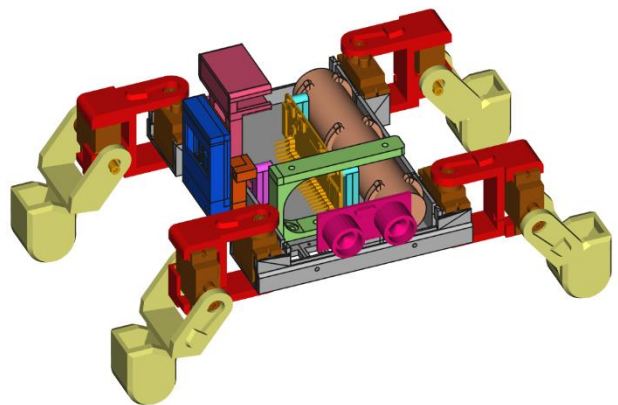
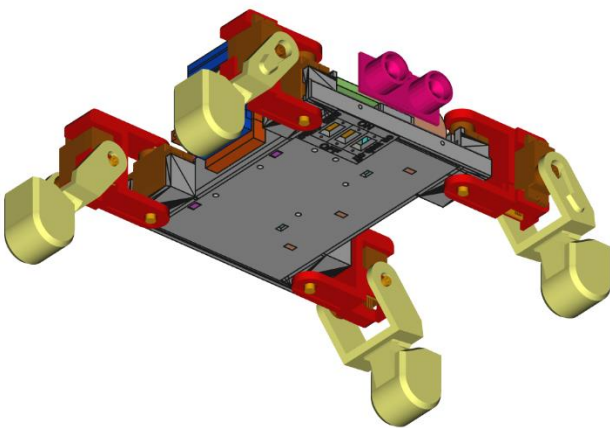
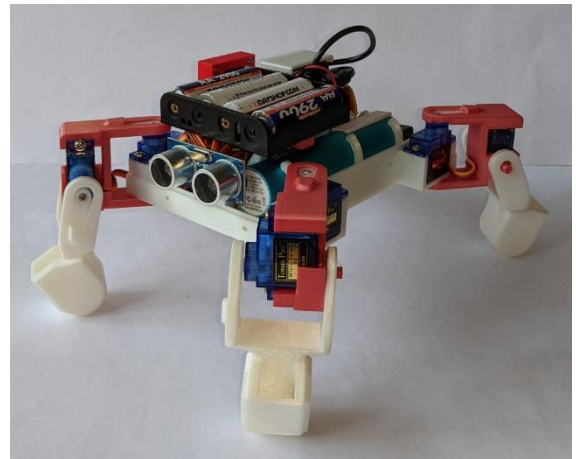
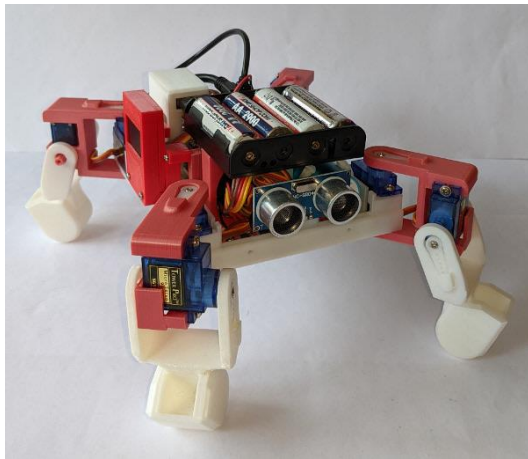


Raspberry Pi Zero 8DOF walking robot build & usage notes



version 1.0

Table of contents

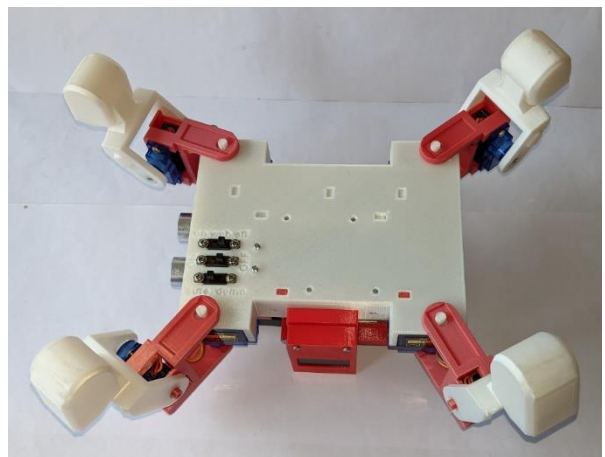
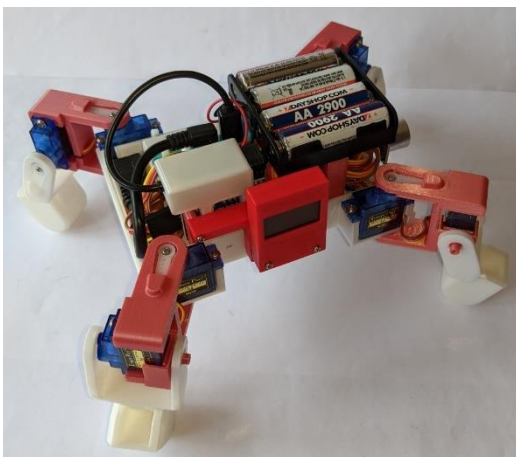
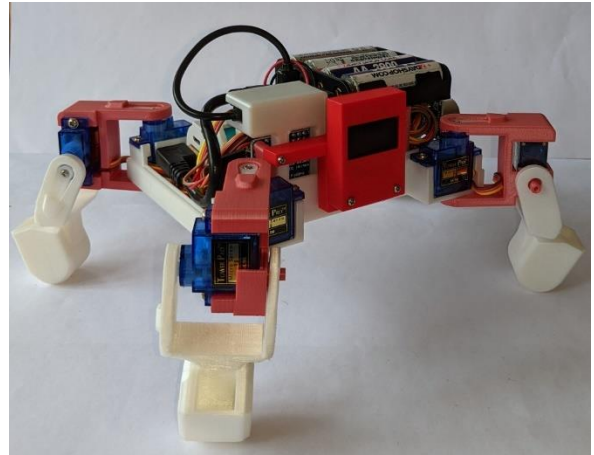
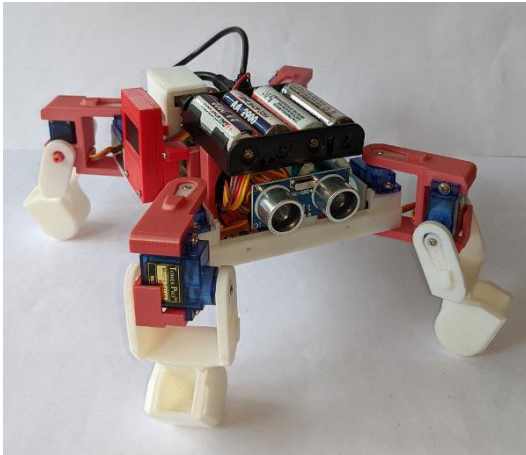
Introduction.....	3
8DOF quadrabot build notes	4
Leg assembly/installation.....	4
Raspberry Pi Zero installation	6
Slide switch installation	6
Ultrasonic sensor installation	7
4AA battery holder installation.....	8
PCA9685 PWM module installation	8
WiFi controller/gamepad 'dongle' holder	9
OLED mounting bar installation.....	9
OLED housing installation	10
22mm battery bank installation	10
Component connections.....	11
PCA9685 PWM module and OLED I ² C connections	11
Slide switch GPIO connections	11
Ultrasonic sensor GPIO connections.....	11
Test and 'production' control software.....	12
Test routines	12
'Production' control software	12
Gait analysis and implementation	13
Quadrabot operational usage notes.....	14
Quadrabot start-up steps:.....	14
Slide switch control options	14
Wireless controller mode:	15
Web mode	15
Quadrabot shut-down steps:	19
Appendix A: 3D printed components	20
Appendix B: Servo movement values	22
Appendix C: control/storage file details	23

Introduction

This Raspberry Pi controlled 4-legged, 8 degrees-of-freedom (8DOF with 4 legs, each with hips and knees) walking robot was 'inspired' by the various Arduino MiniKame robots that can be found on the web.

