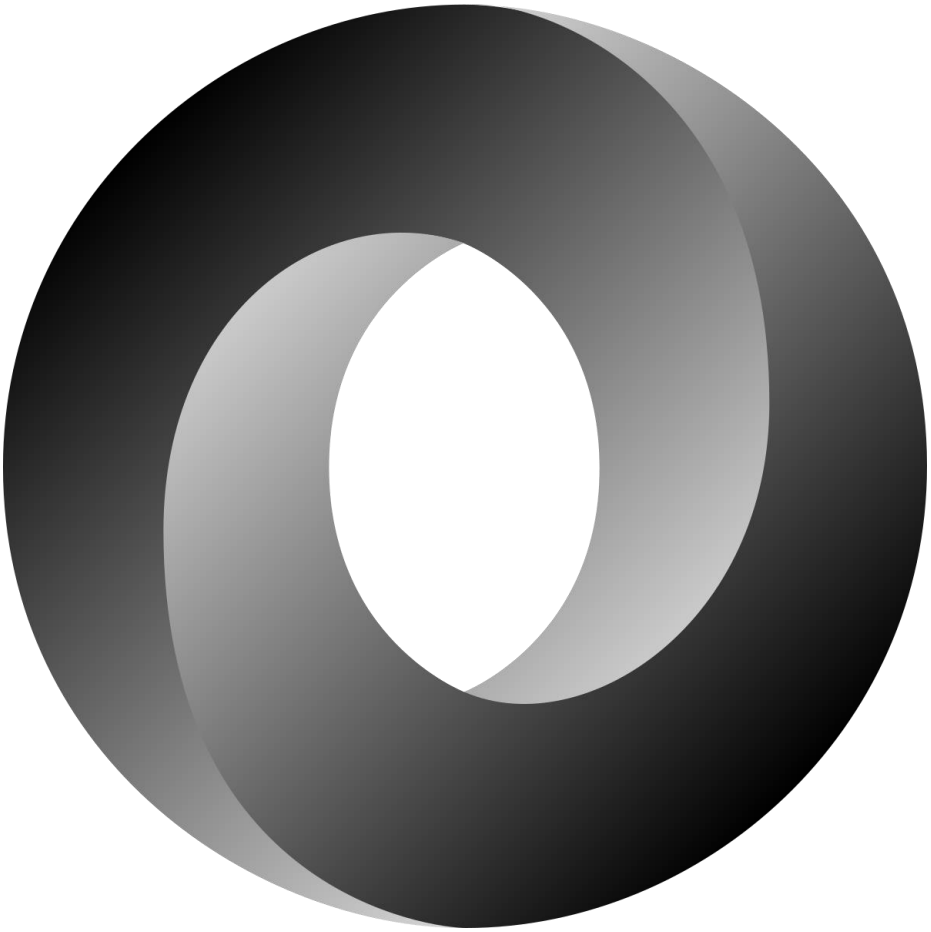
Json Editor

–Documentation –



BIOROB – EPFL

Lucas Massemin

Table des matières

Introduction

Aim of the document

This document aims at explaining both functional and technical aspects of the Json editor program. The Execution procedure section further explains how the program can be launched and exploited.

Modifications table

The table below keeps track of the different versions and should be modified in case of major changes.

Autor	Version	Date	Modification
Lucas Massemin	1.0	07/06/2018	Initial deployment

Aim of the Json editor

This tool has been designed to facilitate the modification of json files

containing a description of muscles and joints in a drosophila model.

.

Requirements

Data

Presentation of the attributes

The json file to modify must be a description of a drosophila model.

We define a drosophila model to be a set of joints and muscles, each having several parameters, referred to as 'attributes'. The different attributes are listed in tables below.

Muscle Parameters:

- lopt : Muscle optimal fiber length [m]
- lslack : Muscle tendon slack length [m]
- theta_ref(θref) : Joint angle at which muscle is at its resting length [deg]
- theta_max(θmax) : Joint angle at which maximal muscle moment arm [deg]
- pennation : Fiber angle [deg]
- r0 : Maximum muscle moment arm [m]
- fmax : Maximum muscle force [N]
- vmax : Maximum muscle velocity [lopt/s]
- joint_attach : Joint to which the muscle exerts a moment
- direction : 'clockwise / cclockwise' : Direction of muscle moment
- muscle_type : 'mono' if muscle spans across only one joint. 'bi' if muscle spans c

Joint Parameters:

- joint_type : 'CONSTANT' - Muscle moment arm computation type
- theta_min : Minimum joint angle [deg]
- theta_max : Maximum joint angle [deg]
- reference_angle : Offset in joint angle [deg]

