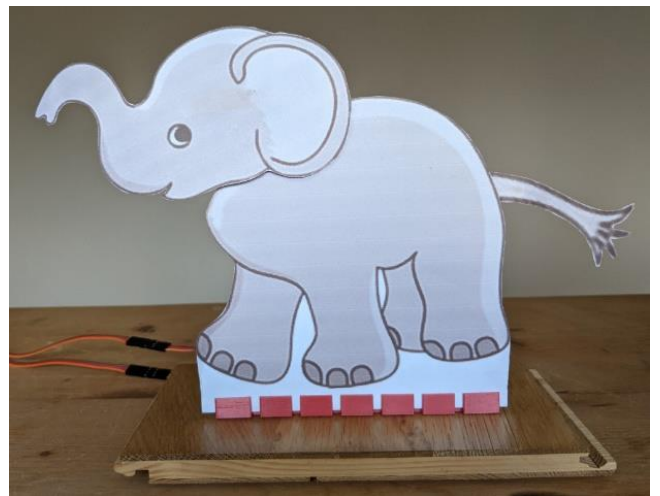
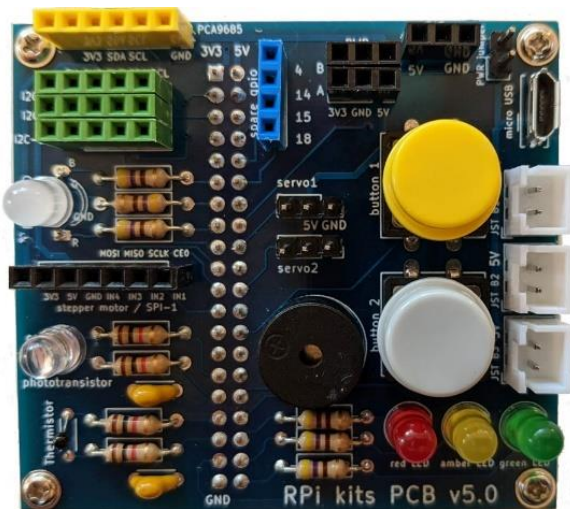


Raspberry Pi Maker Kit

Mechatronics build & usage



version 1.0

Table of contents

Introduction.....	3
Mechatronic builds	4
Servo motor mechatronic builds.....	4
Stepper motor mechatronic builds.....	5
Drive motor mechatronic builds	6
Maker Kit PCB connections	6
Servo motor PCB connections.....	6
Stepper motor PCB connections	7
Drive motor PCB connections.....	7
16 character x 2 line LCD.....	8
Control software.....	9
Web interface.....	9
Appendix A: 3D printed connector components.....	16
Appendix B: control/storage file details	20
Folder structure	20
Action control file details.....	21
Appendix C: Servo movement values.....	24

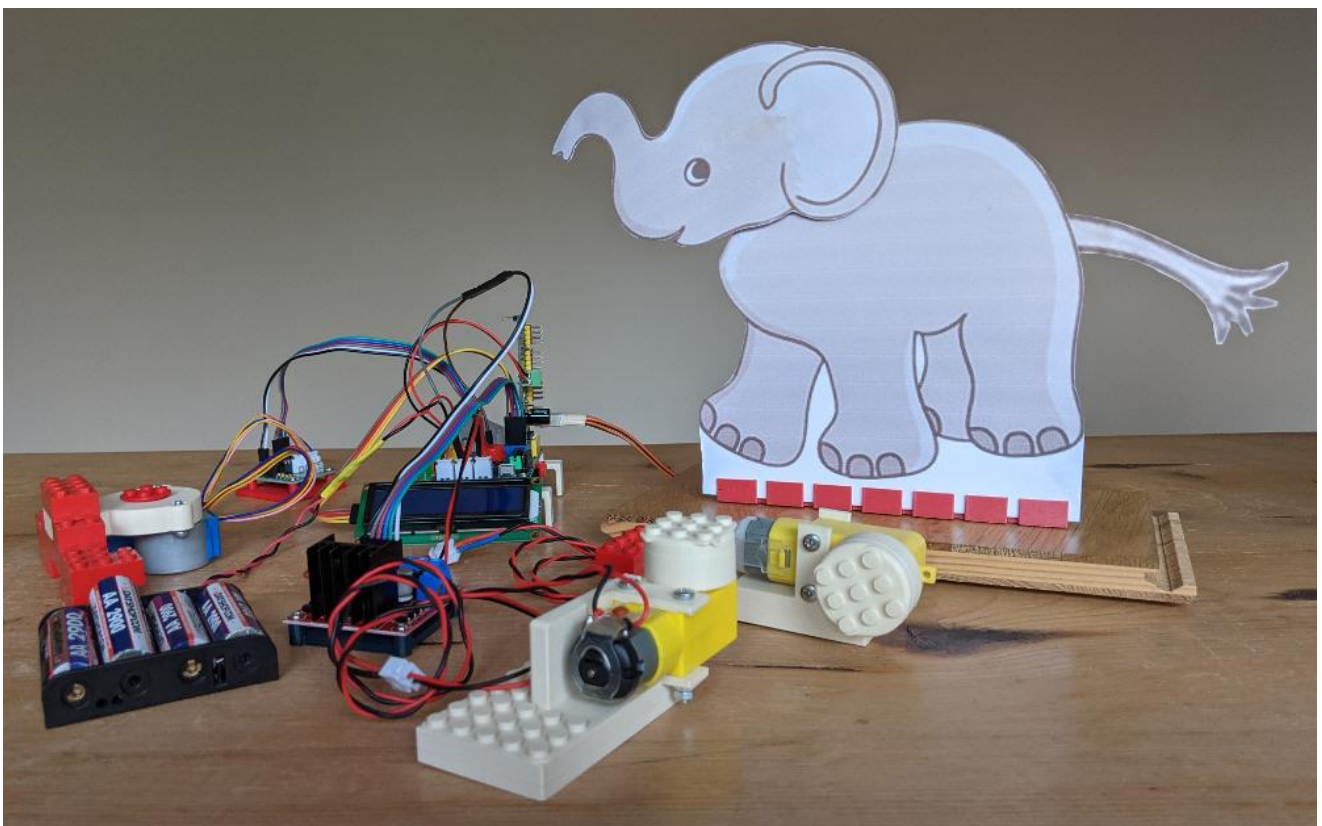
Introduction

The word 'mechatronic' is used to mean any object where the movement, and sometimes sound generation, is managed by a computer such as a Raspberry Pi.

The object could be anything from a 2D animal outline (sometimes called an 'animatronic') through to a complex 3D Lego structure.

A Raspberry Pi + Maker Kit 'mechatronic' build uses 3D printed servo connector components and Lego adaptors to assemble a 2 or 3-dimensional 'structure' of servos or a structure with a rotational element driven by a stepper or drive motor, that may also then be 'clad' with a 'skin' to provide a reasonable representation of the intended 'moving object'.

This document provides build information for a simple 2D mechatronic to illustrate build processes and then details are provided on how to use a Raspberry Pi Maker Kit to control a built assembly using different components as illustrated in the image below.



Full detail on the 3D print designs for the various connector/adaptor components can be found at the Prusa web site [here](#), [here](#) and [here](#) plus all the software and related materials for use with a Raspberry Pi can be downloaded from [here](#).

Mechatronic builds

Version 1.0 of the software constrains individual builds to using only one type of motor i.e., a servo, stepper, or drive motor. To support a wide range of possible builds a number of 3D printed connector components have been developed and are listed in **Appendix A: 3D printed connector components**.

Each of the three mechatronic build types are now described.