The main 3D printed body is a new custom design that accommodates:

- a Raspberry Pi Zero,
- a PCA9685 PWM control module for the 8 servos, plus
- separate power sources for the Raspberry Pi and the servos, as well as
- an ultrasonic sensor
- a holder for a wireless controller/gamepad USB dongle
- an OLED, and
- 3 slide switches on the underside of the body for on/off and operational mode selection.



Additional 'mount' components have been designed to attach the various components to the main body, a number of design tweaks were made to the leg and hip components so that they are more easily 3D printed, and 'channels' were incorporated into the hip and leg prints so that the servo wiring could be threaded internally through to the PCA9685 in the main body.

It should be noted that the overall assembly has been evolved over an extended period of time with the wireless controller 'dongle' holder and OLED 'bolted on' to an earlier overall arrangement, so some 'consolidation' of the 3D print designs would certainly be possible!

Details for all the individual 3D printed components are provided in **Appendix A: 3D printed components** and the 3D print designs can be downloaded from the Prusa web site [here](#), plus all the software and related materials can be downloaded from [here](#).

8DOF quadrabot build notes

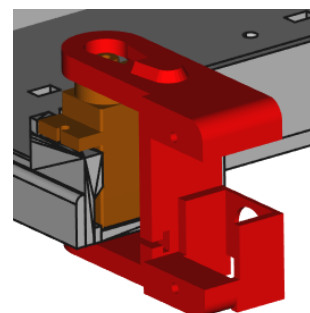
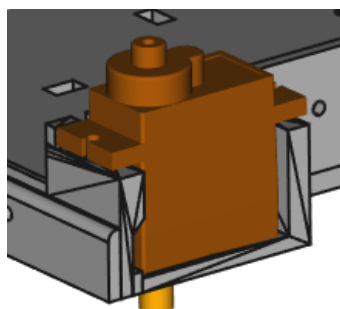
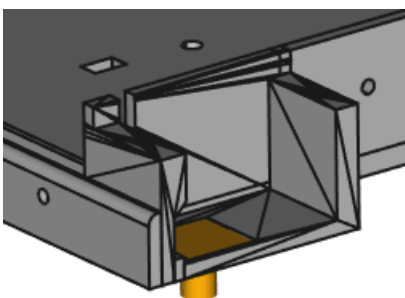
Leg assembly/installation

Each leg assembly uses:

- 2x SG90 servos
- 1x leg shaft (download file: *m-leg-shaftsx6.stl*)
- 1x hip fame (download file: *m-hip_x2_repaired03_mirrored.stl* - mirrored version to suit)
- 1x hip shaft (download file: *m-hip-shaftsx6.stl*)
- 1x leg frame (download file: *m-leg_x2_angled02_mirrored.stl* - mirrored version to suit)

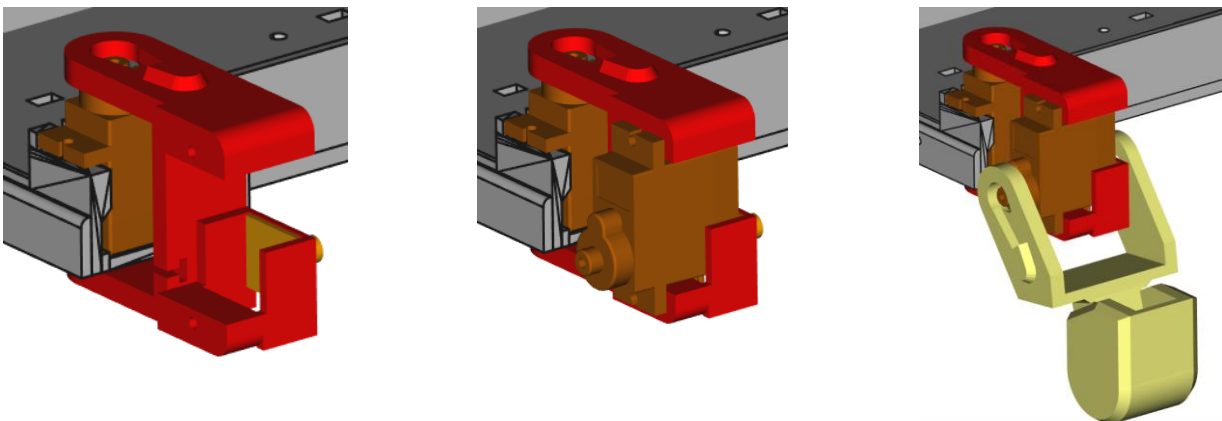
All four legs are assembled in a similar way with the images below illustrating the assembly steps for the back right leg with the first three images below showing the:

- the leg shaft inserted into the back right servo pocket of the main body
- a servo then inserted which is secured with M2 self-tap screws supplied with the servo
- a hip frame connected to the servo: this uses the single arm horn supplied with the servo inserted into the recess on top of the hip frame and pressed onto the spline shaft of the servo (this will in due course be secured with the small self-tap screw supplied with the servo once the servo calibration positioning has been carried out – discussed later).

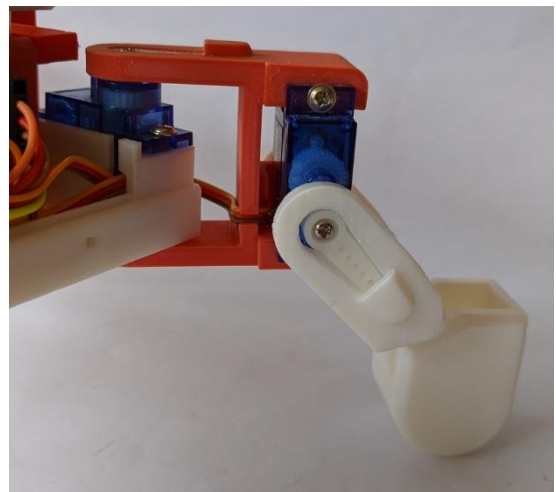


The next three images below then show:

- the hip shaft inserted into the hip frame
- the 2nd servo inserted into the hip frame which is secured in the frame with M2 self-tap screws supplied with the servo
- the leg frame connected to the servo: this uses the single arm horn supplied with the servo inserted into the recess on the side of the leg frame (and as for the 1st servo this will in due course be secured with the small self-tap screw supplied with the servo once the servo calibration positioning has been carried out).



The wiring for each servo is 'threaded' through the various slots and openings in the hip frame and the servo pocket in the main body, which can be seen in the two images of either side of the back right leg shown below. The wiring should be 'looped' at certain points to allow for the movement of the leg components.

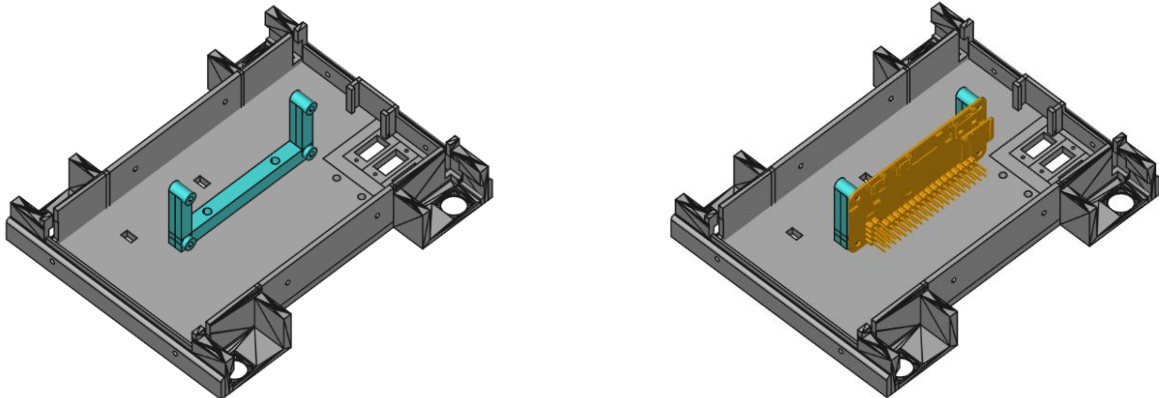


However, it should be noted that it is recommended that the final step of attaching the leg frames to each overall leg assembly is not done at this stage but is deferred until the overall quadrabot assembly is completed since handling for the rest of the build will be easier without them installed.