Python instantiation of a muscle/joint

An instance of a joint/muscle is given in a python dictionary that describes all the attributes, as presented in figure 1.

```
muscle_dict = {"theta_max": theta_max, "l_opt": l_opt, "name": name, "v_max": v_max,
               "joint_attach": joint_attach, "f_max": f_max, "l_slack": l_slack,
               "pennation": pennation, "r_0": r_0, "type": type, "theta_ref": theta_ref}
```

```
joint_dict = {"theta_max": theta_max, "name": name, "theta_min": theta_min, "type": type,
              "reference_angle": ref_angle}
```

fig 1 - instantiation of joint and muscle dictionaries

Content of a valid json file

The json file should consist of a dictionary with two keys, 'joints' and 'muscles', whose associated values are lists of joint and muscle dictionaries respectively.

Moreover, a header should be specified, under the form of key-value 'header' and 'WT-OK'.

If the header is not specified, the program will write it if the user agrees.

The content of a json file without header could be as shown in figure 2, with only one muscle and one joint.

```
{'joints' : [{"theta_max": theta_max, "name": name, "theta_min": theta_min, "type": type,
              "reference_angle": ref_angle}],

 'muscles' : [{"theta_max": theta_max, "l_opt": l_opt, "name": name, "v_max": v_max,
               "joint_attach": joint_attach, "f_max": f_max, "l_slack": l_slack,
               "pennation": pennation, "r_0": r_0, "type": type, "theta_ref": theta_ref}]
 }
```

fig 2 – content of a valid json file without header

Programming environment

The program requires python 3.* to be installed, as well as the following libraries :

- PyQt5
- json
- sys
- copy

json, sys and copy are already included in the python standard version.

Execution procedure

Once your json data file is available, you can launch the program by typing "`python3 json_editor.py`" in your bash.

The windows are connected between them in a specific way, presented below as an undirected graph. Circles correspond to windows/interfaces, whereas a link between two circles means that one can generate the other and vice versa.

The user starts at the top and follows the links until the desired action has been executed.

A circle can be left using another link than the one that led to it if and only if the direction (up-to-down or down-to-up) is the same. It is always possible to leave a circle using the link that led to it. The creation options is a special case, as one cannot change muscle parameters starting from the joint window and vice versa

As an example:

[create joint → creation options → create muscle] is not allowed

[create joint → creation options → create joint] is allowed

[choose action → create joint → creation options] is allowed

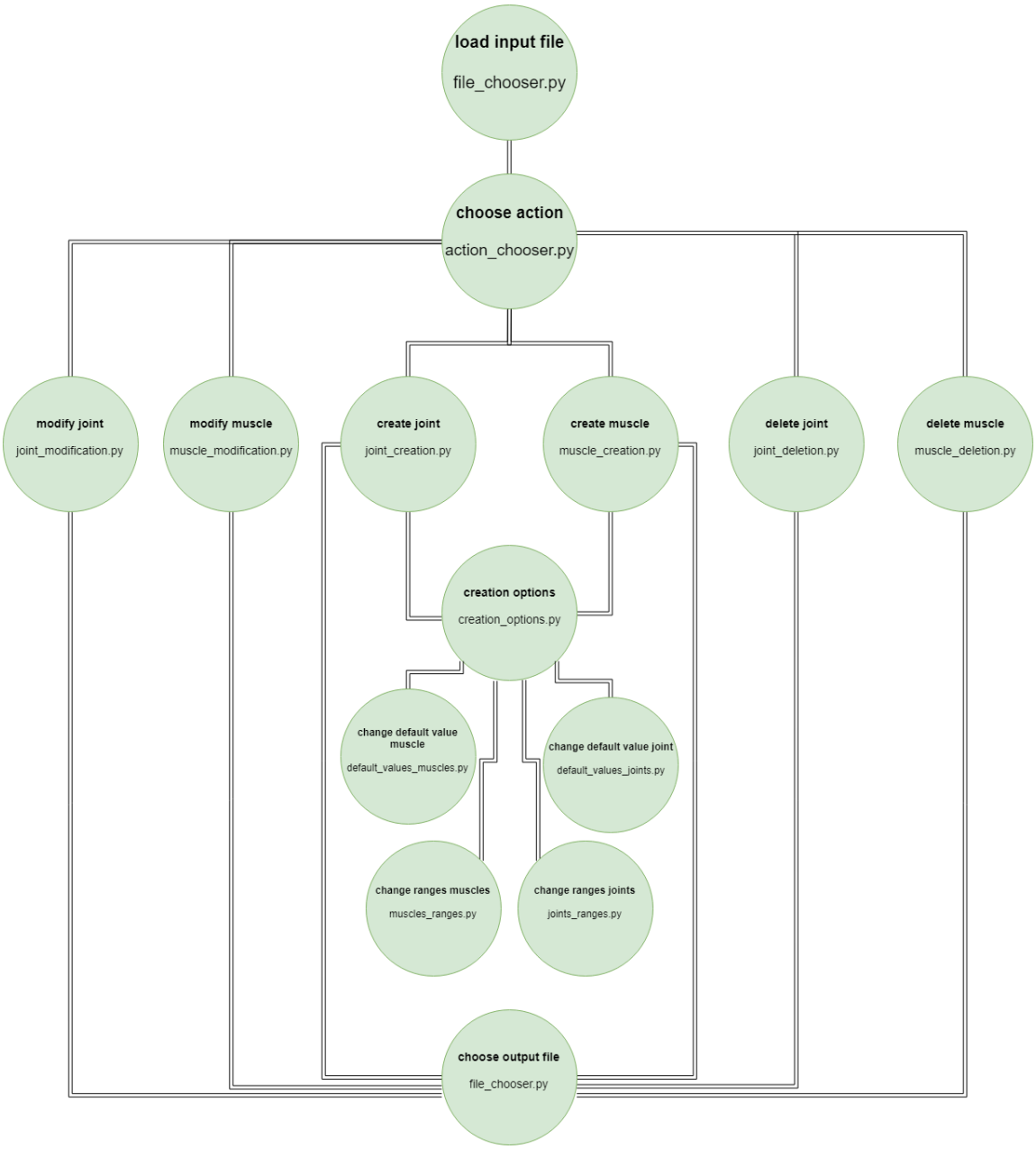


fig 3 – organization of Json editor inner interfaces

Features

Header

The files compatible with json editor are marked with a header. When loading a file that does not contain the valid header, a warning is generated, which is useful to think twice before importing invalid files. In case the user knows the file is valid, json editor will write the header.

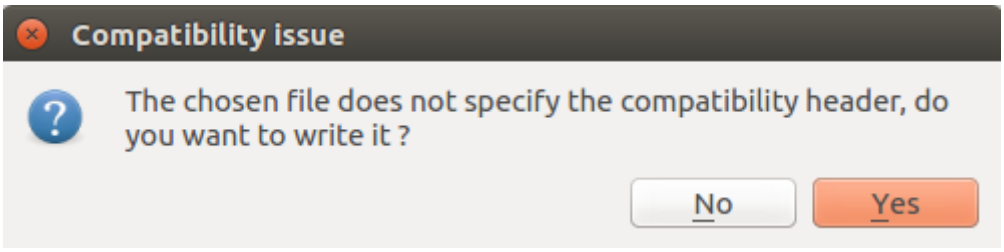
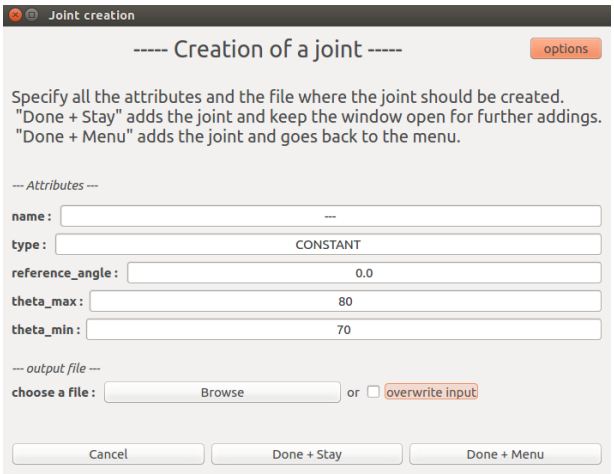


fig 4 – the header warning generated by Json editor

Joint/Muscle Creation

Once the json file is loaded, one can choose to add a joint/muscle to its list and to either overwrite the file or save the result in another file. To do so, one needs to specify all the attributes and choose an output file.

The creation of the muscle is similar. Note that the 'joint_attach' attribute takes the name of an existing joint, which means that, in case you want to create a new muscle attached to a new joint, you will have to create the joint first.



