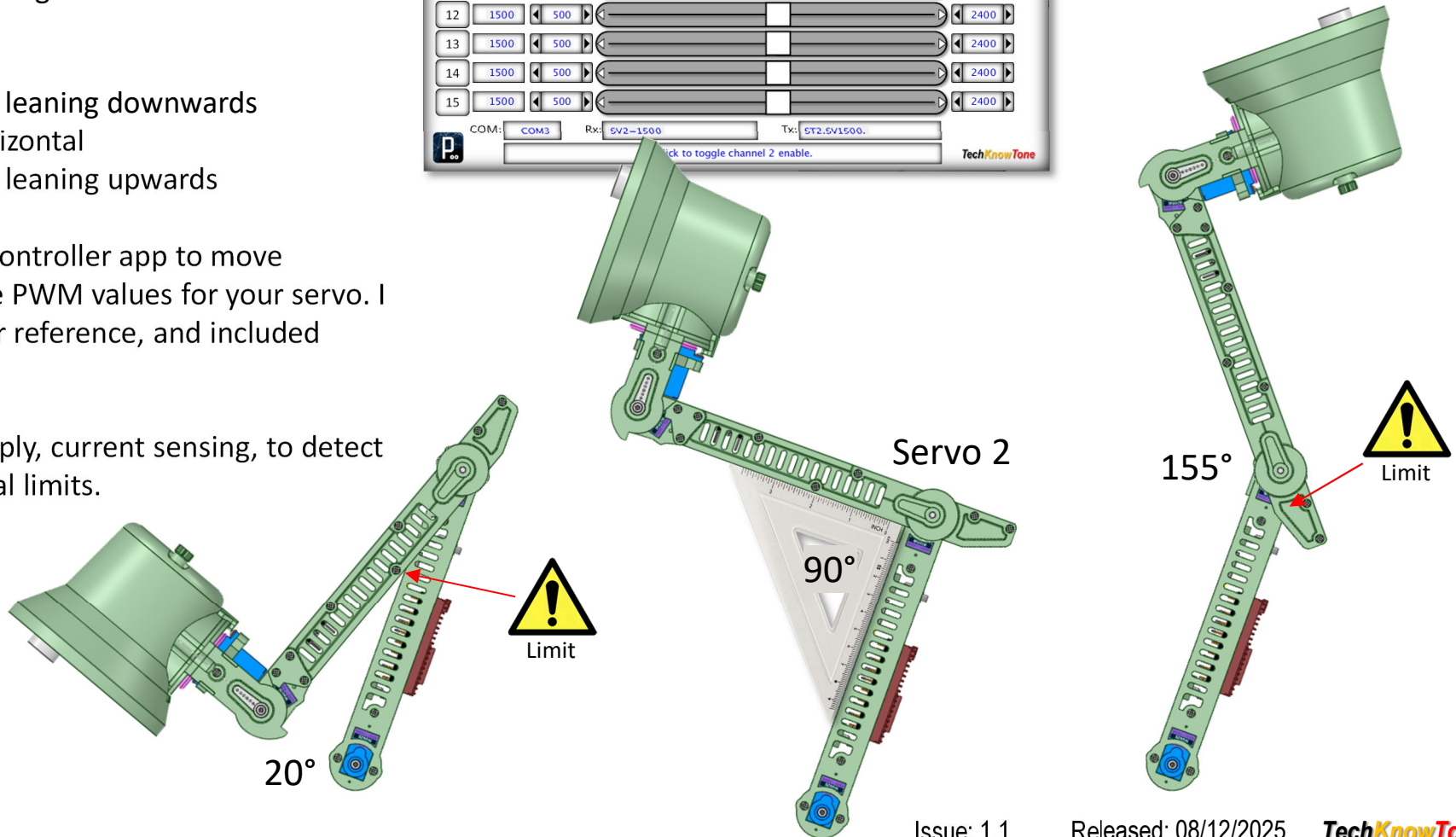
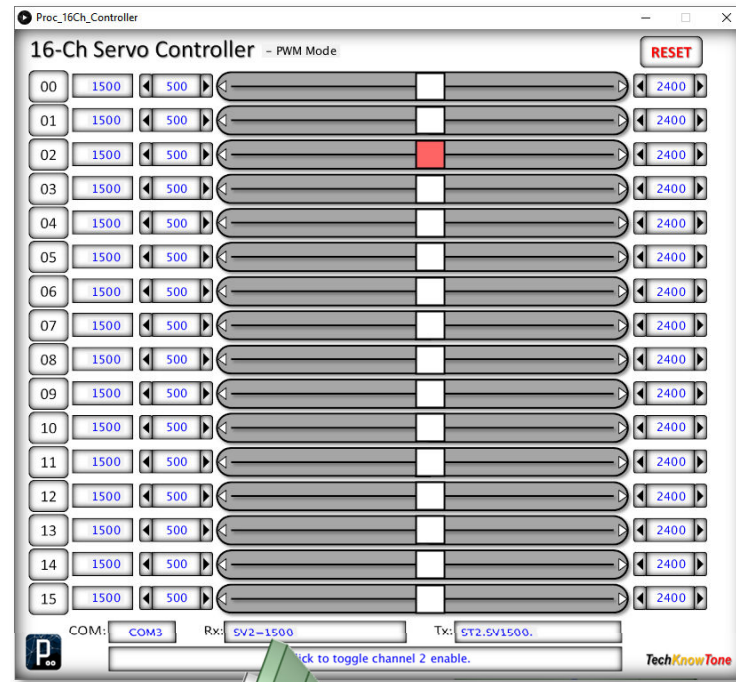
Servo 2, mounted at the top of the lower arm, swings the arm upwards and downwards. The 3-D model includes two physical stops, one at the rear of the forearm, and one below it, to limit the arm movement. These positions can be used for two of the measurements. The other 90° angle is set and measured using a set square, as shown.