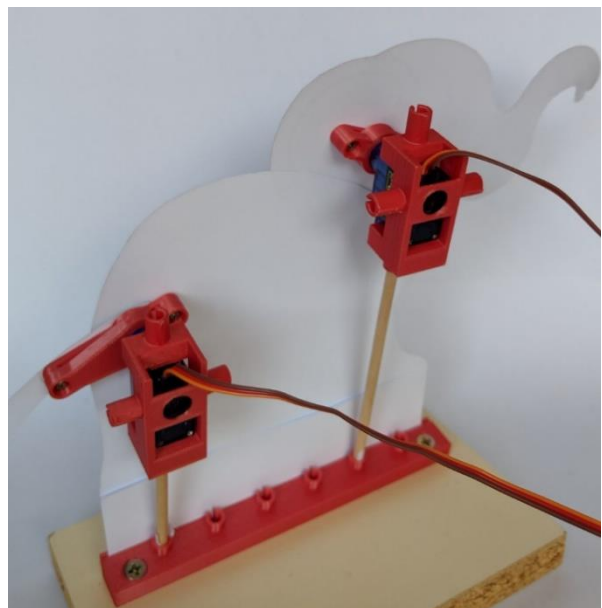
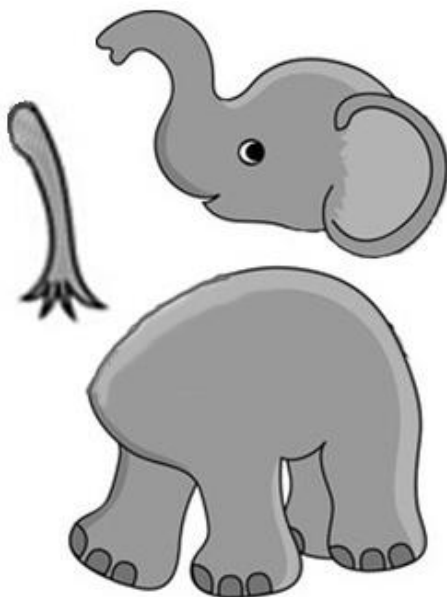
Servo motor mechatronic builds

To illustrate a typical servo 'framework' build a simple 2D elephant mechatronic can be built from cardboard cut-outs of the 'separated' body, head, and tail of an elephant, as shown in the image below left, and two servos are used to move the head and tail.

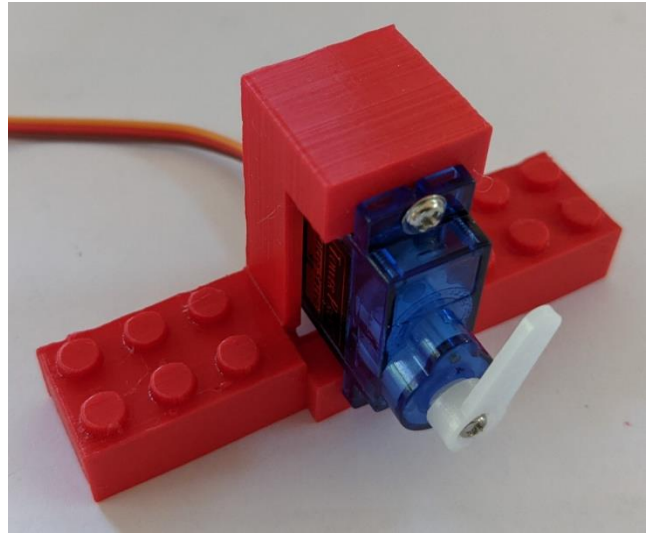
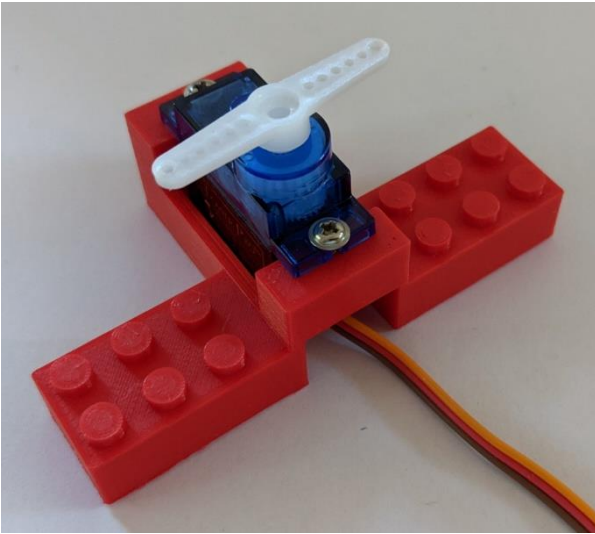
The servos are 'mounted' using various 3D printed servo connector components, as shown in the 'rear view' image shown below.

The head and tail cut-outs are fixed to connector components that 'hold' the servo arms, and the servo motors are fitted into 3D printed 'cases' that are held at the required heights and spacing using wooden dowels inserted into a 3D printed mounting rail that is screwed to a wooden base.

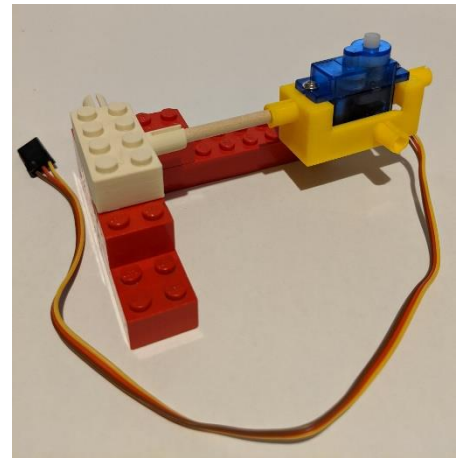
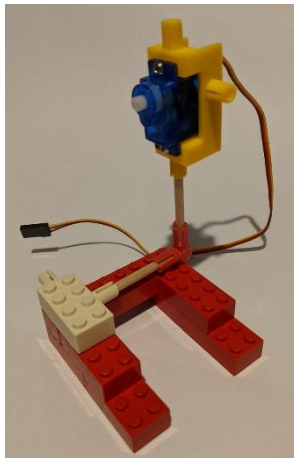
As part of the build set up and alignment, the servo arms need to be attached to the splined servo drive shaft so that the servo movement range is as required for a 'nodding' head and a 'wagging' tail. The servos are then connected to a PCA9865 PWM control module inserted into the PCB of the Raspberry Pi Maker Kit. The current software assumes PWM channels 0 and 1 are used respectively for the head and the tail, but this can obviously be changed.



The 'dowel' interconnect approach described above can be used to construct a wide variety of different frameworks that position servos in a structured arrangement, but LEGO compatible connectors, illustrated below, are also available.



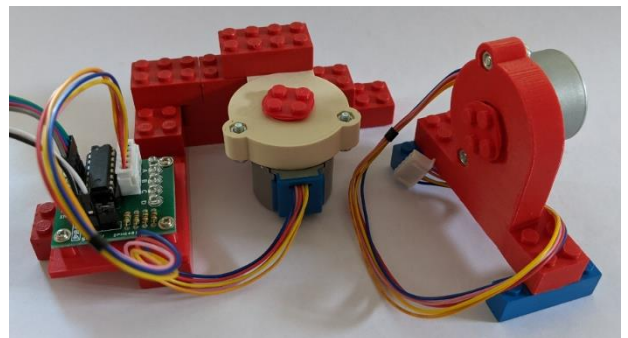
A LEGO-dowel connector is also available, as shown below left, and the further two images below illustrate how dowel and LEGO constructs can be inter-mixed.