Raspberry Pi Zero installation

The Raspberry Pi Zero mounting bracket (download file: *PiZ_v_holder03.stl*) is inserted into the main body, as shown in the image below left, and secured with 2x 8mm long M3 pan head screws that self-tap into the base of the main body.

A Raspberry Pi Zero (Mk2 suggested) with its SD card inserted and already flashed (latest Bullseye OS recommended) is then attached to the mounting bracket using 4x self-tap 6mm long M2 flanged pan head screws, as shown in the image below right.



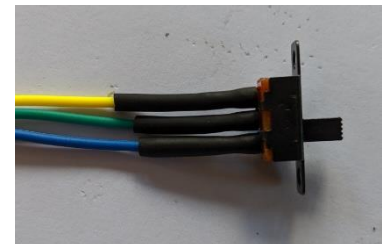
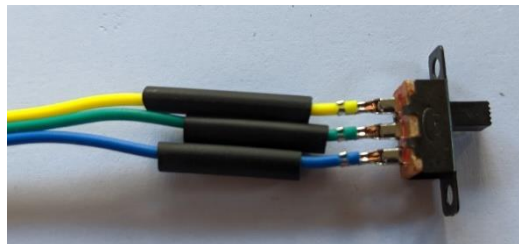
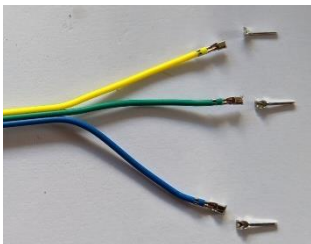
Slide switch installation

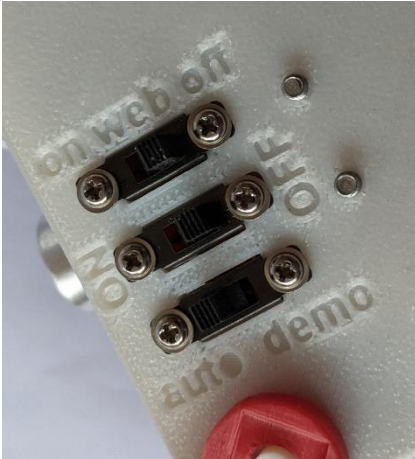
The three slide switches (readily available 5V 0.3A mini SPDT as shown on the right) have wires added for 3V3, signal and GND connections as shown in the three images below.

The 1st image shows how a jumper lead's male connector can have its end snipped off to leave a small opening that (by chance!) is a snug push fit onto the terminals of the switch.

The 2nd image shows three leads installed where each should be carefully soldered in place since the push fit is not sufficient for a good electrical connection, and short sections of heat shrink insulating tubing pre-installed.

The 3rd image then shows the insulating tubing shrunk in place.





Each of the switches are screwed in place into the underside of the main body, as shown on the left, using self-tap 6mm long M2 flanged pan head screws.

The orientation of each switch should be so that whichever leads are 'designated' to be the 3V3 connection are positioned as follows:

- on web: 3V3
- ON: 3V3
- Auto: 3V3

Once the switches are installed, the 3V3 leads for each switch should be connected together so that only one connection is needed to the Raspberry Pi GPIO pins – and

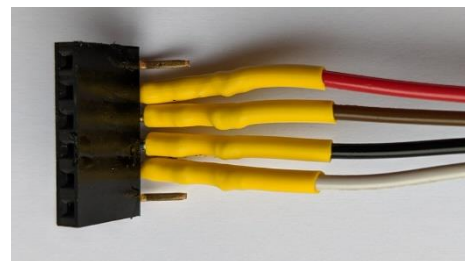
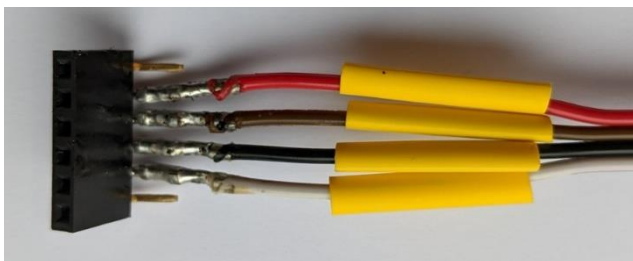
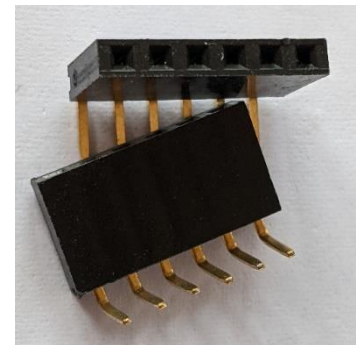
the same carried out for the GND connections for each switch. The middle lead from each switch is the signal connection which are each connected to a separate GPIO pin.

Ultrasonic sensor installation

The installation of the ultrasonic sensor is achieved by inserting the sensor's four connectors (VCC, Trig, Echo, GND) into the middle four holes of a 90° adaptor, examples shown on the right, which in turn is a slide fit into the slot at the front of the main body 3D print, so that the adaptor's connectors are on the base of the main body pointing towards the back.

Using a similar technique as used with the slide switches, snipped off connectors of male jumper leads can be pushed onto the middle four connectors of the adaptor and soldered in place, with short sections of heat shrink insulating tubing pre-installed, as shown below left.

Shrinking the insulating tubing then provides secure and insulated wired connections as shown below right which can then provide connections through to the Raspberry Pi's GPIO pins.

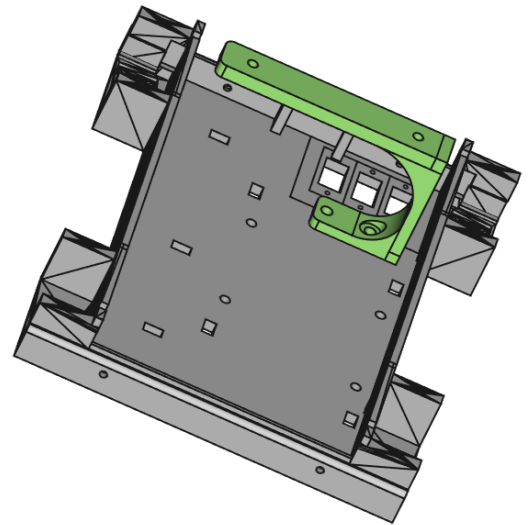


It should be noted however that the ultrasonic sensor should ideally be the newer HC-SR04P type that can be powered at 3.3V so that the output Echo signal is at a safe 3V3 for the Raspberry Pi. However, if only an older 5V powered sensor version is available then a voltage divider circuit can be built into the sensor's wiring harness using a 330Ω and a 470Ω resistor to reduce the Echo output voltage from 5V to 3V3 – but this does make the wiring bulkier in what is already quite a small amount of space!

4AA battery holder installation

The mounting bracket for the 4AA battery holder (download file: [4AA_battery_stand10.stl](#)), positioned as shown on the right, is secured to the main body using 2x 8mm long M3 pan head screws that self-tap into the floor of the main body.

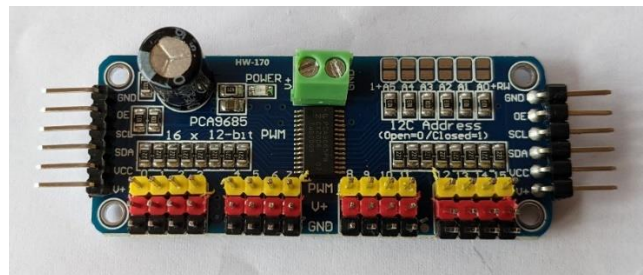
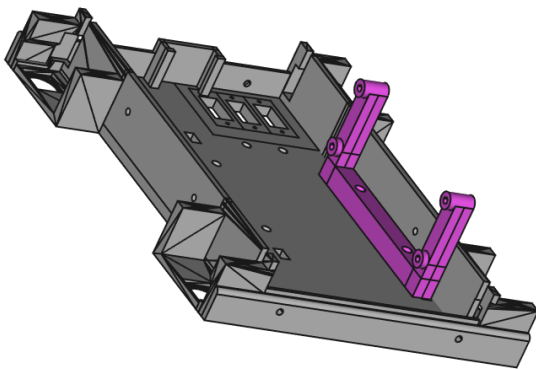
At a later stage of the assembly the 4AA battery holder can then be attached to the top of the bracket using 2x 6mm long M2 CSK head screws that self-tap into the bracket – although using just one CSK screw allows the battery holder to swivel to allow the wireless controller 'dongle' to be easily inserted into the USB cable.



