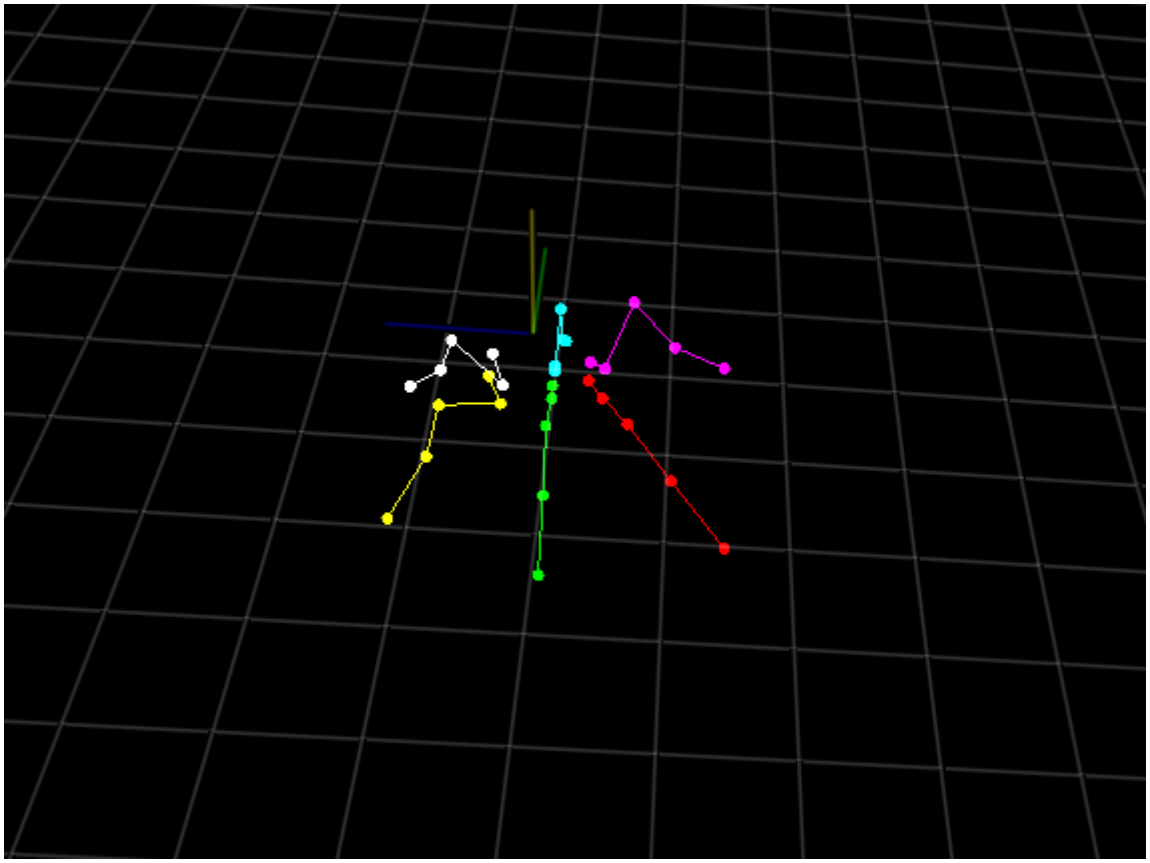
Visualization tool
– Documentation –



BIOROB – EPFL

Lucas Massemin

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Table des matières

Introduction

Aim of the document

This document aims at explaining both functional and technical aspects of the drosophila viewer program. The Features 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 visualization tool

This tool has been designed to facilitate the visualization of simulated drosophila models.

The data should be contained in a .csv file (see “Requirements” section) and consists in 3D coordinates of each leg joints over an arbitrary number of timesteps.

Note that we refer to the claw as to a joint, even if it is not actually one.

Building such a data file can be done thanks to GPS nodes in the Webots software, for instance.

Data file Requirements

File Format

The data should be contained in a .csv file.