Stepper motor mechatronic builds

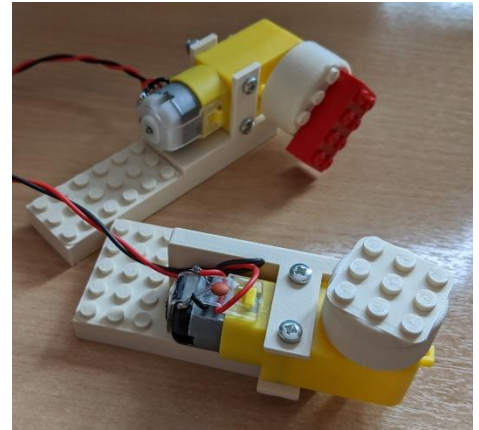
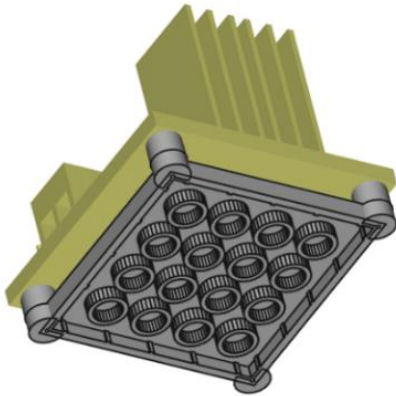
The image on the right shows a pair of LEGO compatible adaptors that can be used to mount a 28BYJ-48 stepper motor in either a horizontal or vertical position with an additional connector that fits the drive shaft of the motor. A connector is also available for the ULN2003 control board.

The available number of GPIO connections on a Raspberry Pi Maker Kit's PCB limits the number of stepper motors that can be used to just one, but with these connectors it can be incorporated into an extensive overall LEGO build.



Drive motor mechatronic builds

The image on the right shows a pair of LEGO compatible adaptors that can be used to mount a geared drive motor (YG2900) in either a horizontal or vertical position with an additional connector that fits the drive shaft of the motor.



A LEGO compatible connector is also available for a L298N PWM control board as illustrated on the left.

The available number of GPIO connections on a Raspberry Pi Maker Kit's PCB limits the number of drive motors that can be used to two, but with these connectors they can be incorporated into an extensive overall LEGO build.

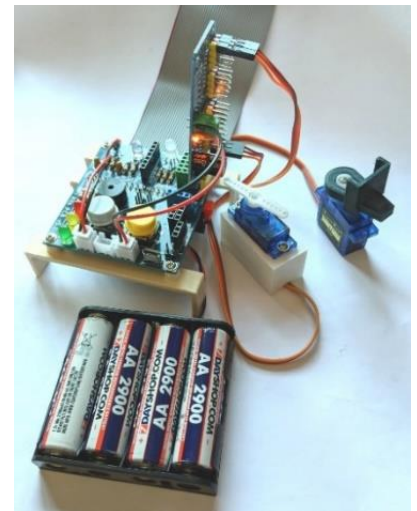
Maker Kit PCB connections

Servo motor PCB connections

Up to 16 servo motors can be connected to a PCA9685 control board that is simply inserted into its dedicated female header on the Maker Kit PCB as illustrated in the image on the right.

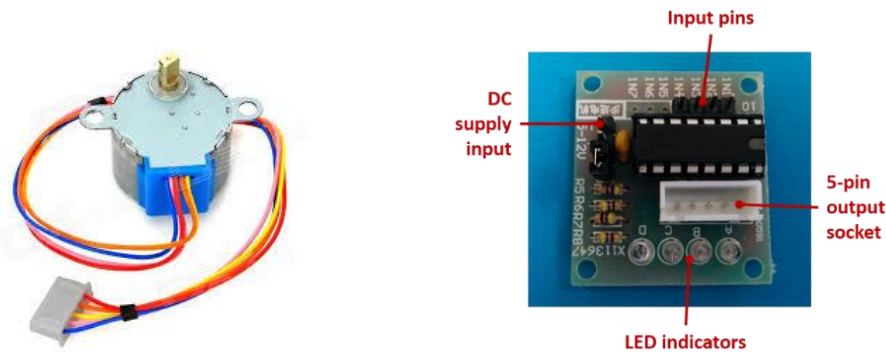
The 3 wires from each servo are usually coloured coded brown, red and yellow (GND, V+, signal) where these colours are matched on the 16 sets of 3 pin male connectors on the PCA9685 board, where the servo channel numbers, 0-15, are usually also labelled on the board. Depending upon the size and 'spread' of the mechatronic build the servo wires may need to be extended in order to reach the control board.

Also shown in this image is a 4xAA battery pack that is connected to the JST female connector of the power bus of the PCB, with a second connection from the power bus to the external power supply of the PCA9685 board which then provides power to the connected servos.

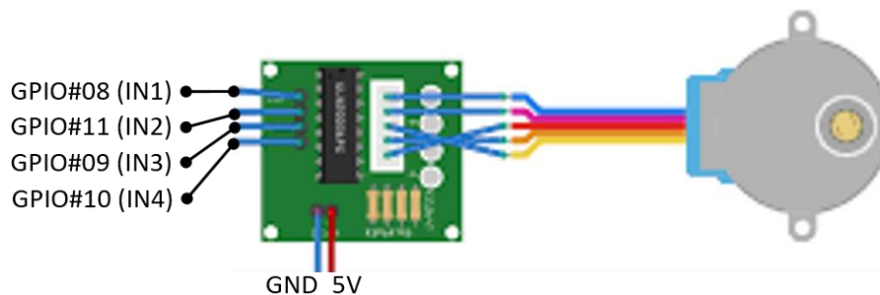


Stepper motor PCB connections

The low cost 28BYJ-48 stepper motor with its associated ULN2005 control board is shown below.



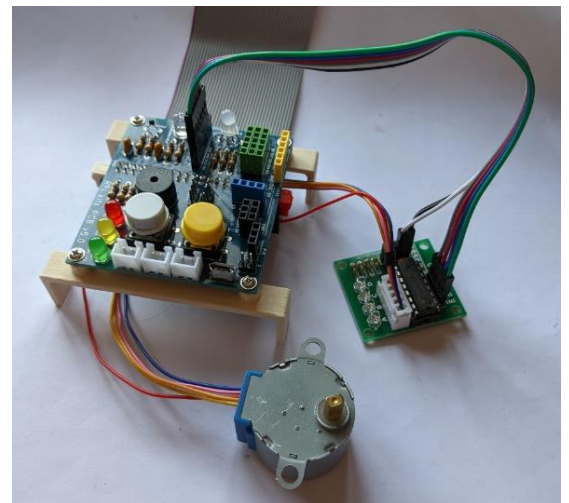
The schematic/circuit diagram below shows how the ULN2005 control board uses the IN1-IN4 connections that are on the 7-pin black female connector on the PCB.



The stepper motor can usually be powered direct from the Raspberry Pi, assuming it is supplied by an adequate 5V DC power supply with the recommended 2.5A capacity for the Raspberry Pi 3, or 3A capacity for the Raspberry Pi 4.

For this direct powered arrangement, the image on the right shows how the GND-5V connections for the ULN2005 control board can also be fed from the 7-pin black female connector on the PCB.

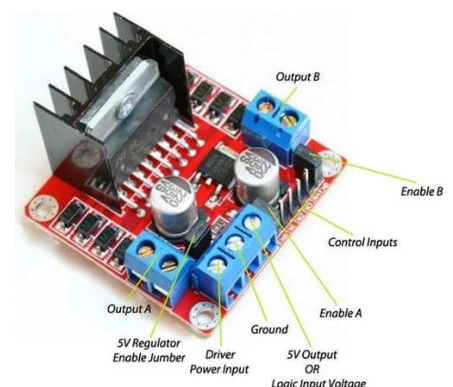
However, if there is any doubt about this, or problems are experienced, then the ULN2005 can be separately powered from the PCB's power bus which also has 4xAA batteries connected.



Drive motor PCB connections

Mechatronic builds using drive motors use the L298N motor controller board, as shown on the right, which provides a wide range of capabilities for controlling two electric motors.