PCA9685 PWM module installation

The mounting bracket for the PCA9685 PWM module (download file: [PWM_module_vertical_holder03.stl](#)) is inserted into the main body as shown in the image below left – although not yet secured with screws as this is done in the next section.

A PCA9685 that has connectors at both ends, as shown below right, can be secured to the bracket using 4x self-tap 6mm long M2 flanged pan head screws.

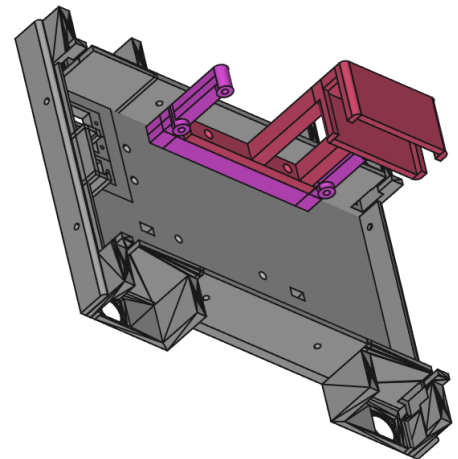


It should be noted however that, as a precaution, small insulating washers should be used between the pan head flanges and the module to avoid any possibility of the screws touching any electrical areas of the module.

In addition, care should be taken when soldering the connector headers to each end of the PWM module to ensure there is no possibility of any high resistance paths being inadvertently created between the SDA or SCL pins and ground due to solder or flux residue, since these pins must be 'pulled-up' by the Raspberry Pi in order that the I²C protocol operates correctly. A prudent check is to measure the voltage on these pins with the module powered, connected to the Raspberry Pi, and with the I²C interface active to ensure that they are both at 3.3V.

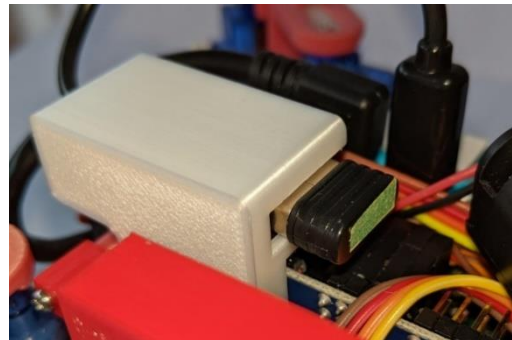
WiFi controller/gamepad 'dongle' holder

As shown in the image on the right, the holder for the female socket of a USB cable (download file: *quadrabot_dongle_holder05_rotx180.stl*), into which the 'dongle' for a wireless controller/gamepad is inserted, is positioned on top of the bottom bar of the mounting bracket for the PCA9685 PWM module. Both these fixtures are secured to the main body of the quadrabot with 2x 12mm long M3 pan head screws that self-tap into the base of the main body.



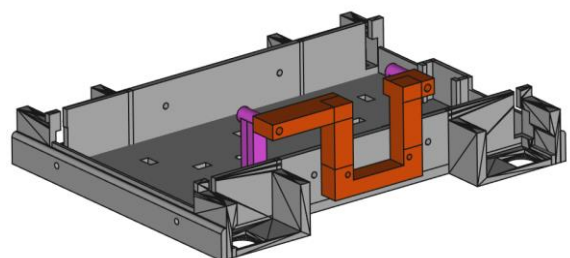
The image below left shows:

- the type of USB cable to be used with a female socket at one end, into which the WiFi dongle inserts, and a type B micro socket plug at the other end that inserts into the Raspberry Pi's USB socket, and below right;
- a close up of the overall assembly arrangement is shown.



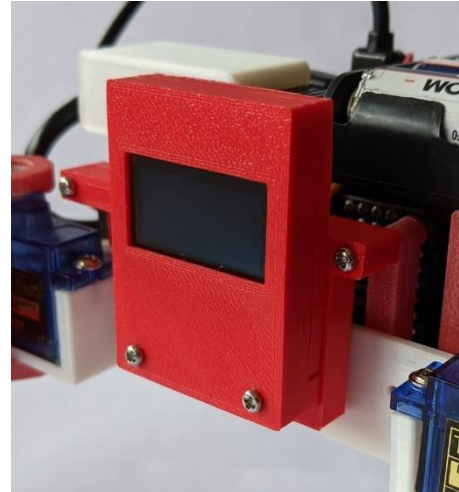
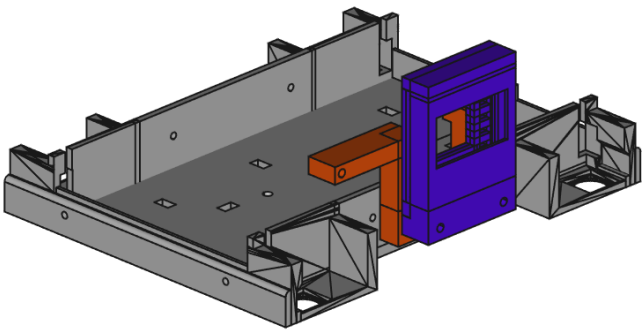
OLED mounting bar installation

The mounting bar for the OLED housing, shown in the image on the right, (download file: *OLED_mounting_bar01_roty90.stl*) is secured to the back of the mounting bracket for the PCA9685 PWM module with 2x 16mm long M2 pan head screws.



OLED housing installation

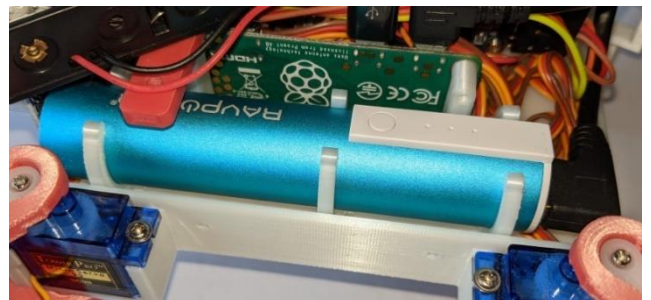
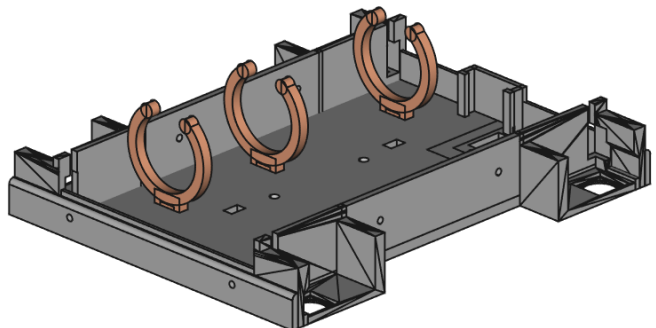
The two-piece OLED housing with a 128x64 pixel OLED installed, as shown in the images below (download file: *OLED1_flat08-front+back.stl*) is secured to the mounting bar for the OLED housing with 2x 16mm long M2 pan head screws. The two halves of the OLED housing are held together with 4x self-tap 6mm long M2 flanged pan head screws.



22mm battery bank installation

A set of 3 clamps for a 22mm diameter rechargeable battery bank (download file: *3x_battery_180deg_clamp01.stl*) are inserted into the main body as shown in the image on the right. These are just push-fit although adhesive could be used to secure them more permanently if they start to work loose.

The images below show the battery bank with a suitable short USB cable that has right-angled connectors at each end, and a close up of the completed installation.



Component connections

PCA9685 PWM module and OLED I²C connections

PCA9685 terminal	OLED terminal	GPIO connection
GND	GND	GND
OE	Not used	
SCL	SCL	GPIO #3 (SCL)
SDA	SDA	GPIO #2 (SDA)
VCC	VCC	3V3
V+	Not used	

The OLED is 'fed' from the pass-thru set of terminals on the PCA9685.

Details about the use of Pulse Width Modulation (PWM) to control the servos is provided in **Appendix B: Servo movement values**.

Slide switch GPIO connections

Left and right terminals as shown in the image on the right.