In the code their corresponding PWM values are defined for servo 2 as follows:

S2_20	2165	20° leaning downwards
S2_90	1428	horizontal
S2_155	710	65° leaning upwards

Use the 16-channel Servo Controller app to move channel 2, to determine the PWM values for your servo. I have included mine here for reference, and included them in the code.

Use the external power supply, current sensing, to detect and avoid exceeding physical limits.



Fine Calibration – Servo 3

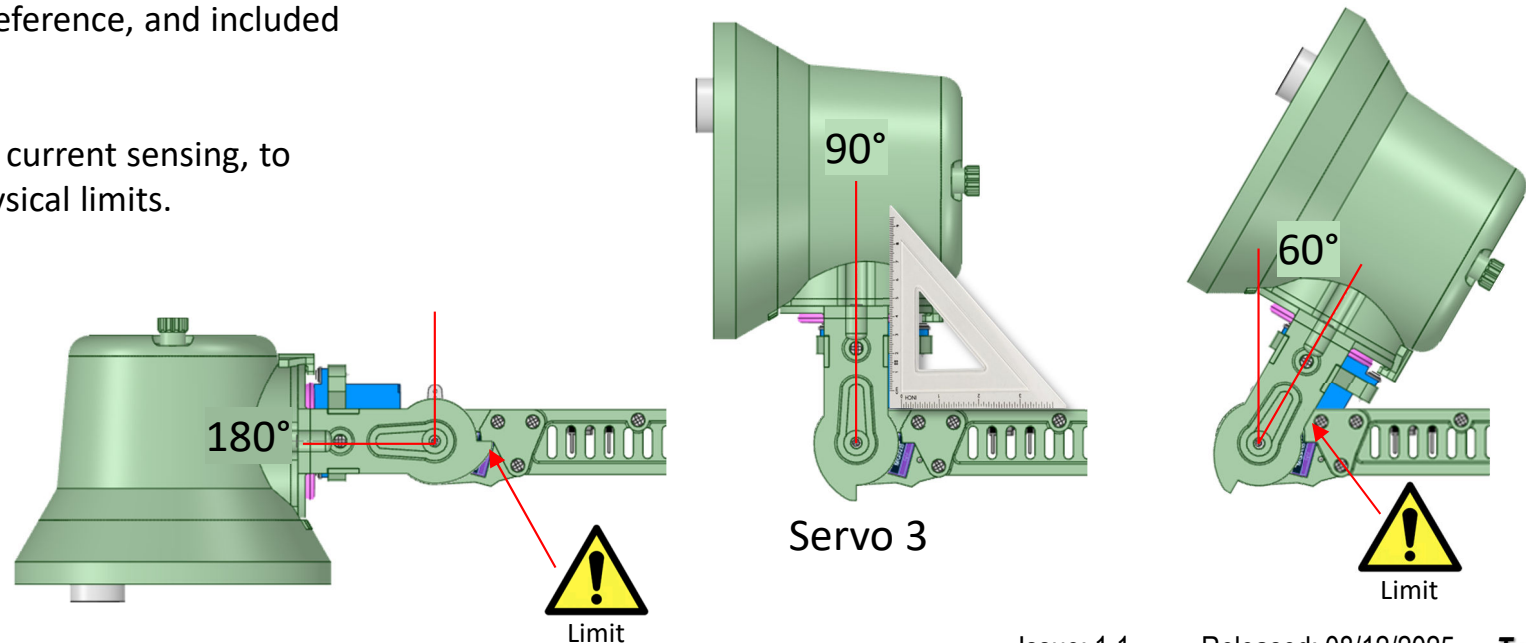
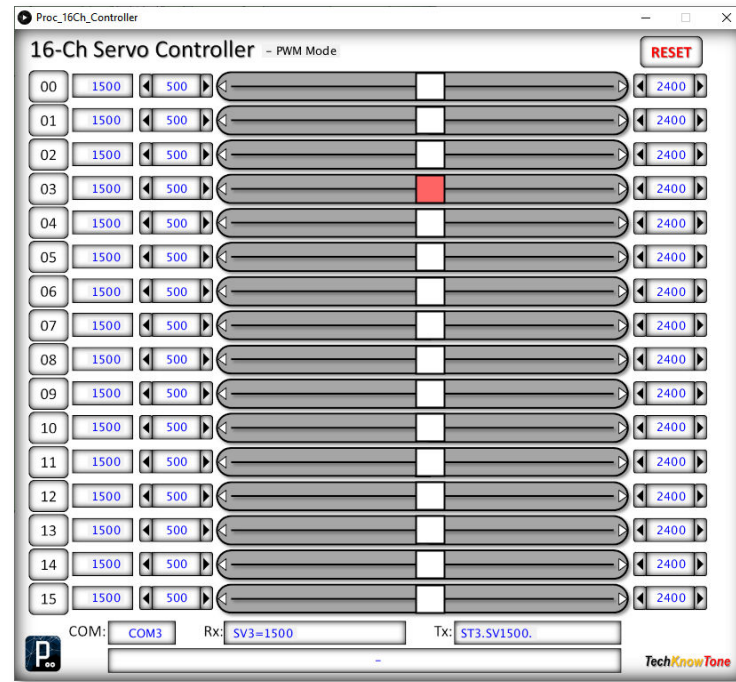
Servo 3, mounted at the end of the forearm, swings the lamp upwards and downwards. The 3-D model includes two physical stops, one at the rear, and one below it, to limit the tilt movement. These positions can be used for two of the measurements. The other 90° angle is set and measured using a set square, as shown.

In the code their corresponding PWM values are defined for servo 3 as follows:

```
S3_60      792      30° leaning backwards
S3_90     1120     vertical, looking forwards
S3_180    2018     90° looking downwards
```

Use the 16-channel Servo Controller app to move channel 2, to determine the PWM values for your servo. I have included mine here for reference, and included them in the code.

Use the external power supply, current sensing, to detect and avoid exceeding physical limits.



Fine Calibration – Servo 4

Servo 4, mounted at the end of the forearm, above servo 3, turns the lamp side to side, left to right. The 3-D model includes a physical stop, which rotates with the lamp, to limit its movement. These positions can be used for two of the measurements. The other 90° angle is set using visual judgement.