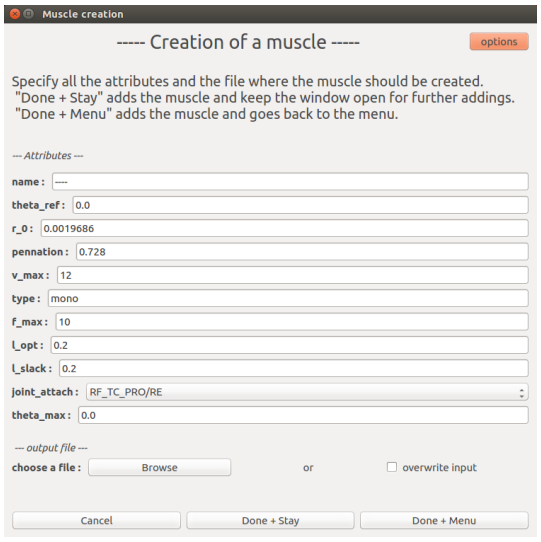


fig 5 – the interfaces to create joints and muscles

Default values

The creation of multiple joints/muscles with almost the same parameters can be painful because requires typing the same values every time.

To solve this issue, it is possible to declare a default value for each attribute.

The default values are loaded and written to a file named 'defaults.json' so that one can keep them even after closing the program.

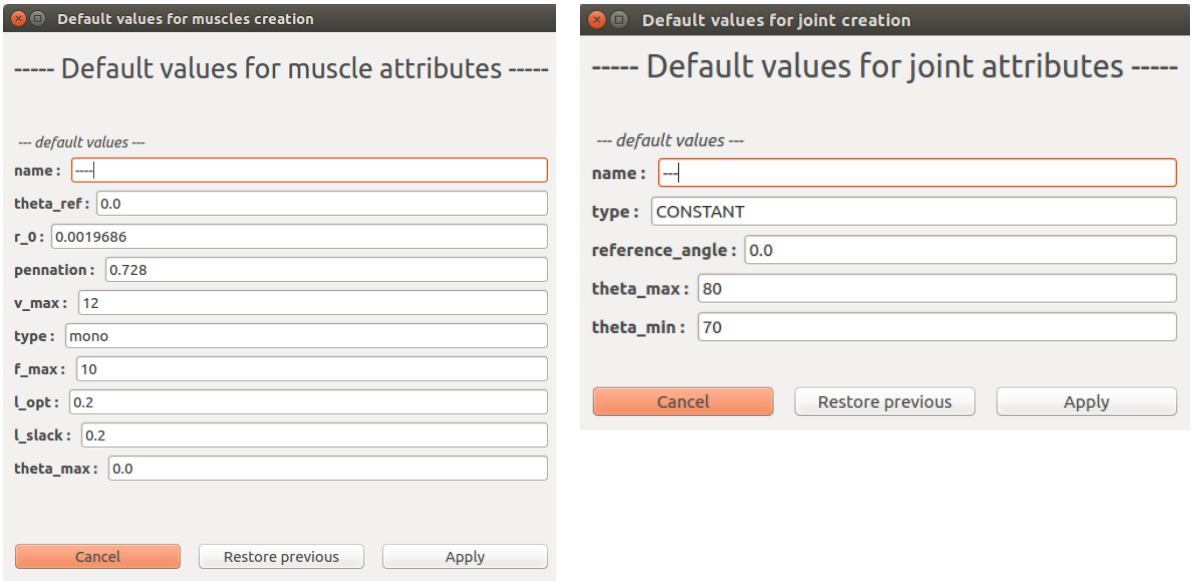


fig 6 – the interfaces to change default values for muscles and joints

Ranges for attributes

It is important that the attributes of both muscles and joints have likely values.

To ensure this is the case, one can define ranges for numerical attributes.

The ranges are loaded and written to a file named ‘ranges.json’ so that one can keep them even after closing the program.