Slide switch	left terminals	Central signal terminal	right terminals
auto / demo switch	3V3	GPIO #10	GND
ON / OFF switch	3V3	GPIO #9	GND
on web / off	3V3	GPIO #11	GND



Ultrasonic sensor GPIO connections

Sensor terminal	GPIO connection
VCC	3V3 for HC-SR04P or 5V if older unit is used with a resistor voltage divider circuit
Trig	GPIO #17
Echo	GPIO #4
GND	GND

Test and 'production' control software

The quadrabot's eight servos move the four legs, each with a 'hip' and 'knee' joint where the movement mechanism or 'gait' is best described as a creeping style, and a spreadsheet has been used to tabulate how a defined set of leg movements are converted into a set of individual servo movements that are then used in the code to create 'walking' functions.

All the software and associated files used for testing and 'production' control of the quadrabot, listed in **Appendix C: control/storage file details**, are available to download from **this GitHub repository**, and whilst it is mainly Python code, there is some compiled 'C' for servo control, as well as HTML and .css files for use in a Flask web server browser-based user interface option.

Test routines

During the quadrabot build process a number of routines are available to test individual components of the assembly as follows:

- *I2C_servo_test.py*: routine to test and calibrate the movement positioning of each of the servos.
- *OLED_simple_text.py*: routine to check the display output from the OLED.
- *robot_control_all_inputs.py*: routine to check the inputs received from a PiHut wireless controller.
- *sensor-distance02.py*: routine to show the performance of the ultrasonic sensor.
- *servo_reset.py*: routine to set the servos into a standard 'calibration' position for setting the leg positions before 'locking' the servo horns onto their splined shafts.
- *switch_test.py*: routine to check how the slide switch combinations are used to indicate the selected control mode.
- *walking_tests.py*: routine to show the various quadrabot movement cycles 'in slow motion' so that they can be visually checked, and servo positionings adjusted as needed.

'Production' control software

The main Python control program, *quadrabot8DOF_action05.py*, and the separate Flask based web control code, *quadrabot8DOF_web02.py*, should be set up with a cron command to automatically start whenever the quadrabot is powered up.

The *crontab* lines would look something like the following:

```
# start main python control code for the 8DOF quadrabot
@reboot python3 /home/pi/quadrabot/quadrabot8DOF_action05.py >> /dev/null 2>> /dev/null

# start web interface code for the 8DOF quadrabot
@reboot sudo python3 /home/pi/quadrabot/quadrabot8DOF_web02.py >> /dev/null 2>> /dev/null
```

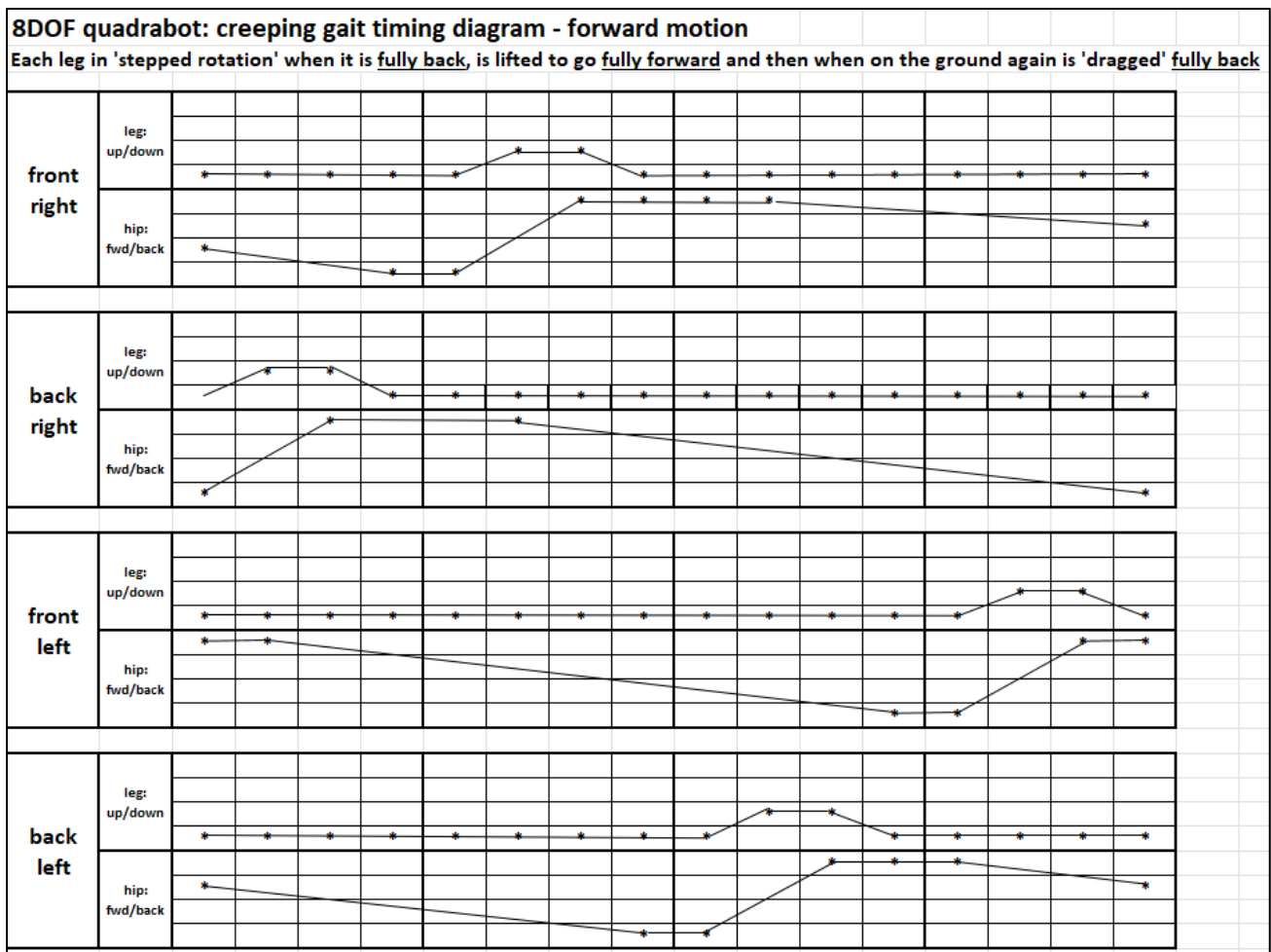
It should be noted that *sudo* is used for the Flask web server command so that the server can be accessed from the usual/default HTTP port 80.

Like any Flask application running on a Raspberry Pi, the server is accessible from whatever local network that the Pi is connected to and can be accessed from a browser using the Pi's hostname or IP address.

Gait analysis and implementation

The currently used 'creeping' gait should be considered as just an initial implementation that, whilst it 'works', is probably not that efficient and further work could almost certainly improve its speed of operation, and perhaps completely different 'gait' styles would also be possible which could be faster and much more efficient.

To develop this initial 'creeping gait' a spreadsheet has been used to capture what have been called *timing diagrams* which, as illustrated in an example screen shot from the spreadsheet below, tries to show the simultaneous leg and hip positioning for each of the four legs, as a percentage of their possible movement, over a cycle divided into 16 time slots. This cycle of movements is repeated continually to sustain the specific movement style, of which four different types have been developed, namely *forwards*, *backwards*, *turn left* and *turn right*.



The percentage movement are then translated into servo PWM signals, adjusted as necessary for the individual servo calibrations.

Quadrabot operational usage notes

For normal usage it is assumed that the two main Python control programs, *quadrabot8DOF_action05.py* and *quadrabot8DOF_web02.py*, have been set to automatically start in sequence when the Raspberry Pi is powered up.

Quadrabot start-up steps:

- Insert the wireless controller 'dongle' into the USB cable female connector held in the 3D printed holder and connect the cable's male micro connector to the Raspberry Pi's USB port.
- Insert all four rechargeable AA batteries into the battery holder, having previously checked that they are fully charged, and are making good contact with the battery holder terminals.
- Connect the battery bank, which should be in a well charged state, to the power in connector on the Raspberry Pi.
- Make sure the three slide switches are set to: demo – OFF – web off.
- Make a single short press of the USB battery bank button to 'switch on' the battery bank, which will result in:
 - a red LED lighting up on PCA9865 board;
 - the Raspberry Pi starting to boot;
 - the OLED displaying a series of start-up messages.