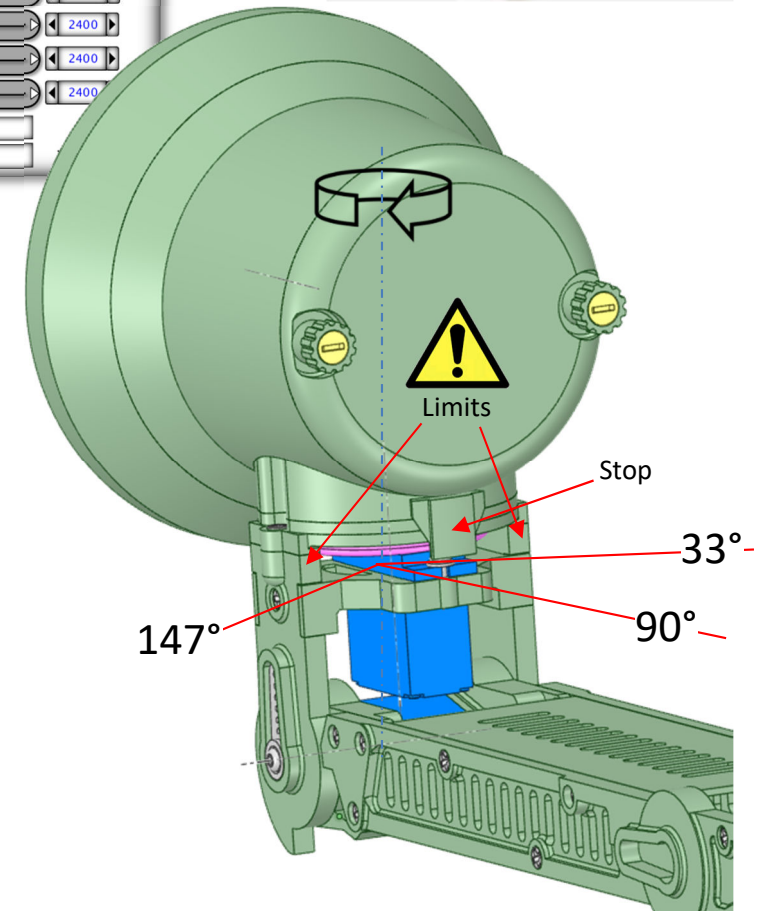
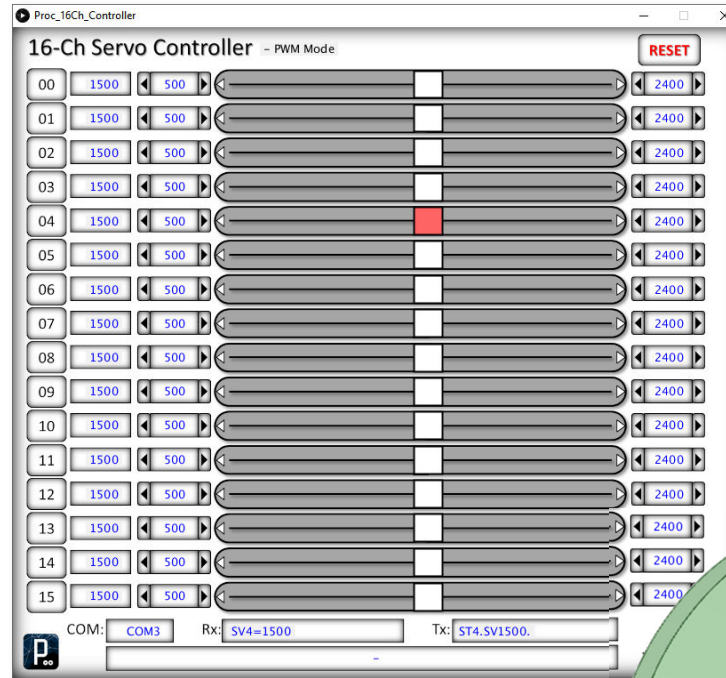
In the code their corresponding PWM values are defined for servo 4 as follows:

S4_33	2030	57° turned to the left
S4_90	1458	central, looking forwards
S4_147	932	57° turned to the right

Use the 16-channel Servo Controller app to move channel 4, to determine the PWM values for your servo. I have included mine here for reference, and included them in the code.

Note - you can turn on servo 3 and set it to the vertical position, to steady the lamp whilst taking these PWM readings.

Use the external power supply, current sensing, to detect and avoid exceeding physical limits.