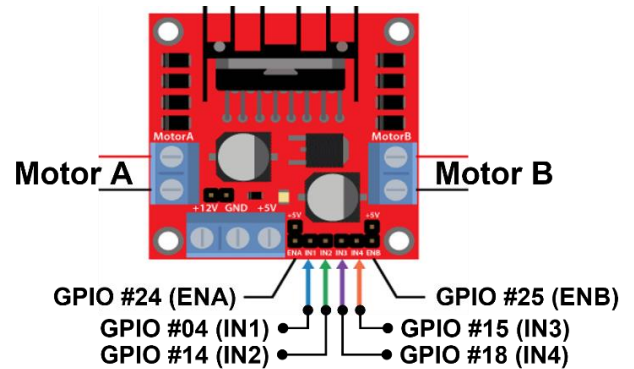
The motors can be powered from a source that can supply from 5 to 35 volts DC and each motor can be supplied with 2A continuously.



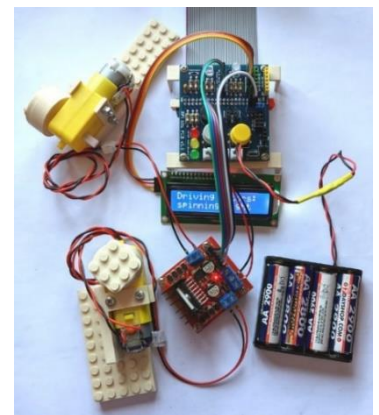
The board itself is powered from the same supply source with the voltage automatically stepped down to 5 volts DC and when powered correctly the L298N board has a red LED switched on.

The schematic/circuit diagram on the right shows how the mechatronic L298N motor control method uses a set of GPIO pins that do not conflict with stepper motor usage so that both types of motor can be connected at the same time.

The IN1-4 connections use the 'spare' GPIO pin female header connections on the PCB (GPIOs 4, 14, 15 and 18) and the ENA & ENB connections use the two male header connections normally used for direct management of servos (GPIOs 24 & 25)

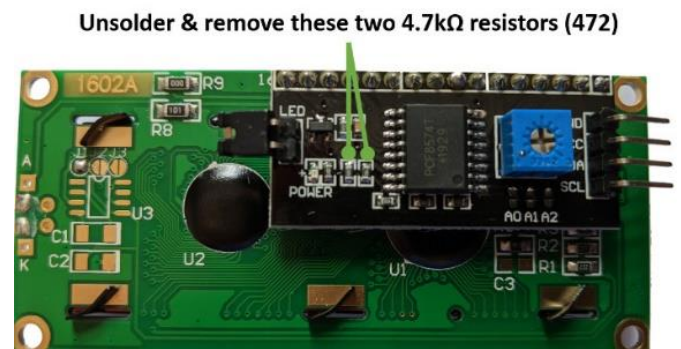


As shown in the image on the right, a L298N motor controller can be used to independently control two drive motors when connected to the Maker Kit PCB, with power for the motor controller and the two drive motors supplied by a 4xAA battery pack connected the PCB's power bus, with a second connection from the power bus to the L298N's power input terminals.



16 character x 2 line LCD

For all the different mechatronic builds the software currently assumes an I²C controlled 16x2 LCD is installed, as shown in the image above right. These readily available 5V powered displays can be connected to one of the 5-pin green female headers on the Maker Kit PCB. But for a 5V powered device there will normally be 4.7k Ω pull-up resistors between both the SDA and SCL control lines as shown in the image on the right, and these must be removed to avoid exceeding the permitted voltage on these connections to a Raspberry Pi.



Control software

The software used to control an individual mechatronic build, available to download from [this GitHub repository](#), uses Python with some compiled 'C' and has a browser-based user interface controlled by a Flask web server. Like any Flask application running on a Raspberry Pi, it 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. Assuming the Python code is in the default named folder structure it can be run with the following command:

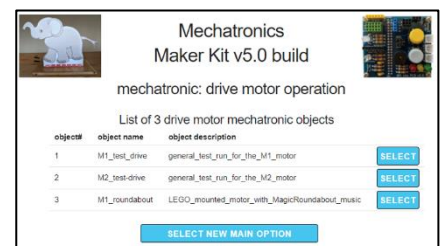
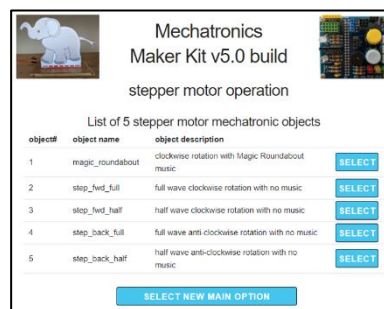
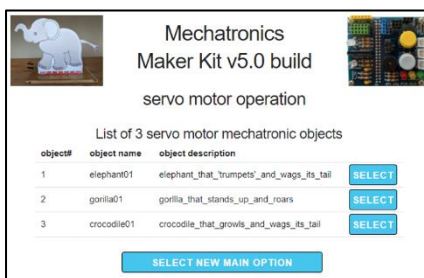
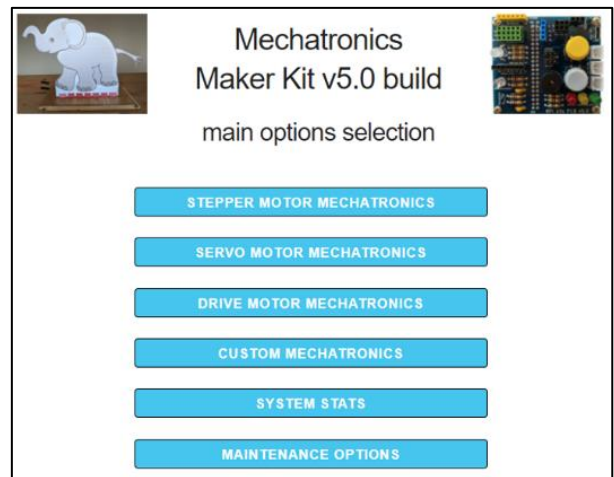
```
sudo python3 /home/pi/RPi_maker_kit5/mechatronics/mechatronics01.py
```

- where sudo is used so that the Flask web server can be accessed from the usual/default HTTP port 80.

Web interface

The image on the right is a screen shot of the browser view of the main selection web page which provides links to allow the 3 types of mechatronic builds to be controlled, along with a 4th 'custom' build option described in more detail later, plus options to display various system statistics and to action various maintenance activities.

Clicking one of the three main mechatronic build types then displays a screen, as shown below for each type, which lists the builds that have been 'loaded' into the system and allows an individual build to be selected.



The builds that are currently in the system are stored in text files as follows:

Stepper motor builds: *stepper_options4x.txt*

Servo motor builds: *servo_objects4x.txt*

Drive motor builds: *mdrive_objects4x.txt*

- where each of these files are stored in the:

/home/pi/RPi_maker_kit5/mechatronics/action_sequences01 folder on the Raspberry Pi.

Each of these build 'listing' files has a similar structure as shown in the example *servo_objects4x.txt* file below:

object_name	reference	sequence_file	object_description
elephant01	1	elephant_sequence04.txt	elephant_that_'trumpets'_and_wags_its_tail
gorilla01	2	gorilla_sequence04.txt	gorilla_that_stands_up_and_roars
crocodile01	3	crocodile_sequence04.txt	crocodile_that_growls_and_wags_its_tail

In the Python code this data is 'read' into an array using the NumPy .loadtxt function where the first line is ignored – it is there just to provide a visual description for the data values. The four values on each row only need a single space to separate them but multiple spaces are used to simply align the data with the headings in the first row.

For servo motor builds an additional data file, *servo_servos4x.txt*, is used to provide additional detail about each of the builds in the system. As with other control files, the first line is just descriptive headers. Subsequent lines then consist of 34 parameter all on the same line.

Shown below is the current default text file content, split into two sections just because the lines are too long to fit into this document's width.