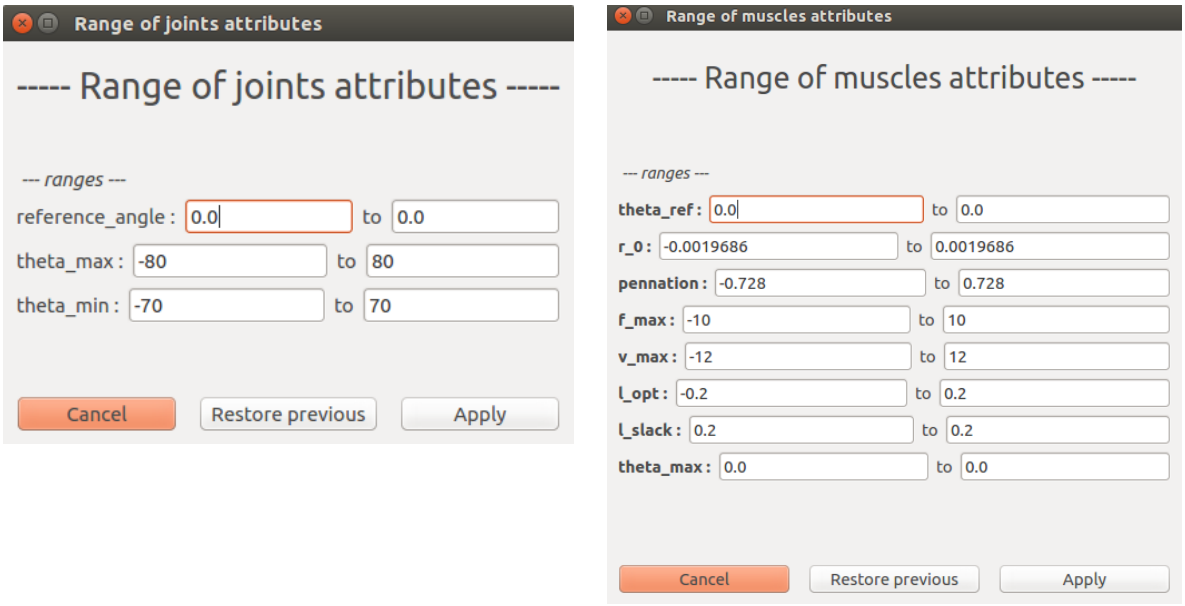


fig 7 – the interfaces to change ranges of attributes for muscles and joints

Joint/Muscle modification

One can modify the attributes of existing joints and muscles. The new numerical values should be in the ranges defined above.

The first step is to select one or several joints/muscles thanks to the checkable combobox, then to give the new values for the attributes that should be modified and to choose an output file.

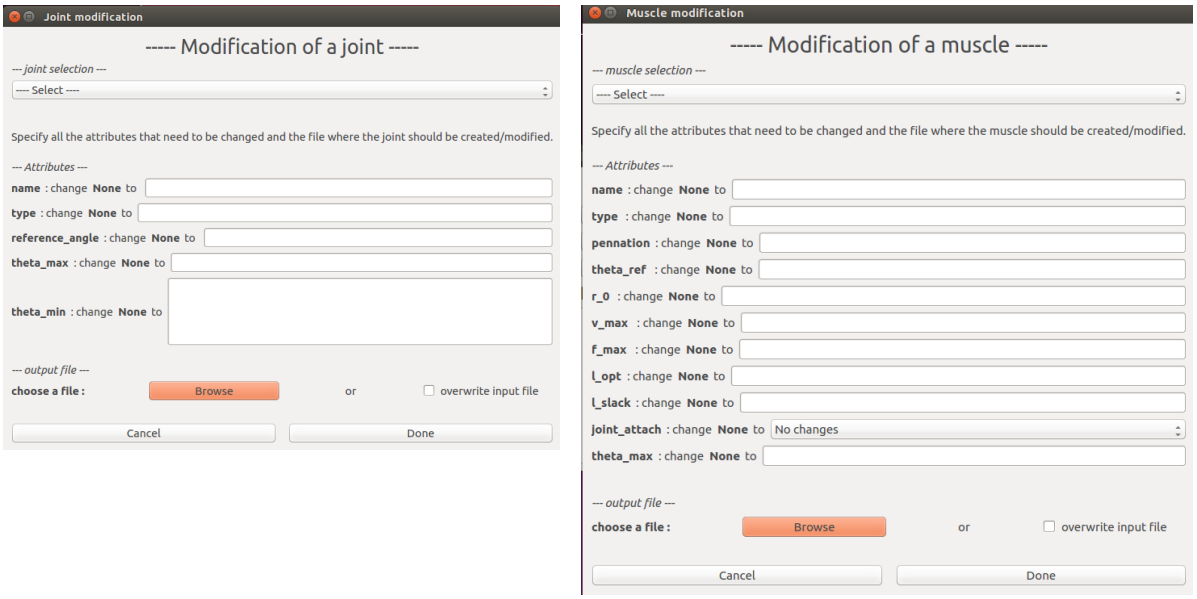


fig 8 – the interfaces to modify joints and muscles

Joint/Muscle deletion

One can remove existing joints and muscles and save the result in any file.

The first step is to select one or several joints/muscles to delete thanks to the checkable combobox. Once this is done, it remains to choose an output file.