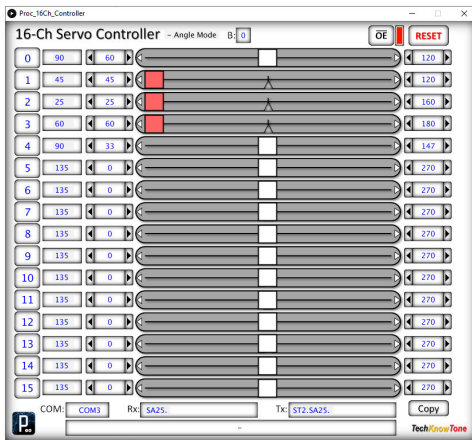
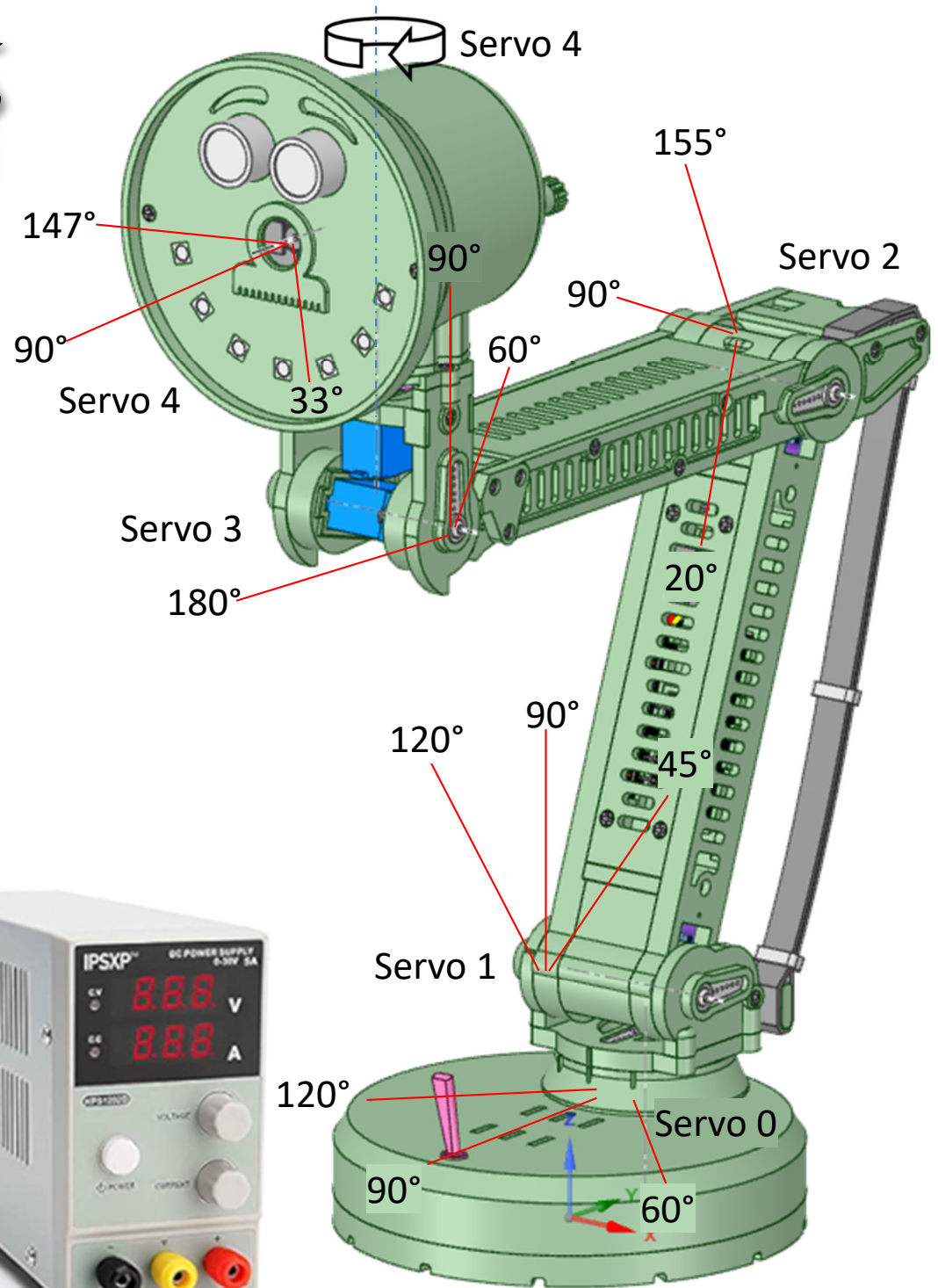


Angle Summary

This diagram summarises the range of angles defined for each of the five servos, and acts as a useful reference, when working out moves for PIX.