The first set of 22 parameters:

servo_object	num	c	desc	c	desc	c	desc	c	desc	c	desc	c	desc	c	desc	c	desc	c	desc	c	desc
elephant01	2	0	head	1	tail	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
gorilla01	1	15	body	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
crocodile01	2	0	jaw	1	tail	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

The next set of 23 to 34 parameters:

c	desc	c	desc	c	desc	c	desc	c	desc	c	desc
-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-

Details for each of the 34 parameters are as follows, where the example values shown in bold text are from the first executable line of the text file shown above:

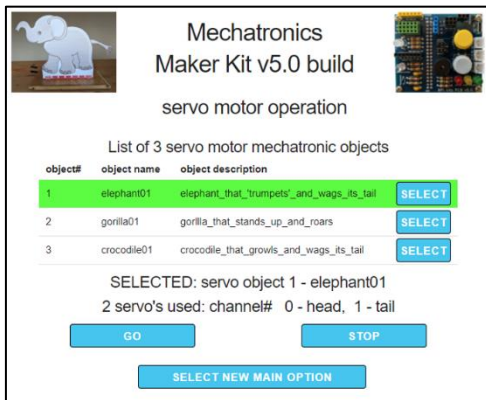
servo_object: descriptive text for the object, e.g., **elephant01**

num: the number of servo's used to construct the object, e.g., **2**

3-4 through to 33-34 **c desc:** pairs of channel number and short contiguous text, usage descriptive text, e.g., **0 head** and **1 tail**

As can be seen in the above example, even though the servo object only uses 2 servo's, because the system is capable of operating 16 servo's for a single object, all the unused servo entries must still be entered into the line of text in the file using - - characters as dummy text.

Further details on the structure of all the various control/storage files are provided in **Appendix B: control/storage file details**.



When an individual object is 'selected' in one of the previously shown 'listing' screens, the screen refreshes showing the selected object highlighted in green with additional detail about the build extracted from the various data files discussed above, along with **GO** and **STOP** buttons being shown that can be clicked to operate the mechatronic build.

The screen shot on the left shows, as a representative example, the result for a selected servo motor build, but the other two types of motor

build will display very similar updated screens.

When an object is set to GO, individual action movement sequences plus sounds to be played that are 'programmed' by a simple control text file, are executed. These *action control files*, all stored in the:

`/home/pi/RPi_maker_kit5/mechatronics/action_sequences01` folder, can be edited directly on the Raspberry Pi, i.e., outside of the application, but one of the **MAINTENANCE OPTIONS** described later allows them to be edited via the web interface.

A typical *action control file* (elephant_sequence04.txt) is shown below for a servo motor build:

servo	ref	move1	sound1_file_name	time_ms	move2	sound2_file_name	time_ms	repeat
0	5	150	elephant8.mp3	3500	300	nothing	500	1
1	6	300	nothing	500	150	nothing	500	2
0	5	150	elephant8.mp3	2500	300	nothing	500	1
1	6	300	nothing	500	150	nothing	500	2
0	5	150	elephant8.mp3	2500	300	nothing	500	1
1	6	300	nothing	500	150	nothing	500	2

The first line of the object control file is ignored by the Python code, as it is there simply to remind the user of the content structure, but each subsequent line of the text file is 'executed' in sequence.

Each of these subsequent lines are sets of nine parameters separated by at least one space – several spaces can also be used, as in the example above, just so that the text elements are aligned in columns that match the content headings in the first line.

Details for each of the nine parameters are as follows, where the example values shown in bold text are from the first executable line of the text file shown above:

1. **servo**: channel# connection to the PWM board e.g., **0** for the elephant head
2. **ref**: a unique servo Id reference (added for potential future 'calibration' use) e.g., **5**
3. **move1**: the first servo movement setting (typically in the range 150 to 560 – see **Appendix C: Servo movement values** for further details) e.g., **150**
4. **sound1_file_name**: the file name for a sound to be played during this first servo 'movement' e.g., **elephant8.mp3** (if nothing is to be played the text must be set to exactly '**nothing**'). It should be noted that only the file name need be added here since the 'path' to the file i.e., **/home/pi/RPi_maker_kit5/mechatronics/sounds/** is defined in the code
5. **time_ms**: the delay in milliseconds after the first servo movement is initiated (this is to allow the servo movement and/or the sound to complete) e.g., **2500**

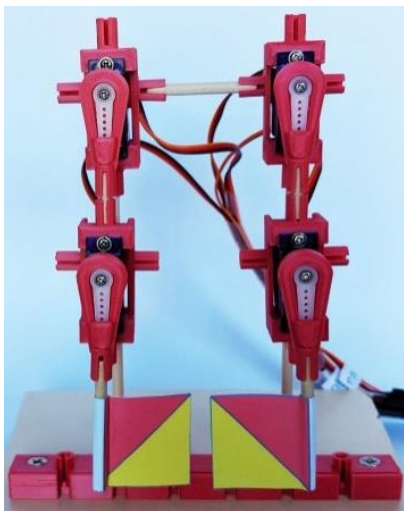
6. **move2:** the second servo movement setting (typically in the range 100 to 600) e.g., **300**
7. **sound2_file_name:** the file name for a sound to be played during this second 'movement' e.g., **nothing**
8. **time_ms:** the delay in milliseconds to wait after the second servo movement is initiated (to allow the movement and/or the sound to complete) e.g., **500**
9. **repeat:** the number of times that the first + second servo movements in the individual execution line and their associated sounds are to be repeated e.g., **1**


It should be noted however that when the overall movement cycle is initiated through the web interface, all the execution lines are repeatedly executed in sequence until the operation is stopped. So, this overall repeat cycle coupled with the individual 'line repeats' provides a great deal of flexibility in defining a sequence of movements.

The three types of mechatronic builds discussed above (using servo, stepper, and drive motors) all use a similar operational method, as described above, for which there is a common set of management code for each motor type.

A 4th option available from the web interface however is for **CUSTOM MECHATRONICS** which provides access to dedicated code for a particular custom build.

When this 'custom' option is selected from the main menu a listing of the current custom builds is displayed as shown on the right. But it should be noted that whilst two items are shown, only the 'semaphore' build, shown below left, has so far been coded.






Mechatronics Maker Kit v5.0 build

custom mechatronic object list

List of 2 custom mechatronic servo objects

object#	object name	object description	
1	semaphore01	four servo: shoulders and elbows semaphore flags	SELECT
2	robot_arm01	four servo XYZ and gripper robot arm	SELECT

[SELECT NEW MAIN OPTION](#)



Selecting the 'semaphore' build will display the screen shown on the right which allows a specific short text message to be entered and then the build to operate – which will move the arms of the 4-servo build to 'flag' the text message.


The last two main menu items are **SYSTEM STATS**, which displays the screen shown below right and the **MAINTENANCE OPTIONS** which displays a series of sub menus as shown below left:

Individual screen shots for the third, second and first **MAINTENANCE OPTIONS** are discussed below (the 4th option is not yet coded).


The **SOUND FILE MANAGEMENT** 3rd option shown on the right:

- lists the current .mp3/.wav files that are currently in the system
- allows each sound to be played, and
- provides an upload mechanism to add new sounds

file#	file name	
1	lion4.mp3	SELECT
2	gorilla1.wav	SELECT
3	camera_click.mp3	SELECT
4	ape1.mp3	SELECT
5	monkey1.mp3	SELECT
6	rooster01.mp3	SELECT
7	crocodile5.mp3	SELECT
8	elephant8.mp3	SELECT
9	MagicRoundabout.mp3	SELECT
10	gorilla2.mp3	SELECT



Mechatronics Maker Kit v5.0 build



mechatronic: sound output selection

Sound output is set to the local analog port, but if an HDMI screen with sound e.g. a TV, is connected to the Raspberry Pi / Maker Kit running the mechatronic then use the button below to select HDMI output

Sound output currently set to: *local analog port*


SET SOUND OUTPUT TO HDMI CABLE

BACK TO MAINTAIN OPTIONS


SELECT NEW MAIN OPTION

The **SOUND OUTPUT MANAGEMENT** 2nd option shown left sets the sounds to be output to either an HDMI display (which obviously should have sound output) and to which the Raspberry Pi is connected, or to a powered loudspeaker that is connected to the Pi's analog sound port.