When the Raspberry Pi power up sequence has completed, the robot sets itself to the upright standing position with each leg set at 45° to the body and the OLED will display:

```
system idle/OFF
* legs reset *
```

Slide switch control options

The three slide switches on the underside of the body 'logically' control the start/stop of different operational modes as follows:

ON - OFF: logical start / stop of an individual mode (but does not switch the power!)

auto – demo: } the four possible combinations of these two
web on – off: } switches set different modes as shown below

Switch combination	Operational mode
auto – web off	autonomous mode: forward walking with automatic avoidance of obstacles
demo – web off	demonstration mode: simple demo routine that ends with a 'bow' followed by standing up
demo – web on	web mode: web browser interface, more below
auto – web on	wireless controller mode: further details below

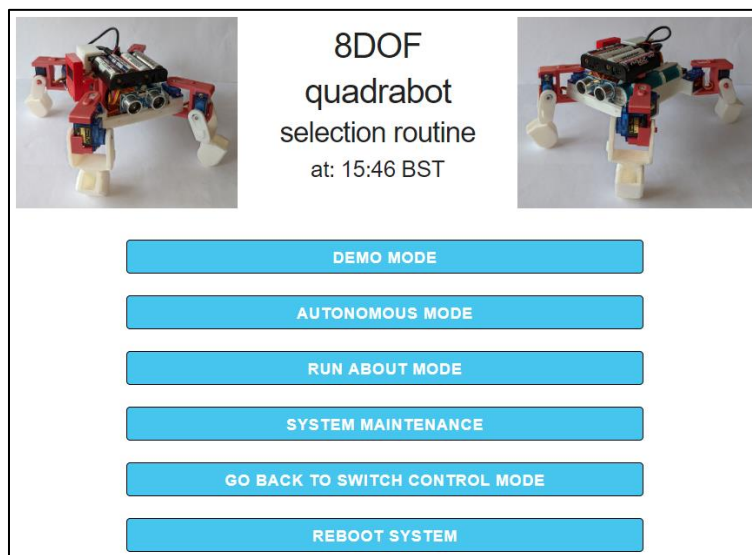
Wireless controller mode:

- With the Raspberry Pi fully powered up, switch the quadrabot into active wireless controller mode i.e., **auto – ON – web on**
- Switch on the 'matched' PiHut wireless controller:
 - Powered by 3xAAA rechargeable batteries.
 - Check that the controller 'matches' the USB dongle that is inserted into the USB cable holder
- Use the D-pad to carry out individual forward, right, back, left movement controls. Each D-pad button press is cached and does a small series of robot 'walking' steps that each take a short time to execute, so don't press an individual button too many times too quickly.
- Use the right hand buttons as follows:
 - Use the pink square button to wave the front left leg.
 - Use the green triangle button to wave the front right leg.
 - Use the blue cross button to wave the back left leg.
 - Use the red circle button to wave the back right leg.
- Use the left joystick button to do a 'squat'.
- Use the right joystick button to do a 'bow'.

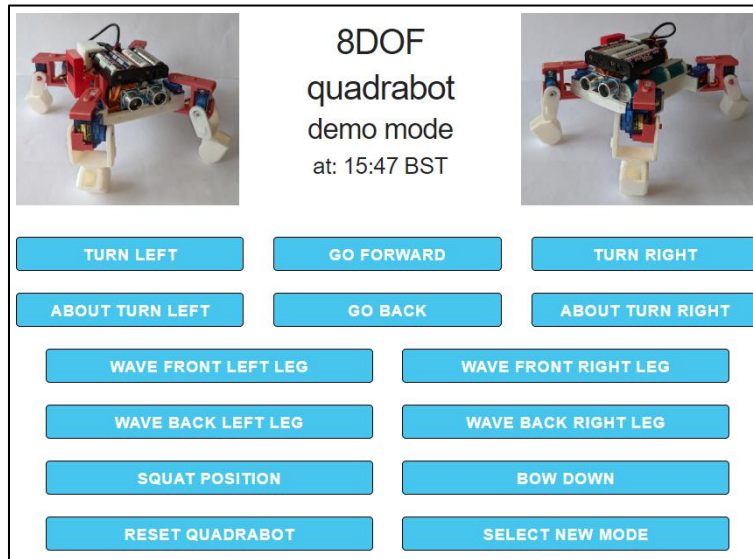
Web mode

The images below show all the various screens that can be accessed from the Flask based web interface.

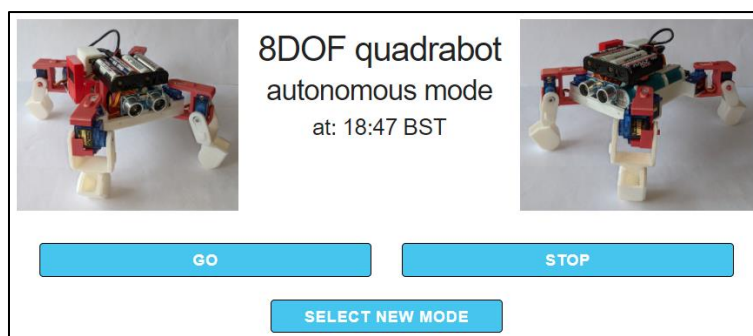
The 'home' / main selection page is shown below, from which all the various web usage options can be selected.



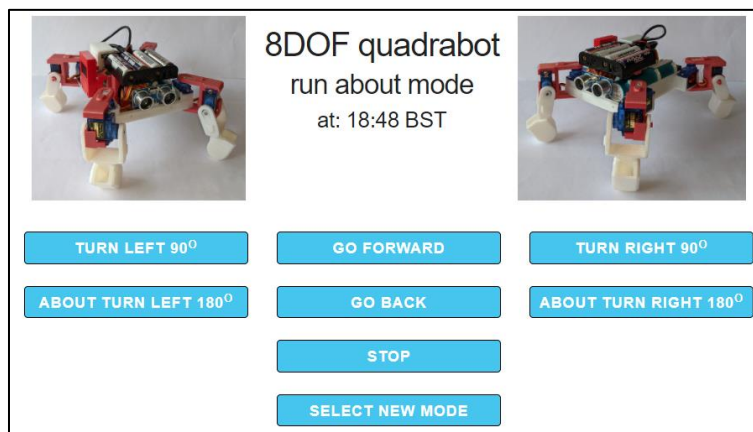
When the 'DEMO MODE' option is selected from the main screen the screen below allows individual quadrabot 'actions' to be demonstrated.



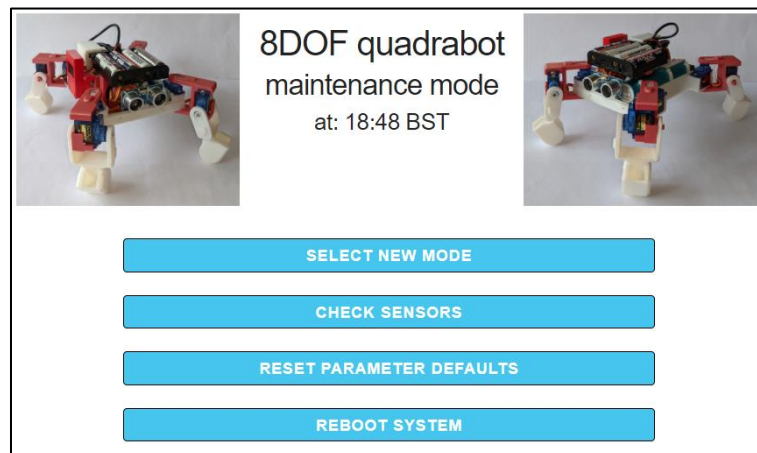
Selecting the 'AUTONOMOUS MODE' from the main screen, with GO/STOP buttons being displayed as shown below, allows the quadrabot to move around autonomously, automatically detecting obstacles with the ultrasonic sensor and attempting to avoid them by backing away and turning in a different direction.