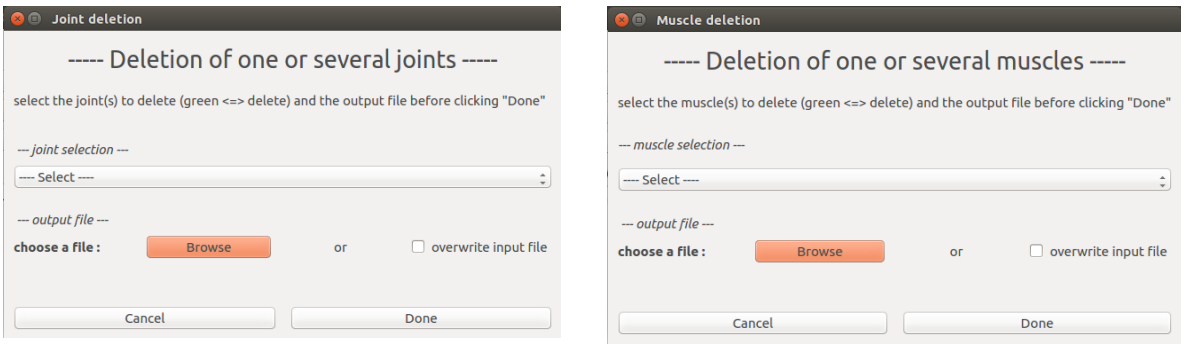


fig 9 – the interfaces to delete joints and muscles

History

The history at the bottom of the ActionChooser interface allows the user to see what has been done since the program was launched.

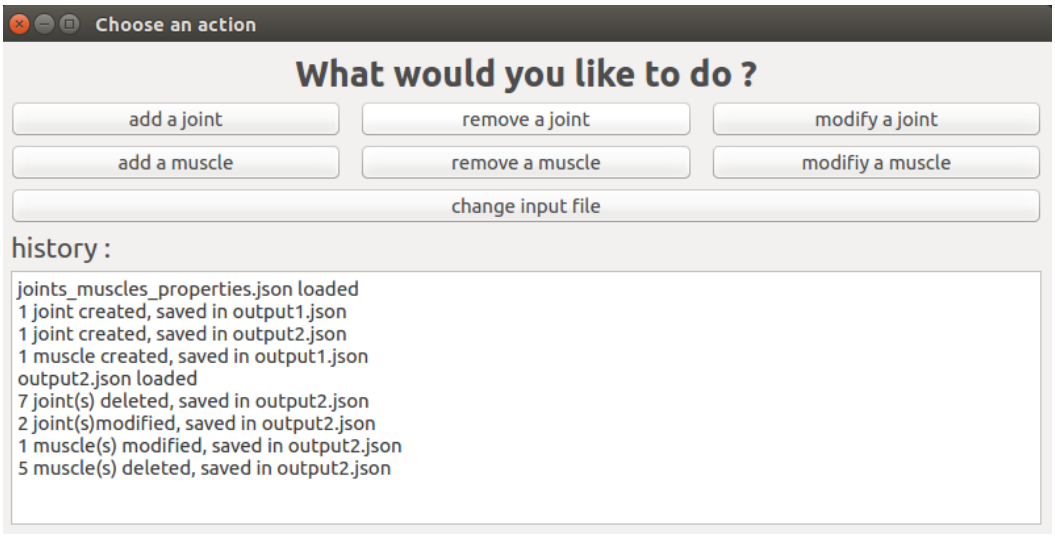


fig 10 - a history with nine events

Change input file

The user can change input file without closing the program by clicking on the 'change input file' button. In figure 10, we see that 'output2.json' has been loaded this way.

Implementation

Qt Designer

The interfaces have been created using QtDesigner. It is a graphical interface to create graphical interfaces, saved as ".ui" files.

Those ".ui" files were then transformed into python files using pyuic5 and loaded in the "ui" field of QDialog windows.

The ui files are in the folder "ui" and can be loaded into QtDesigner to modify the interfaces.

The python modified ui files are in the folder "pyui", whereas the interfaces "behaviours" are in the files at the top level.

File-Interface matching

Each interface is defined by a "pyui" file and a behaviour.

The pyui file defines how the interface should look like, whereas the behaviour file defines what actions should be done when a component is clicked.

The behaviour file defines a class which triggers the interface when instantiated. In figure 11 you can find a matching between the interfaces and the files that define them.