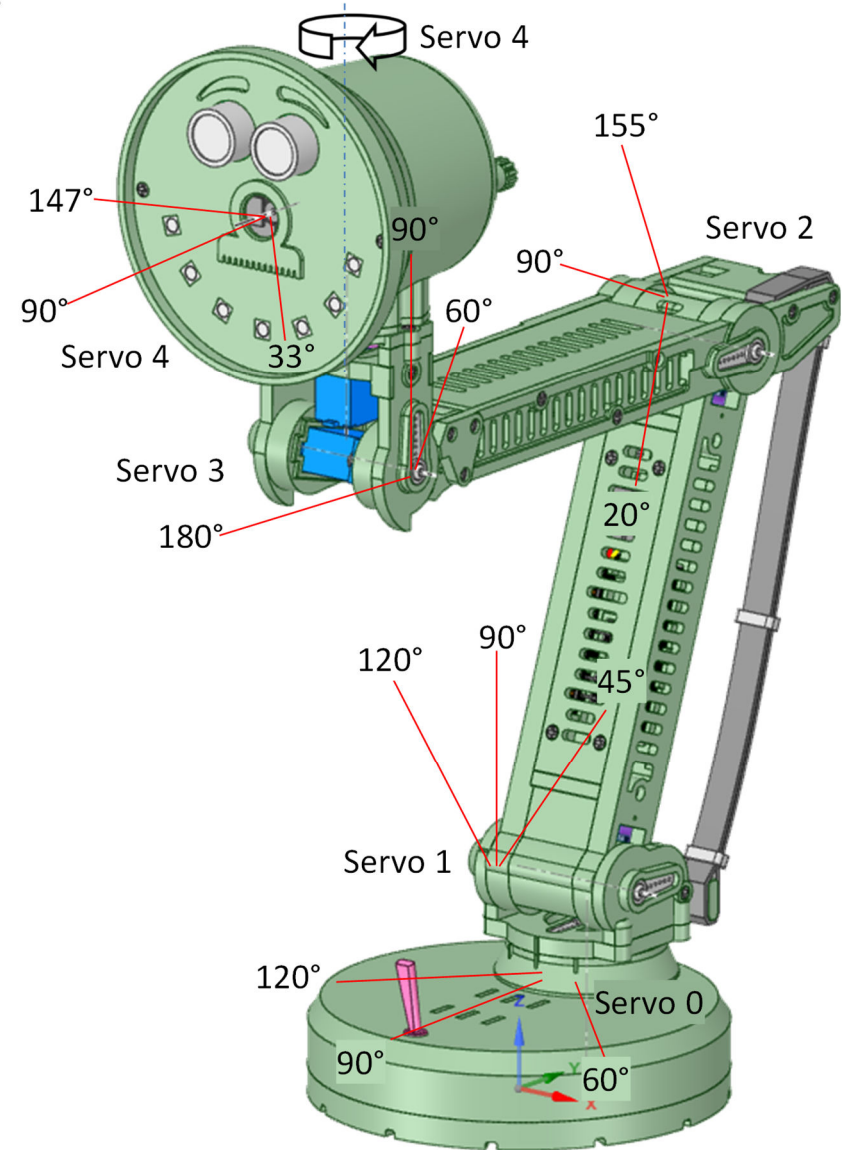
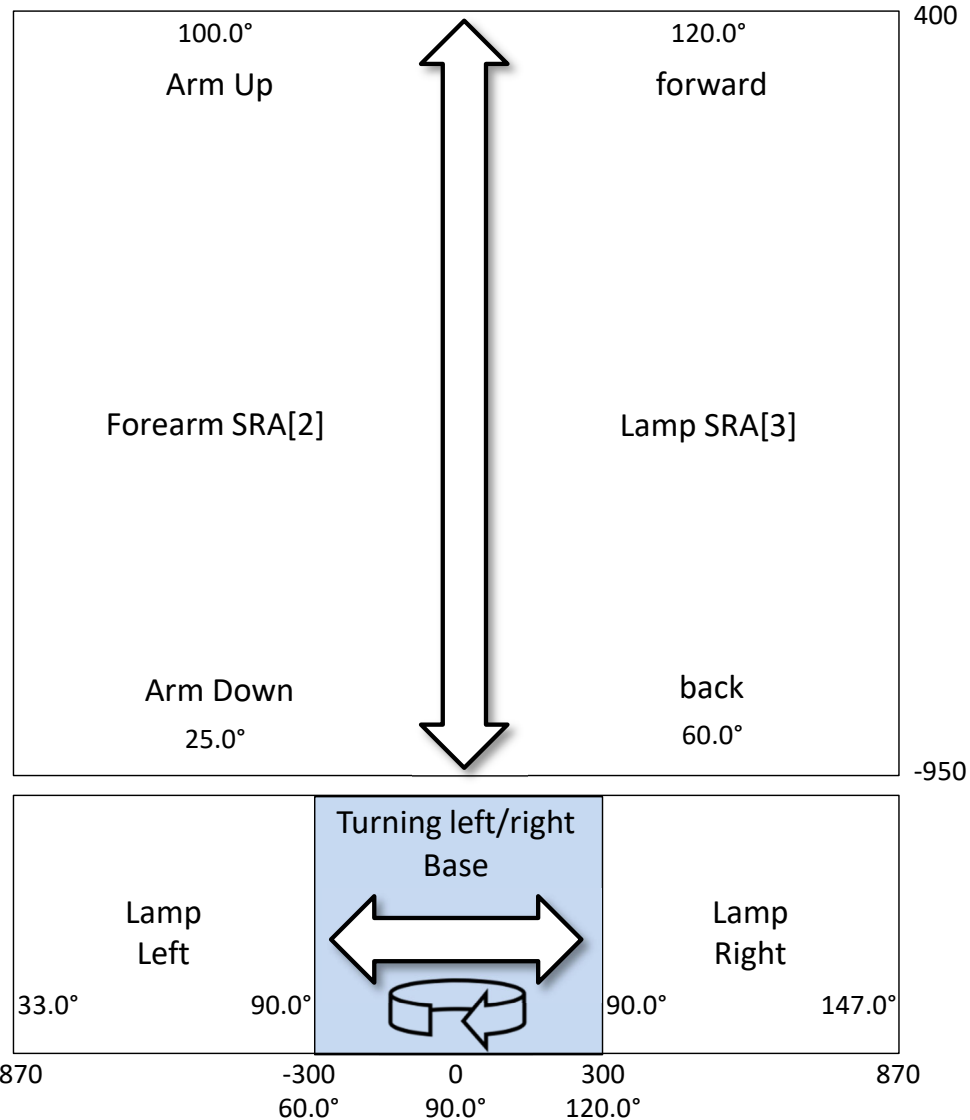
Note that the elastic tensioner works to not only assist with the lifting of the forearm, but also has a tendency to pull the lower arm backwards, This makes it necessary to raise the forearm, to reduce the tension, when wanting to move the lower arm forward. Otherwise, it will fail to do so.