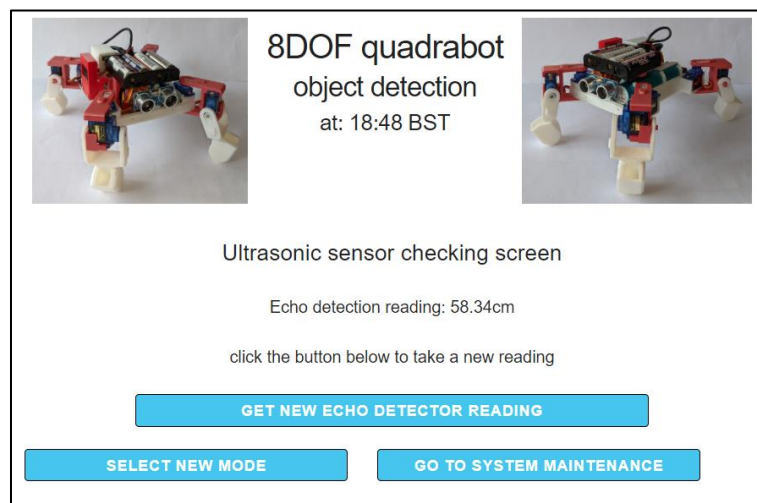
If the 'RUN ABOUT MODE' is selected from the main screen the screen below allows the quadrabot to 'run about' controlled by clicking the various control buttons shown.




Selecting the 'SYSTEM MAINTENANCE' option from the main screen shows another selection screen as shown below:



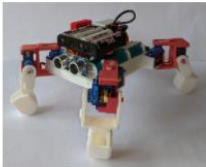
Using the 'CHECK SENSORS' maintenance option, the performance of the ultrasonic sensor can be demonstrated and can be used repeatedly to show the 'sensed' distance to an object by clicking the 'GET NEW ECHO DETECTOR READING' button.



The 'RESET PARAMETER DEFAULTS' maintenance option allows various default system parameters to be updated and re-stored in the file held on the system's SD card. It should be noted that ordinarily the GPIO pin numbers would never be changed!



8DOF quadrabot
reset defaults
at: 18:49 BST



Below are the current values for a number of system parameters - these should not normally be changed!

SELECT NEW MODE

SYSTEM MAINTENANCE

GPIO pinTrigger default value:

UPDATE

GPIO pinEcho default value:

UPDATE

GPIO onoff default value:

UPDATE

GPIO auto_demo default value:

UPDATE

GPIO web default value:

UPDATE

HowNear default value:

UPDATE

ReverseCycles default value:

UPDATE

TurnCycles default value:

UPDATE

Alternate default value:

UPDATE

The 'REBOOT SYSTEM' option, available from both the maintenance and the main selection screens, will display the screen shown below and then reboot the Raspberry Pi's operating system.



8DOF quadrabot
rebooting system
at: 15:49 BST

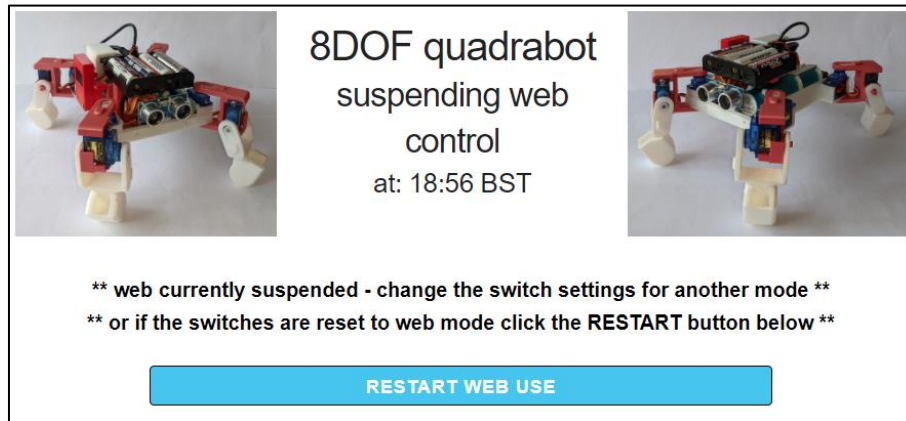


**** system rebooting - reset the ON/OFF slide switch to OFF ****
**** once the reboot is complete - use the slide switches to set a new mode ****
**** and if web mode is set just click the RESTART button below ****

RESTART WEB USE

Finally, the 'GO BACK TO SWITCH CONTROL MODE' from the main screen will display the screen shown below and suspend the use of the Flask web server, allowing the slide switches on the quadrabot to be changed so that another non-web usage mode can be used.

If, using the slide switches, the web option is reselected at some stage, then the browser interface can be restarted by clicking the 'RESTART WEB USE' button which will redisplay the 'home' / main selection screen.

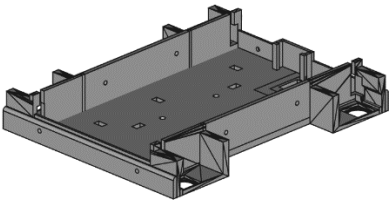
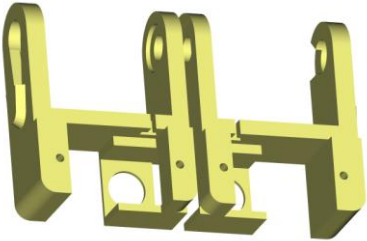
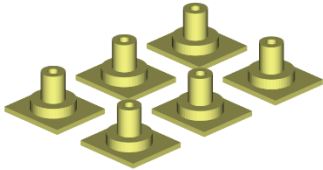
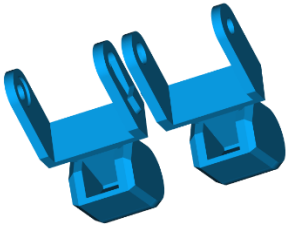
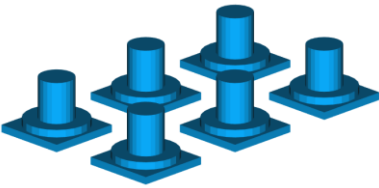
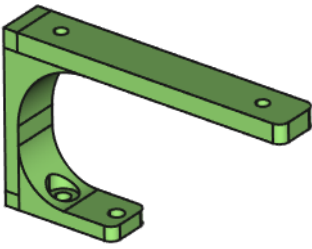


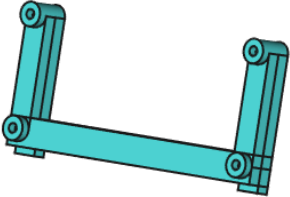
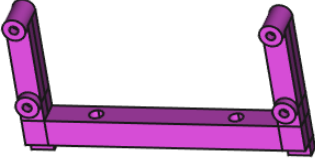
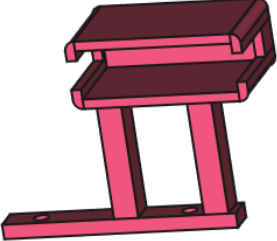
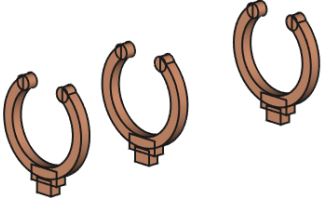
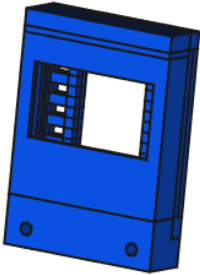
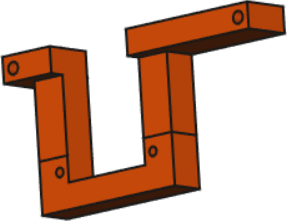
Quadrabot shut-down steps:

- Press the USB battery bank button for several seconds until the lights go out on both the PCS9865 and the Raspberry Pi.
- Remove at least one of the AA batteries to prevent them all from discharging unnecessarily.

Appendix A: 3D printed components

All the 3D print components listed below use 1.75mm PLA with a layer height of 0.15mm and the design files can be downloaded from the Prusa web site [here](#), where further print details are provided for each component when produced on a Prusa i3 Mk3 printer.