It gives the global 3D coordinates for each joint to be displayed (global means that the coordinates do not refer to other positions than the origin).

- The header should be the first line and contain the column names in order.
- The n^{th} line contains the coordinates corresponding to timestep (n-1).

Header rules

Format of column names

The columns of the CSV file should have the format <A>_<B>_<C> where:

- <A> is the leg identifier, only two characters (RF for Right Front for instance)
- <B> is the name of the part, undefined length (« COXA » for instance)
- <C> is the position in 3D, it can be (case not important).
    - « x »
    - « y »
    - « z »

A valid column name would be « RF_COXA_x » for instance

Order of column for a part of leg (Joint)

The order of the coordinates is x, y, z and should be consistent for all fields.

Thus, the columns for RF_COXA will appear in the following order

RF_COXA_x	RF_COXA_y	RF_COXA_z

Order of column for a leg

The joints of a leg must be given in order, from COXA to CLAW, because the leg will be drawn by joining the fields in the order they were given.

RF_COXA_x	RF_COXA_y	RF_COXA_z	RF_FEMUR_x	RF_FEMUR _y	RF_FEMUR _z	…

Finally, all the joints corresponding to a leg A must be “given” successively. It means that if a joint of leg A has been given, you cannot give a joint of a leg B≠A unless you already gave all fields of all joints related to leg A.

Order of legs

No rule: the legs can be given in any order (respecting order of columns for a leg).

Example illustrating header rules

We want to display two legs, LEG1 and LEG2. Each of these legs has three joints, j1, j2 and j3, as shown in figure 1.

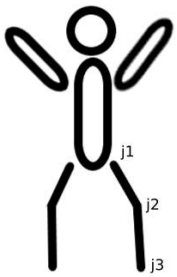


figure 1: a simple model with two articulated legs

Considering only the legs, a valid header would be generated by the snippet of figure 2.

```
header = []
joints = ['j1', 'j2', 'j3']
coords = ['x', 'y', 'z']
legs = ['l1', 'l2']
for l in legs :
    for j in joints :
        for c in coords :
            header.append(l + '_' + j + '_' + c)
```

figure 2: code snippet for header generation

Note that the joints are given in an order so that they can be linked by a line. The combination ['j3', 'j2', 'j1'] would lead to the correct display, but the kinematic matching analysis would be wrong.

The "coords" array cannot be modified, whereas the legs can be given in any order.

Finally, note that the order of iteration (legs-joints-coords) is mandatory to create a valid header.

View of header in Excel (1st line split):

A	B	C	D	E	F	G	H	I
l1_j1_x	l1_j1_y	l1_j1_z	l1_j2_x	l1_j2_y	l1_j2_z	l1_j3_x	l1_j3_y	l1_j3_z

J	K	L	M	N	O	P	Q	R
l2_j1_x	l2_j1_y	l2_j1_z	l2_j2_x	l2_j2_y	l2_j2_z	l2_j3_x	l2_j3_y	l2_j3_z

Data rule: axis choice

The ground is a plane along x and z axis, its purpose is to give an idea of the speed of the drosophila, as well as to indicate the "bottom" of the drosophila.

There is no feature to rotate or hide the ground in the program. If your coordinate system is not consistent with the ground plane, you can either change your coordinate system, or modify the "draw_floor" function in the "droso_viewer.py" file.

Programming environment

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

- PyQtGraph
- PyQt5
- NumPy
- Matplotlib

Execution procedure

Once the data files are available, the program can be launched with python3 by typing "`python3 droso_viewer.py`"

The first task of the user is to find a name for the simulated model, and to indicate the path leading to the csv file by clicking "Browse file". The name should be a non-empty string, there are no further requirements. It will be useful to colour the legs once there are two models, and to do the kinematic matching analysis.