The **OBJECT FILE MANAGEMENT** 1st option provides a browser-based set of facilities for editing and adding new **action control files** for non-custom motor types and when selected from the **MAINTENANCE OPTIONS** menu the screen shown on the right is displayed.



Mechatronics Maker Kit v5.0 build



mechatronic: object type selection


STEPPER MOTOR OBJECTS

SERVO MOTOR OBJECTS


DRIVE MOTOR OBJECTS

BACK TO MAINTAIN OPTIONS

SELECT NEW MAIN OPTION



Mechatronics Maker Kit v5.0 build



mechatronic: servo object management

List of 3 servo motor mechatronic objects

object#	object name	object description	
1	elephant01	elephant that 'trumpets' and wags its tail	EDIT
2	gorilla01	gorilla that stands up and rears	EDIT
3	crocodile01	crocodile that growls and wags its tail	EDIT

add a new servo object:

new object name: sequence file:

new object description:

[submit new servo object](#)

clone an existing servo object:

object# to clone: new object name:

new sequence file name:


new object description:

[submit clone details](#)

BACK TO OBJECT TYPE CHOOSE


SELECT NEW MAIN OPTION

An individual motor build type can then be selected and, as an example, the screen shot on the left shows the screen for the servo motor type which allows a new action control file to be added for that motor type or an existing one to be edited.



Mechatronics Maker Kit v5.0 build

mechatronic: servo object management



List of 3 servo motor mechatronic objects

object#	object name	object description	
1	elephant01	elephant that 'trumpets' and wags its tail	EDIT
2	gorilla01	gorilla that stands up and roars	EDIT
3	crocodile01	crocodile that growls and wags its tail	EDIT

EDITING: servo object 1 - elephant01
2 servo's used: channel# 0 - head, 1 - tail

2 servo's defined below in 'channel# : short description' data pairs

0	head	1	tail	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

[update the servo channel data](#)

6 actions defined for this object

servo	ref	move1	sound file1 name	pause1	move2	sound file2 name	pause2	repeat			
0	5	150	elephant8.mp3	3500	300	nothing	500	1	edit	del	test
1	6	300	nothing	500	150	nothing	500	2	edit	del	test
0	5	150	elephant8.mp3	2500	300	nothing	500	1	edit	del	test
1	6	300	nothing	500	150	nothing	500	2	edit	del	test
0	5	150	elephant8.mp3	2500	300	nothing	500	1	edit	del	test
1	6	300	nothing	500	150	nothing	500	2	edit	del	test

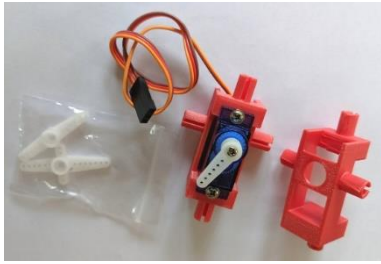




new action#: [insert new action](#)

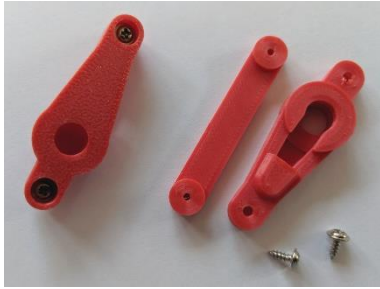
[BACK TO OBJECT TYPE CHOOSE](#)
[SELECT NEW MAIN OPTION](#)

Selecting an individual object to be edited displays the screen on the left which, because it is for a servo motor object, not only allows 'actions' to be added or edited, but also allows the object's entry in the *servo_servos4x.txt* to be updated.

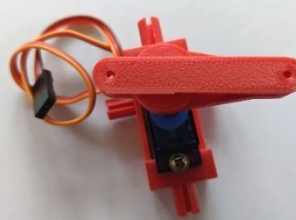
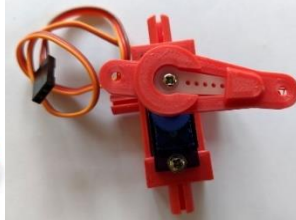
Appendix A: 3D printed connector components

To enable the design of whatever creative 'moving object' a user wants to build, a set of 3D printed components and Lego adaptors, listed below, have been developed that allow servo, stepper or drive motors to be assembled into 2D or 3D structures using 3.8mm (5/32-inch) wooden dowel and standard Lego bricks.

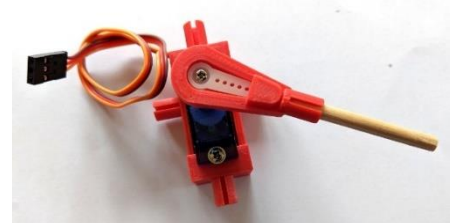
	<p>This servo case allows a small, lightweight SG90 servo to be provided with dowel fixing points on four sides.</p> <p>The SG90 servo is held in place in the case by the 2 self-tap screws that are supplied with the rotor 'horns' of the SG90 or by other 2mm self-tap screws. The single sided rotor 'horn', used in various ways as described below, is attached to the SG90 with the very small screw supplied with the SG90 'horns'. Take care when trying to screw this into the splined hub of the servo since it is very easy to drop it onto the floor where it will get lost and there is only one supplied with each servo horn!</p>
	<p>The 6-point base bar/card slot allows several upright dowels that support a set of interconnected servos to be attached to a base behind a card image. The 6 dowel fixing points are at a small angle to the vertical in order to compensate for dowel bending due to the offset weight of the attached servos.</p> <p>The base bar has 2 fixing holes so that it could be screwed to a larger wooden block to provide a suitable weighted and stable overall base, or sticky foam pads could be used to adhere the bar to a base instead. If screws are used, as the base bar fixing holes are also at an angle, the fixing screws will need to be screwed into the wood at an angle.</p>
	<p>The longer card slot bar matches the length of the 6-point base bar/card slot above for arrangements where the card image needs to be set further away from the servo dowel fixings</p>
	<p>The 3-point base bar/card slot provides the same function as the 6-point base bar/card slot but can be used in multiples to achieve different upright dowel spacings or offset positioning if the overall servo structure needs this.</p>
	<p>The shorter card slot bar matches the length of the 3-point base bar/card slot above for arrangements where the card image needs to be set further away from the servo dowel fixings</p>



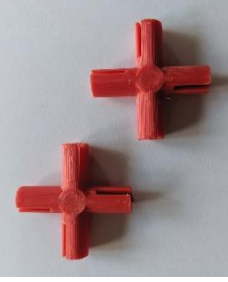



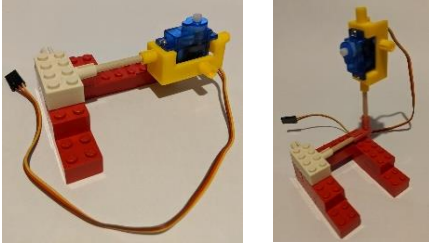
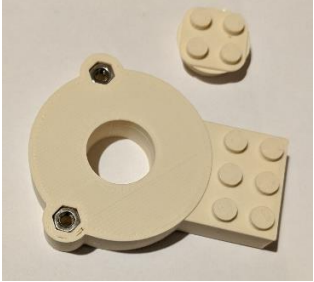

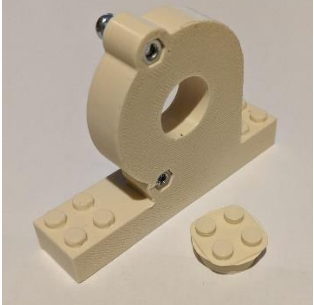
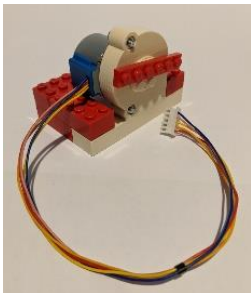
The **2-piece servo arm** allows large mechatronic cardboard cut-outs to be attached to a SG90 servo using the single arm rotor 'horn' supplied with the SG90. As shown in the series of images below: first the horn is pushed firmly into the main servo arm piece; the combined servo arm and horn are then pushed onto the splined drive of the servo (the exact rotational position will need to be adjusted for the specific mechatronic design) and the small screw supplied with the servo horns used to secure the horn to the servo; then the connecting piece of the servo arm can be glued to the cardboard cut-out and the cut-out + connecting piece screwed to the main servo arm piece.