Interface function	Associated behaviour file	Associated pyui file
Load file	file_chooser.py	ui_file_chooser.py
Joint creation	joint_creation.py	ui_joint_creation.py
Joint modification	joint_modification.py	ui_joint_modification.py
Joint deletion	joint_deletion.py	ui_joint_deletion.py
Change joint ranges	joints_ranges.py	ui_joints_ranges.py
Change joint default values	default_values_joints.py	ui_default_values_joints.py
Muscle creation	muscle_creation.py	ui_muscle_creation.py
Muscle modification	muscle_modification.py	ui_muscle_modification.py
Muscle deletion	muscle_deletion.py	ui_muscle_deletion.py
Change muscle ranges	muscles_ranges.py	ui_muscles_ranges.py
Change muscle default values	default_values_muscles.py	ui_default_values_muscles.py
Options	creation_options.py	ui_creation_options.py
Choose an action	action_chooser.py	ui_action_chooser.py

fig 11 – The matching between function-behaviour-pyui

Classes

FileChooser

The class used to choose a file.

Function name	Function behaviour
connect_buttons(self)	Connects the components of the ui to their corresponding function
open_file_name_dialog(self)	Retrieves a filename chosen by the user

ActionChooser

Triggers the main window of the editor from which you can choose what to do.

Function name	Function behaviour
center_window(self)	Centers the window
connect_buttons(self)	Connects the components of the ui to their corresponding function
add_joint(self)	Triggers the interface to add a joint
modify_joint(self)	Triggers the interface to modify a joint
remove_joint(self)	Triggers the interface to remove a joint
add_muscle(self)	Triggers the interface to add a muscle
modify_muscle(self)	Triggers the interface to modify a muscle
remove_muscle(self)	Triggers the interface to remove a muscle
change_input_file(self)	Triggers the interface to change the input file
load_data(self)	Loads the data located in the input file

JointCreation

Triggers the interface from which you can add a joint.

Function name	Function behaviour
set_default(self)	Sets the defaults values in the lineEdit components of the interface
define_ranges(self)	Records the ranges for joints
connect_buttons(self)	Connects the components of the ui to their corresponding function
options(self)	Called when a user wants to see the options, triggers the interface
overwrite(self)	Called when a user decides whether to overwrite the input file or not, changes the inner field “output_name”
get_inputs(self)	Reads and returns the inputs from the LineEdit components of the interface
write_and_stay(self)	Writes the file to the output file and let the window open, returns True iff write was successful
error_message(self, name, attribute, min_value, max_value)	Generates a warning if the given 'attribute' is not between 'min_value' and 'max_value'
write_and_leave(self)	Tries to create a joint and to close the window. Let the window open if any error occurs.
exists(self, name)	Returns true iff data contains a joint named with the given 'name'
close_window (self)	Closes the window
open_file_name_dialog(self)	Retrieves a filename chosen by the user

MuscleCreation

Triggers the interface from which you can add a muscle.

Function name	Function behaviour
set_joints_combobox(self)	Fills the combobox component of the interface with the names of all joints
write_and_stay(self)	Writes the file to the output file and let the window open, returns True iff write was successful
error_message(self, name, attribute, min_value, max_value)	Generates a warning if the given 'attribute' is not between 'min_value' and 'max_value'
open_file_name_dialog(self)	Retrieves a filename chosen by the user
set_default(self)	Sets the defaults values in the lineEdit components of the interface
define_ranges(self)	Records the ranges for muscles
connect_buttons(self)	Connects the components of the ui to their corresponding function
options(self)	Called when a user wants to see the options, triggers the interface
overwrite(self)	Called when a user decides whether to overwrite the input file or not, changes the inner field “output_name”
get_inputs(self)	Reads and returns the inputs from the LineEdit components of the interface
write_and_leave(self)	Tries to create a joint and to close the window. Let the window open if any error occurs.
close_window(self)	Closes the window
exists(self, name)	Returns true iff data contains a muscle named with the given 'name'

JointModification

Triggers the interface from which you can modify a joint. Note that the pyui file has been modified to use a checkable combobox, not known by the QtDesigner.

Function name	Function behaviour
open_file_name_dialog(self)	Retrieves a filename chosen by the user
close_window(self)	Closes the window
overwrite(self)	Called when a user decides whether to overwrite the input file or not, changes the inner field “output_name”
get_inputs(self)	Reads and returns the inputs from the LineEdit components of the interface
define_ranges(self)	Records the ranges for joints
connect_buttons(self)	Connects the components of the ui to their corresponding function
error_message(self, name, attribute, min_value, max_value)	Generates a warning if the given 'attribute' is not between 'min_value' and 'max_value'
set_joints_combobox(self)	Fills the checkable combobox component of the interface with the names of all joints + ‘All joints’
set_current_values(self)	Writes the current values of the attributes of the selected joints on the interface
joints_list(self)	Returns a list containing the names of all joints
modify(self)	Called when a user chooses to modify the selected joints. Modifies them and save result in output file
set_for_selected(self, attribute_name, value)	Modifies the data (gives value 'value' to attribute 'attribute_name') for selected joints
change_selected_joints(self, index)	Called when a user changes her selection of joints to be modified, records her selection

MuscleModification

Triggers the interface from which you can modify a muscle. Note that the pyui file has been modified to use a checkable combobox, not known by the QtDesigner.