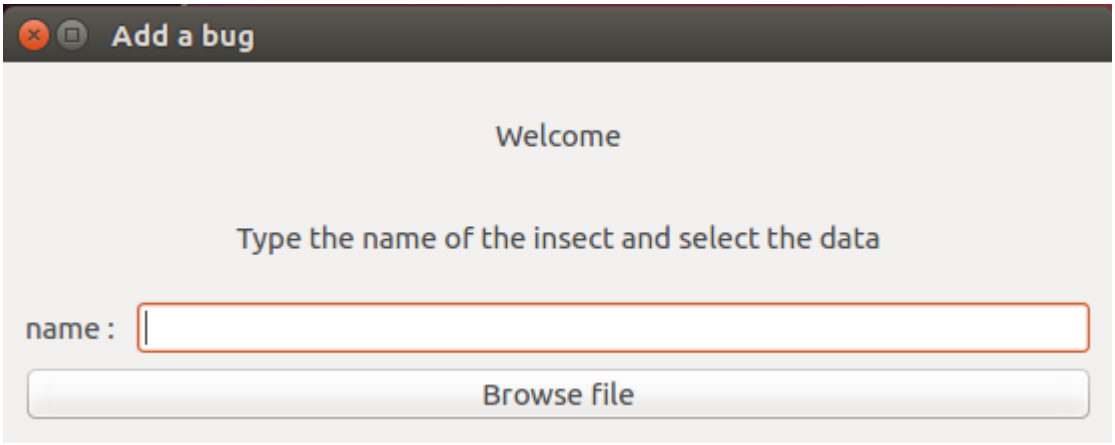


figure 3: Dialog to add new drosophila model

Once the name and the data file are chosen, the main window appears.

All the features will be presented in detail later, for the moment we quickly present the interface :

- The left part is composed of a 3D view of the drosophila model and of a player
- The right side shows plots of chosen joints and arrays with the exact values. It also contains a combobox to choose the fields to plot.
- The central column is used to hide the legs and change their colours.
- The button "Add new Drosophila" allows the user to enter a second drosophila, to compare the gaits.

.

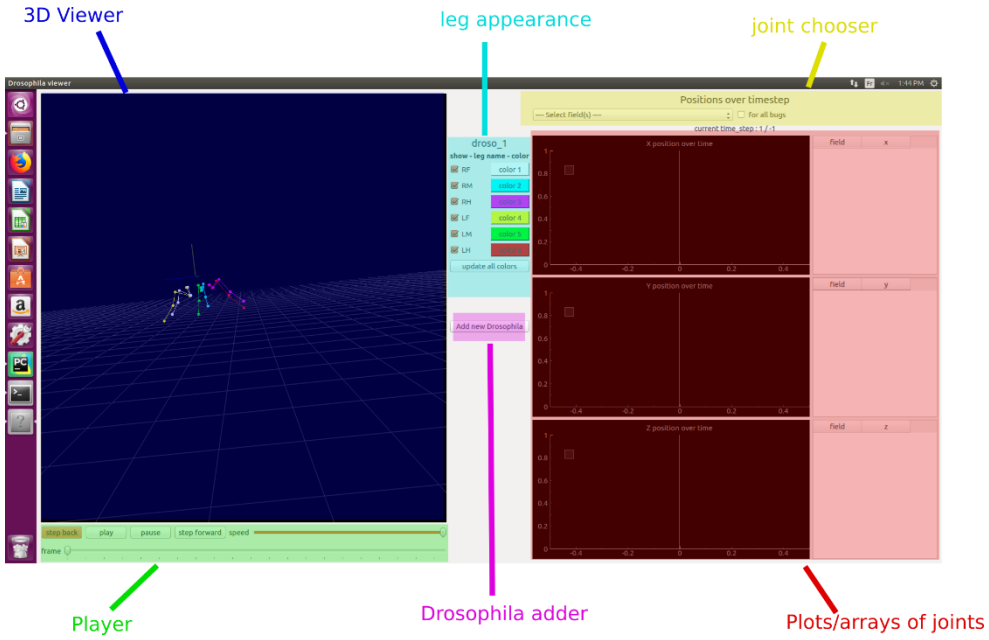


figure 4: Main window of the program

When the button is clicked, a dialog window opens. It is similar to the first one, but also asks if the user wants to do a kinematic analysis and the name of the folder that should contain the analysis, see figure 5.