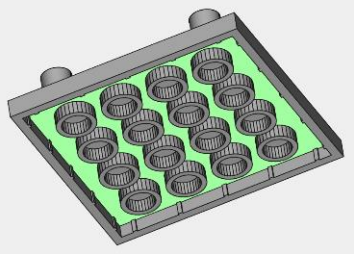
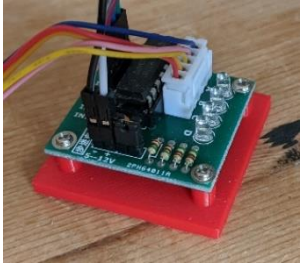
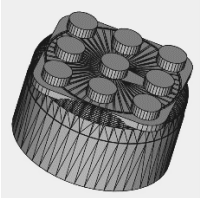
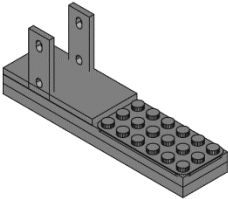
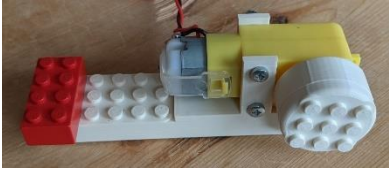
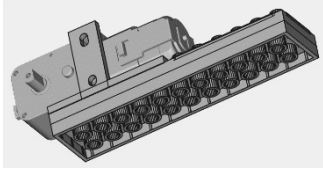
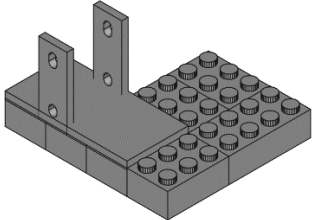
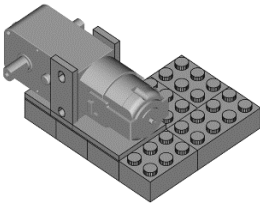
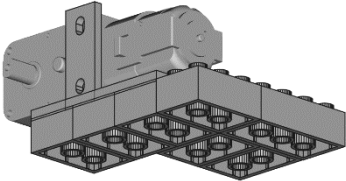
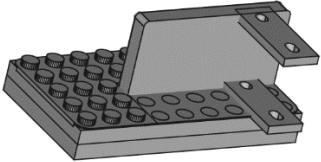
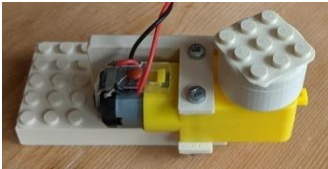
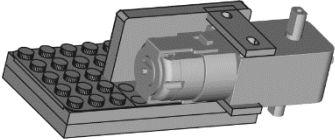
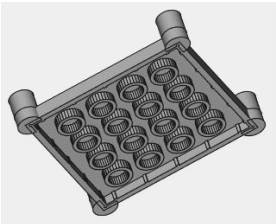
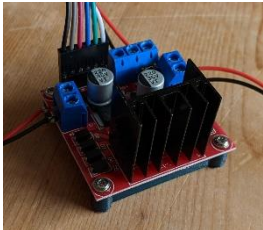


The **servo arm + shaft** allows smaller/lighter items to be attached to a servo such as a thin animal tail or the flags used in a semaphore messaging servo structure. As shown in the series of images below: first the horn is pushed firmly into the servo arm; then the servo arm can be pushed onto the splined drive of the servo (the exact rotational position will need to be adjusted for the specific mechatronic design) and the small screw supplied with the servo horns used to secure the horn to the servo – a small washer can be used as a spacer to make sure the arm does not tilt once fitted; then an appropriate length of 3.8mm dowel can be inserted into the arm to which a cardboard cut-out or flag etc., can be fixed.



The **right-angled connector** allows two dowels to be connected at right angles.

	<p>The cross connector allows up to four dowels to be interconnected at right angles in the same plane.</p>
	<p>The T-connector allows up to three dowels to be interconnected at right angles in the same plane.</p>
	<p>The corner connector allows 3 dowels to be connected in the x, y, z directions.</p>
	<p>The 4x2 Lego brick right-angled connector allows one or two dowels to be connected to a standard Lego brick, which in turn can be part of an overall Lego built structure, which when used in conjunction with other connectors allows servo motors to be mounted in various ways as shown above right.</p> <div data-bbox="1026 887 1457 1133">  </div>
	<p>The 3x2 Lego brick horizontal stepper motor housing and drive adaptor allows the standard stepper motor to be added to an overall Lego brick structure as shown on the right.</p> <div data-bbox="1206 1290 1457 1626">  </div>
	<p>The 3x2 Lego brick vertical stepper motor housing and drive adaptor also allows the standard stepper motor to be added to an overall Lego brick structure as shown on the right.</p> <div data-bbox="1206 1648 1457 1939">  </div>

	<p>The 5x5 Lego tile adaptor, as shown on the right, for a ULN2005 stepper motor control board, allows the control board to be integrated in an overall Lego brick structure.</p> 
	<p>Drive motor shaft LEGO adaptor allows LEGO based elements to be rotated by a drive motor as shown in further images below.</p>
	<p>This horizontal drive motor LEGO adaptor mounts a YG2900 geared drive motor and allows it to be integrated in an overall Lego brick structure as shown in the images below.</p>  
	<p>This alternative horizontal drive motor LEGO adaptor mounts a YG2900 geared drive motor and allows it to be integrated in an overall Lego brick structure as shown in the images below.</p>  
	<p>This vertical drive motor LEGO adaptor mounts a YG2900 geared drive motor and allows it to be integrated in an overall Lego brick structure as shown in the images below.</p>  
	<p>L298N 5x5 LEGO tile adaptor, as shown in the image on the right, mounts a L298N motor controller allowing it to be integrated in an overall Lego brick structure.</p> 

Appendix B: control/storage file details

Folder structure

The mechatronic files on a Raspberry Pi SBC are, as a default, stored in the following sub-folder structures:

/home/pi/RPi_maker_kit5/mechatronics/ - main 'holding folder' for all the files

/home/pi/RPi_maker_kit5/mechatronics/ - folder for all the code as follows:

mechatronics01.py - main Python control program

I2C_LCD_driver.py - LCD display driver library

libpicontrol_servo.so - custom compiled 'C' library for fast servo actions

/templates - folder of the various Flask web control HTML templates

/static - folder of Flask .css file, images, etc., used in web control

Additional compiled 'C' code enabling fast instructions to be sent to the PWM control board will have been installed when the Raspberry Pi SBC is first set up and would be at:

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

/home/pi/RPi_maker_kit5/mechatronics/action_sequences01/

- folder for servo, stepper, and drive control files:

stepper_options4x.txt - list of the stepper motor objects

servo_objects4x.txt - list of all the servo motor objects

servo_servos4x.txt - a sub list of the servo channel# used by each object

mdrive_objects4x.txt - list of all the drive motor moving objects

custom_servo_objects4x.txt - list of the current custom servo moving objects

crocodile_sequence04.txt - example crocodile servo sequence file

elephant_sequence04.txt - example elephant servo sequence file

gorilla_sequence04.txt - example gorilla servo sequence file

magic_roundabout04.txt - example stepper run file (Magic Roundabout music)

stepper_back_full04.txt - stepper full speed backwards run file (no sound)

stepper_back_half04.txt - stepper half speed backwards run file (no sound)

stepper_fwd_full04.txt - stepper full speed forwards run file (no sound)

stepper_fwd_half04.txt - stepper half speed forwards run file (no sound)

M1_roundabout.txt - example drive motor sequence file (Magic Roundabout music)

M1_test_drive04.txt - example drive motor sequence file (no sound)

M2_test_drive04.txt - example drive motor sequence file (no sound)

semaphore_message.txt - semaphore message to be 'flagged'

semaphore_letter_gap.txt - delay in seconds between 'flagged' letters

letters.csv - data file of 'arm' angle positions for each letter of the alphabet

semaphore_servo_data.csv - semaphore servo calibration data for angle positions

`/home/pi/RPi_maker_kit5/mechatronics/sounds/` - folder for the sound mp3/wav files as follows:

```
ape1.mp3
camera_click.mp3
crocodile5.mp3
elephant8.mp3
gorilla1.wav
gorilla2.mp3
lion4.mp3
MagicRoundabout.mp3
monkey1.mp3
rooster01.mp3
```

`/home/pi/RPi_maker_kit5/mechatronics/cut-out_sheets/` - folder for items as follows:

```
crocodile_cut_out_sheet01.pdf
elephant_cut_out_sheet01.pdf
gorilla_cut_out_sheet01.pdf
semaphore_flag_positions_A-Z.pdf
semaphore_flag_cut_out_sheet01.pdf
```

Action control file details

Stepper motor action control files

Below is an example stepper motor build *action control file* (`magic_roundabout.txt`).

```
step  direction  speed  sound_file_name  time_ms  play
1      clock      full   MagicRoundabout.mp3  all      repeat
```

The first line of all these control files is ignored, as it is there simply to remind the user of the content structure, and the file contents then consists of a single line of text containing a set of six parameters separated by one or more spaces. For the *magic_roundabout.txt* shown above, each parameter is as follows with the example values shown in bold text:

- 1. step:** just a reference for the stepper motor, which is not used at present, but at some point, may be used if a future software version has more than one stepper motor connection, e.g., **1**
- 2. direction:** the rotation direction for the stepper motor viewed from the top of the drive shaft, which if set to *clock* makes the rotation direction clockwise (forwards), but if set to any other text will make the direction anti-clockwise (backwards), e.g., **clock**
- 3. speed:** the rotational speed of the stepper motor is somewhat limited, so the only options that can be set here are '*full*' or '*half*', meaning the fastest speed possible (1 rotation in about 7 seconds) and half that speed value, e.g., **full**

4. **sound_file_name:** the file name for a sound to be played during the stepper motor rotation, e.g., **MagicRoundabout.mp3** (if nothing is to be played the text must be set to exactly '**nothing**'). It should be noted that only the file name need be added here since the 'path' to the file i.e., `/home/pi/RPi_maker_kit5/mechatronics/sounds/` is defined in the code.
5. **time_ms:** this parameter is not currently used but will in a subsequent software release be used to stop the sound playing after a set number of milliseconds (ms), or if set to 'all' will play the full recorded duration, e.g., **all**
6. **play:** this is set to either '**once**' or '**repeat**' to either play the sound once for its recorded duration or to restart the sound and repeat playing it on a continuous basis whilst the stepper motor is running, e.g., **repeat**

Drive motor action control files

A typical *action control file* (M1_roundabout.txt) is shown below for a drive motor build:

motor	direction	%speed	run_duration(sec)	sound_file_name	play_time_ms	play
M1	forwards	90	15	MagicRoundabout.mp3	all	once
M1	stop	n/a	5	nothing	all	once
M1	backwards	70	10	MagicRoundabout.mp3	all	once
M1	stop	n/a	5	nothing	all	once

The first line of the object control file is ignored by the Python code, as it is there simply to remind the user of the content structure, but each subsequent line of the text file is 'executed' in sequence.

Each of these subsequent lines are sets of seven parameters separated by at least one space – several spaces can also be used, as in the example above, just so that the text elements are aligned in columns that match the content headings in the first line.

Details for each of the seven parameters are as follows, where the example values shown in bold text are from the first executable line of the text file shown above:

1. **motor:** motor# use either of the possible two drive motors e.g., **M1**
2. **direction:** motor rotation forwards/backwards/stop e.g., **forwards**
3. **%speed:** rotational speed as a percentage e.g., **90**
4. **run_duration(sec):** time spent rotating in seconds e.g., **15** but can be set to *continuous*
5. **sound_file_name:** the file name for a sound to be played during the rotational movement e.g., **MagicRoundabout.mp3** (if nothing is to be played the text must be set to exactly '**nothing**'). It should be noted that only the file name need be added here since `/home/pi/RPi_maker_kit5/mechatronics/sounds/` - the 'path' to the file is defined in the code
6. **play_time_ms:** not currently used for a designated play time in milliseconds, but will be executed as if the parameter is set to *all* i.e., to play the full sound file: e.g., **all**
7. **play:** either *once* or *repeat* to set how often the sound file should be played e.g., **once**

Servo motor action control files

A typical *action control file* (elephant_sequence04.txt) is shown below for a servo motor build:

servo	ref	move1	sound1_file_name	time_ms	move2	sound2_file_name	time_ms	repeat
0	5	150	elephant8.mp3	3500	300	nothing	500	1
1	6	300	nothing	500	150	nothing	500	2
0	5	150	elephant8.mp3	2500	300	nothing	500	1
1	6	300	nothing	500	150	nothing	500	2
0	5	150	elephant8.mp3	2500	300	nothing	500	1
1	6	300	nothing	500	150	nothing	500	2

The first line of the object control file is ignored by the Python code, as it is there simply to remind the user of the content structure, but each subsequent line of the text file is 'executed' in sequence.

Each of these subsequent lines are sets of nine parameters separated by at least one space – several spaces can also be used, as in the example above, just so that the text elements are aligned in columns that match the content headings in the first line.

Details for each of the nine parameters are as follows, where the example values shown in bold text are from the first executable line of the text file shown above:

1. **servo**: channel# connection to the PWM board e.g., **0** for the elephant head
2. **ref**: a unique servo Id reference (added for potential future 'calibration' use) e.g., **5**
3. **move1**: the first servo movement setting (typically in the range 150 to 560 – see [Appendix C: Servo movement values](#) for further details) e.g., **150**
4. **sound1_file_name**: the file name for a sound to be played during this first servo 'movement' e.g., **elephant8.mp3** (if nothing is to be played the text must be set to exactly '**nothing**'). It should be noted that only the file name need be added here since `/home/pi/RPi_maker_kit5/mechatronics/sounds/` - the 'path' to the file is defined in the code
5. **time_ms**: the delay in milliseconds to wait after the first servo movement is initiated (this is to allow the servo movement and/or the sound to complete) e.g., **2500**
6. **move2**: the second servo movement setting (typically in the range 100 to 600) e.g., **300**
7. **sound2_file_name**: the file name for a sound to be played during this second 'movement' e.g., **nothing**
8. **time_ms**: the delay in milliseconds to wait after the second servo movement is initiated (to allow the movement and/or the sound to complete) e.g., **500**
9. **repeat**: the number of times that the first + second servo movements in the individual execution line and their associated sounds are to be repeated e.g., **1**

Appendix C: Servo movement values

The servos connected to the Raspberry Pi Maker Kit are controlled by a control 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 to 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 all the mechatronic servo objects 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' for use in the control file, described above as being 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



END OF DOCUMENT