I used the Windows app in angle mode, along with an external DC power supply, to determine the best range of angle for ease of moving the forearm and lower arm servos. These ranges were then built into the code as limit ranges.



View Space

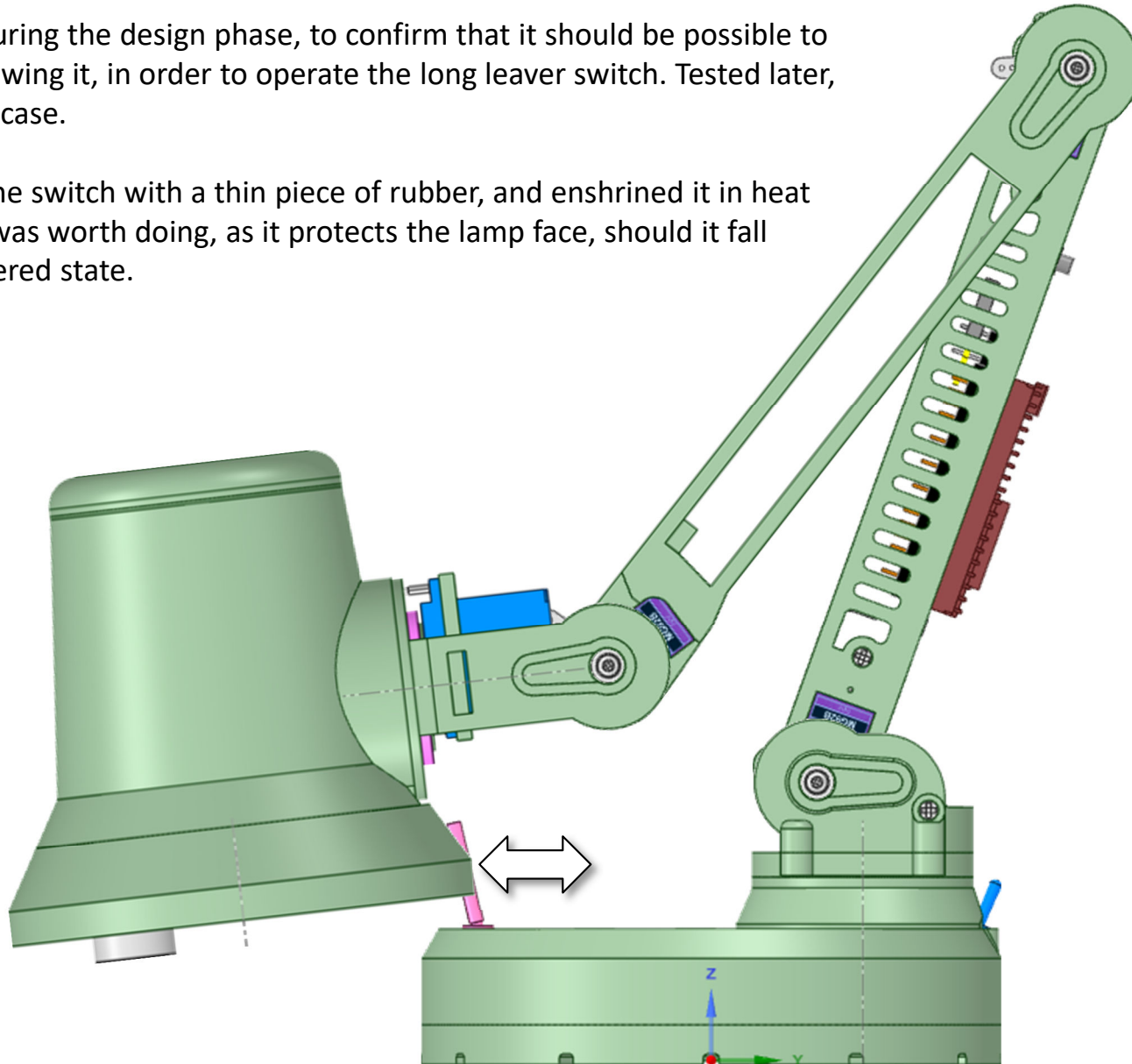
The view space map can be used to determine angles for the robot, where it has detected a face. Servo 0 and 4 map the turning of the base and head into a horizontal angle. Servo 2 and 3 map the vertical angle. Servo 1 is ignored, as its contribution to height is low. The robot control system aims to keep the lamp looking horizontally. The location of faces fades over time, but are refreshed. All internal angles are in x10 integers, to improve speed, whilst not compromising accuracy.