Function name	Function behaviour
open_file_name_dialog(self)	Retrieves a filename chosen by the user
close_window(self)	Closes the window
overwrite(self)	Called when a user decides whether to overwrite the input file or not, changes the inner field “output_name”
get_inputs(self)	Reads and returns the inputs from the LineEdit components of the interface
connect_buttons(self)	Connects the components of the ui to their corresponding function
error_message(self, name, attribute, min_value, max_value)	Generates a warning if the given 'attribute' is not between 'min_value' and 'max_value'
set_joints_combobox(self)	Fills the joint_attach combobox component of the interface with the names of all joints
define_ranges(self)	Records the ranges for muscles
set_current_values(self)	Writes the current values of the attributes of the selected muscles on the interface
set_muscles_combobox(self)	Fills the selection combobox component of the interface with the names of all muscles + 'All muscles'
muscles_list(self)	Returns a list containing the names of all muscles
modify(self)	Called when a user chooses to modify the selected muscles. Modifies them and save result in output file
set_for_selected(self, attribute_name, value)	Modifies the data (gives value 'value' to attribute 'attribute_name') for selected muscles
change_selected_muscles(self, index)	Called when a user changes her selection of muscles to be modified, records her selection

JointDeletion

Triggers the interface from which you can delete a joint. Note that the pyui file has been modified to use a checkable combobox, not known by the QtDesigner.

Function name	Function behaviour
connect_buttons(self)	Connects the components of the ui to their corresponding function
open_file_name_dialog(self)	Retrieves a filename chosen by the user
close_window(self)	Closes the window
overwrite(self)	Called when a user decides whether to overwrite the input file or not, changes the inner field “output_name”
set_joints_combobox(self)	Fills the combobox component of the interface with the names of all joints + 'All joints'
joints_list(self)	Returns a list containing the names of all joints
change_selected_joints(self, index)	Called when a user changes her selection of joints to be deleted, record her selection
deletion_done(self)	Called when a user chooses to delete the selected joints. Deletes them and save result in output file

MuscleDeletion

Triggers the interface from which you can delete a muscle. Note that the pyui file has been modified to use a checkable combobox, not known by the QtDesigner.

Function name	Function behaviour
connect_buttons(self)	Connects the components of the ui to their corresponding function
open_file_name_dialog(self)	Retrieves a filename chosen by the user
close_window(self)	Closes the window
overwrite(self)	Called when a user decides whether to overwrite the input file or not, changes the inner field “output_name”
set_muscles_combobox(self)	Fills the combobox component of the interface with the names of all joints + 'All muscles
muscles_list(self)	Returns a list containing the names of all muscles
change_selected_muscles(self, index)	Called when a user changes her selection of muscles to be deleted, record her selection
deletion_done(self)	Called when a user chooses to delete the selected muscles. Deletes them and save result in output file

CreationOptions

Triggers the interface from which you can choose to modify ranges or default values.

Function name	Function behaviour
connect_buttons(self)	Connects the components of the ui to their corresponding function
close_window(self)	Closes the window
change_default(self)	Triggers an interface to change the default values of the muscles/joints depending on the type
change_ranges(self)	Triggers an interface to change the range of the muscles/joints depending on the type

MusclesRanges and JointRanges

Trigger the interfaces from which you can modify ranges of muscle and joint attributes respectively.

Function name	Function behaviour
connect_buttons(self)	Connects the components of the ui to their corresponding function
close_window(self)	Closes the window
set_values(self)	Sets the current ranges in the lineEdit components of the interface
apply(self)	Applies the changes made by user iff the ranges are valid
order_wrong(self, name, min_value, max_value)	Generates a warning if 'min_value' is greater than 'max_value'

DefaultValuesMuscles and DefaultValuesJoints

Trigger the interfaces from which you can modify default values of muscle and joint attributes respectively.

Function name	Function behaviour
connect_buttons(self)	Connects the components of the ui to their corresponding function
close_window(self)	Closes the window
apply(self)	Apply the changes made by user
set_values(self)	Sets the defaults values in the lineEdit components of the interface

AppWindow

Launches the program, no interesting function to name.