	<p>Download file: <i>m-body_assembly20.stl</i> - main body 'pan' to which the legs are fitted on all four corners and the various control components are installed.</p>
	<p>Download file: <i>m-hip_x2_repaired03_mirrored.stl</i> - pair of 'hip' frames: 2 pairs to be printed.</p>
	<p>Download file: <i>m-hip-shaftsx6.stl</i> - 6 shafts for the non-drive ends of the servos fitted into the hip frames (only 4 of the 6 needed).</p>
	<p>Download file: <i>m-leg_x2_angled02_mirrored.stl</i> - pair of 'legs': 2 pairs to be printed - support should not be needed but could obviously be added but a large brim is used as a precaution.</p>
	<p>Download file: <i>m-leg-shaftsx6.stl</i> - 6 shafts for the non-drive ends of the servos fitted into the 'pockets' on the main body (only 4 of the 6 needed).</p>
	<p>Download file: <i>4AA_battery_stand10.stl</i> - mounting bracket for a 4AA battery holder: uses 2x M3 8mm long pan head screws to self-tap into the main body 'pan' base and 2x M2 6mm long CSK head screws to secure the battery holder to the bracket - although using just one CSK screw allows the battery holder to swivel to allow the wireless controller 'dongle' to be easily inserted into the USB cable.</p>

	<p>Download file: <i>PiZ_v_holder03.stl</i> - mounting bracket for Raspberry Pi Zero: secured with 2x M3 8mm long pan head screws that self-tap into the main body 'pan' base and 4x self-tap 6mm long M2 flanged pan head screws used to fix the Raspberry Pi to the mount.</p>
	<p>Download file: <i>PWM_module_vertical_holder03.stl</i> - mounting bracket for PCA9685 module: secured with 2x M3 12mm long pan head screws that go through both the wireless controller 'dongle' holder and this mounting bracket and self-tap into the main body 'pan' base. 4x self-tap 6mm long M2 flanged pan head screws used to fix the PCA9685 module to the mount.</p>
	<p>Download file: <i>quadrabot_dongle_holder05_rotx180.stl</i> - holder for the female socket of a USB cable, into which a wireless controller 'dongle' can be inserted. As discussed above this fits on top of the bottom arm of the PCA9685 mounting bracket.</p>
	<p>Download file: <i>3x_battery_180deg_clamp01.stl</i> - three clamps for holding a 22mm diameter rechargeable battery bank that powers the Raspberry Pi: they are just push fits into slots on the base of the main body 'pan'.</p>
	<p>Download file: <i>OLED1_flat08-front+back.stl</i> - two piece holder that encases a small 64x128 pixel OLED. The two pieces are secured together using 4x self-tap 6mm long M2 flanged pan head screws and the overall assembly is secured to the mounting bar (described below) using 2x 16mm long M2 pan head screws.</p>
	<p>Download file: <i>OLED_mounting_bar01_roty90.stl</i> - bolt-on mounting bar for the OLED assembly described above and secured to the back of the PCA9685 mounting bracket with 2x 16mm long M2 pan head screws.</p>

Appendix B: Servo movement values

The servos managed by the Raspberry Pi Zero are controlled by a board called the PCA9685 which allows up to 16 servos to be provided with what is called a Pulse Width Modulation (PWM) signal from a single electronic interface to/from the Raspberry Pi known as I²C.

The I²C protocol was originally designed by Philips in the early 1980s to allow easy communication between components mounted on the same circuit board, and the name stood for "Inter IC" i.e., Inter Integrated Circuit, sometimes shown as IIC but more commonly as I²C.

I²C is now, not only used on single boards, but also to connect components which are linked via cable, and it is its simplicity and flexibility that make this protocol attractive in many applications.

Pulse Width Modulation (PWM), as illustrated in the diagram above right, simply means a digital signal that is on/off (pulsed) for a period of time, where the **frequency** at which the on/off switching occurs can be set, and where the amount of time spent on, versus off, known as the **duty cycle** and expressed either as a time or as a percentage, can also be set.

The small, low-cost SG90 servos used for the quadrabot leg movements can move a servo arm through approximately 180° (90° in either direction), by sending them a PWM signal at a frequency of 50Hz (i.e., one cycle takes 20 milliseconds) and where the 90° and the -90° servo arm positions are set by the PWM duty cycle being set roughly between 1 and 2 milliseconds (ms).

However, the PCA9685 board that manages the servos, and sends these PWM control signals, does not take a duty cycle time as its input, instead it divides a complete cycle into 4096 steps and the 'servo movement settings' used in the code, typically in the range 150 to 560, are the number of steps i.e., 1ms is 1/20th of a cycle or about 205 steps and 2ms will equate to 410 steps. The 150 to 560 typical range mentioned above then allows for individual servos to have a wide tolerance, which is typical for low-cost servos like the SG90.

50% duty cycle



75% duty cycle



25% duty cycle



Appendix C: control/storage file details

The quadrabot files on a Raspberry Pi Zero are assumed, as a default, be stored in the following sub-folder structures: */home/pi/quadrabot/*

Files in the main sub-folder:

<i>quadrabot8DOF_action05.py</i>	main Python control program
<i>quadrabot8DOF_web02.py</i>	Flask based web interface program
<i>libquadrabot_servo.so</i>	custom compiled 'C' library for fast servo actions
<i>quadrabot_servo.c</i>	source code for the <i>libquadrabot_servo.so</i> custom library
<i>quadrabot_servo.h</i>	source code for the <i>libquadrabot_servo.so</i> custom library
<i>inputs.py</i>	inputs library from https://pypi.org/project/inputs (https://github.com/zeth/inputs)
<i>I2C_servo_test.py</i>	component test routine for servos
<i>OLED_simple_text.py</i>	component test routine for OLED
<i>robot_control_all_inputs.py</i>	component test routine for PiHut wireless controller inputs
<i>sensor-distance02.py</i>	component test routine for ultrasonic sensor
<i>servo_reset.py</i>	component test routine calibrating the servo horn positioning
<i>switch_test.py</i>	component test routine for slide switches
<i>walking_tests.py</i>	component test routine for quadrabot movement checking
<i>/templates</i>	folder of the various Flask web control HTML templates
<i>/static</i>	folder of Flask .css file, images, etc., used in web control

Flask web control files in the */templates* folder:

<i>autonomous_mode.html</i>	HTML template for the autonomous mode web page
<i>check_sensors.html</i>	HTML template for the web page that checks the ultrasonic sensor
<i>defaults.html</i>	HTML template for the web page that can update the various system default parameters
<i>demo_mode.html</i>	HTML template for the demonstration mode web page
<i>maintain_mode.html</i>	HTML template for the top level maintenance web page
<i>quadrabot_header_insert.html</i>	HTML template for the master header insert for all the web pages
<i>quadrabot_layout.html</i>	HTML template for the master layout for all the web pages
<i>run_about.html</i>	HTML template for the web page for controlling the quadrabot in 'run about' mode
<i>select_mode.html</i>	HTML template for the master/home page that allows all the main web page options to be selected
<i>system_reboot.html</i>	HTML template for the web page that is displayed when a system reboot is carried out
<i>web_suspended.html</i>	HTML template for the web page that is displayed when the web mode is suspended and quadrabot operation reverts to other slide switch controlled options

Flask web control files in the */static* folder:

<i>/css/normalize_advanced.css</i>	.css file for web page formatting
<i>/css/skeleton_advanced.css</i>	.css file for web page formatting
<i>/images/8DOF_quadrabot_20220329_103800232_250h.jpg</i>	image used in web page header
<i>/images/ 8DOF_quadrabot_20220329_103837756_250h.jpg</i>	image used in web page header

Custom 'C' code:

The custom compiled 'C' code *libquadrabot_servo.so*, enabling fast instructions to be sent to the PWM control board, is underpinned by an installed libPCA9685 library (see <https://github.com/edlins/libPCA9685> for details).

The libPCA9685 library, once installed, will be at:

/usr/local/include/PCA9685.h and */usr/local/lib/libPCA9685.so*

This custom code *libquadrabot_servo.so* was compiled on a Raspberry Pi Zero 2 - so may need to be recompiled for a different SBC.

The compilation command used is as follows:

```
gcc -shared -o /home/pi/quadrabot/libquadrabot_servo.so -fPIC  
/home/pi/quadrabot/quadrabot_servo.c -I/usr/local/include -L/usr/local/lib -lPCA9685
```

Text file handling:

The 'main' and the 'web' Python code exchange text 'messages' by writing/reading to/from text files that are stored in a ramdrive and are set up at:

- */mnt/interproc01/proc_message01.txt*
- */mnt/interproc01/proc_message02.txt*
- */mnt/interproc01/proc_message03.txt*
- */mnt/interproc01/proc_message04.txt*

A simpler 'virtual log' in memory, using io.StringIO, is used in the web Python code as a temporary 'log' for the last issued command.

A set of system default parameters are stored in a text file on the SD card system at:

/home/pi/logfiles/quadrabot_logs/quadrabot_default.txt

END OF DOCUMENT