Leaver switch operation

I used this diagram during the design phase, to confirm that it should be possible to move the lamp, and swing it, in order to operate the long lever switch. Tested later, this proved to be the case.

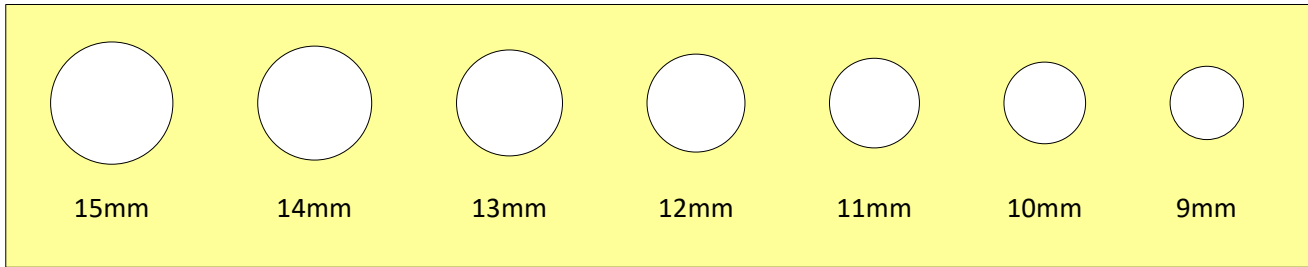
Note that I covered the switch with a thin piece of rubber, and enshrined it in heat shrink sleeving. This was worth doing, as it protects the lamp face, should it fall forward in an unpowered state.



HuskyLens Camera field of view

I wanted the camera to see the whole image being presented to it, and not be obscured by the size of the whole in the face plate. Therefore there is a Lense Disc which was sized accordingly, and then glued into the face plate.

The method I used to get this centre hole as small as possible, was to print strips of paper with different sized holes in them, and place them over the opening. There by reducing the size of the hole until it was seen to obscure parts of the cameras display image.



You need to have the HuskyLens camera powered up for this, and the rear cover of the lamp housing removed, in order to view the screen on the rear of the camera. The hole size I came up with was 12 mm, which suggests that the view angle of the camera is in the region of 60°.

In the end I included a moustache shape for the Lense Disc, and, as it is a glued insert, you could include whatever design you wanted for this.