The kinematic analysis will be explained in the “Features” section.

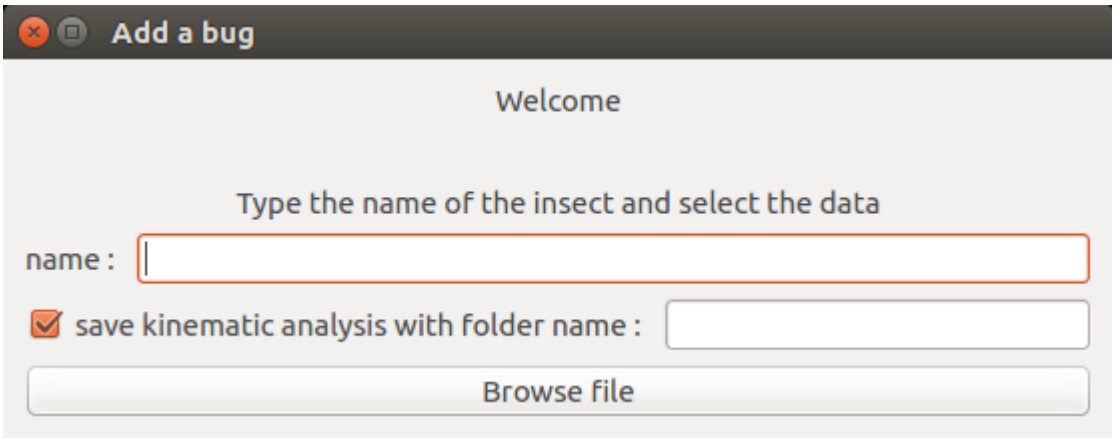


figure 5: adding a second drosophila model

After you browsed the data file, a file system explorer opens to ask where to store the kinematic analysis folder if the checkbox was checked. Once the folder is chosen, the kinematic analysis is launched, and the main window gets darker (see figure 6). This is a normal consequence, and it should come back to the normal in less than a minute.

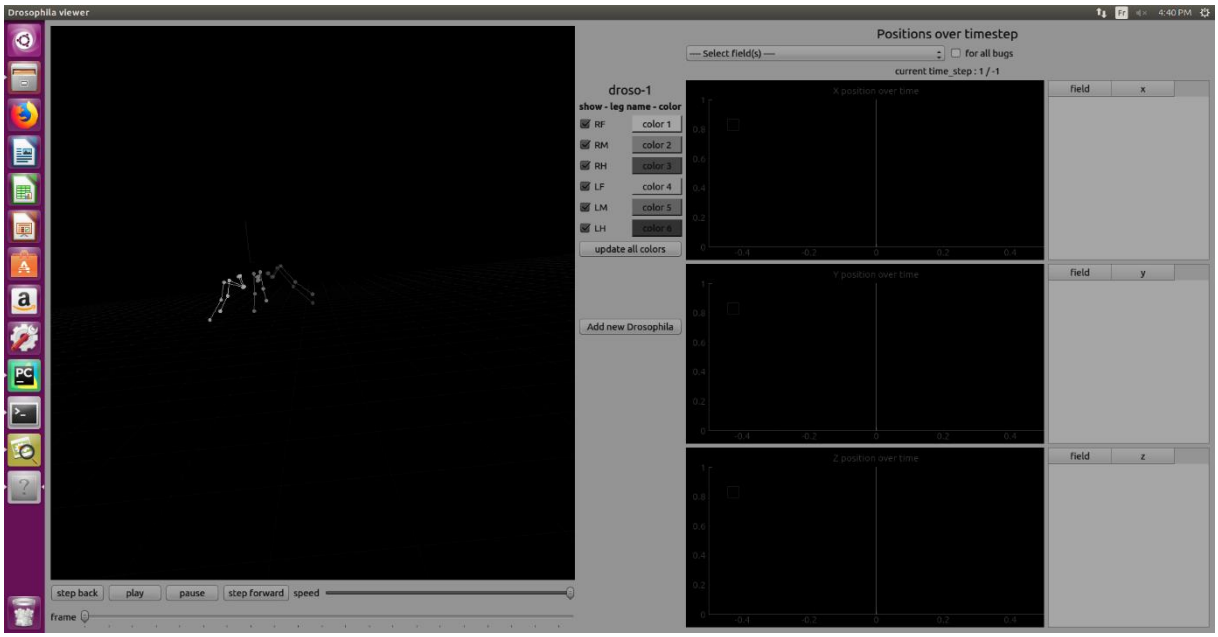


figure 6: darker main window during kinematic analysis

Finally, the window comes back to its initial state except that we can now see two drosophila displayed in the 3D viewer. Also, one can change the appearance of the legs of the second model. The resulting window is shown in figure 7.

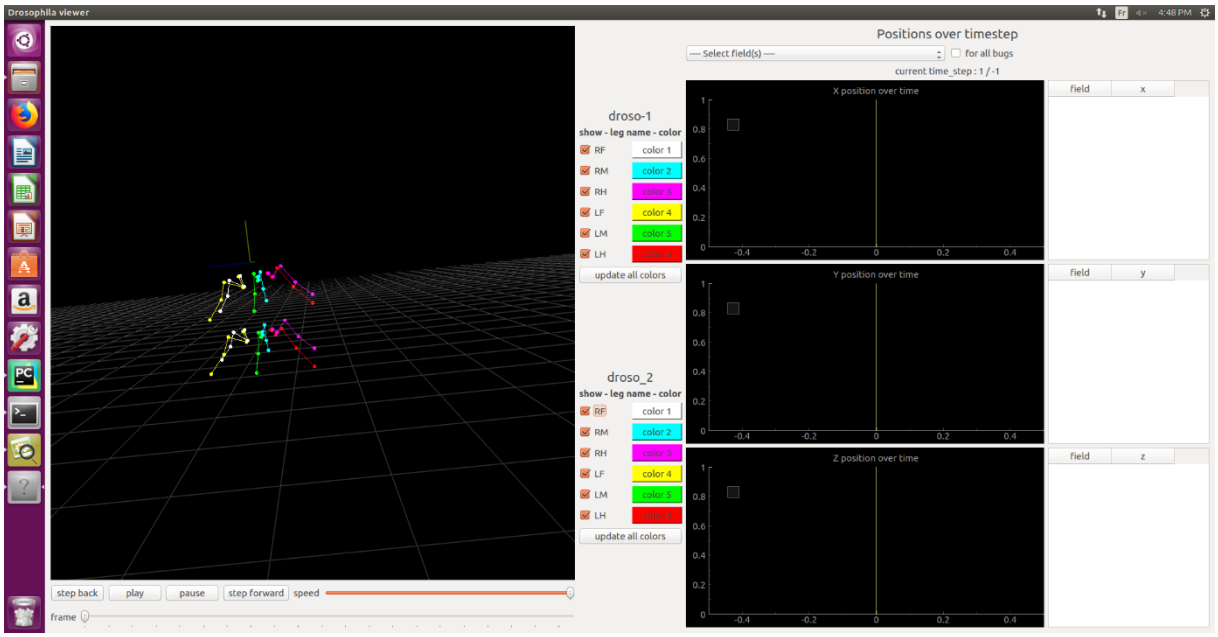


figure 7: Display of two drosophila models simultaneously

Features

The features of the Drosophila viewer are presented below.

Player

The player is situated at the bottom left corner of the main window. As its name suggests, it is used to play the animation.. but not only. What is referred as "timestep" is not the time between each data sample, but the index of the current data sample (timestep n-1 corresponds to the n^{th} position).

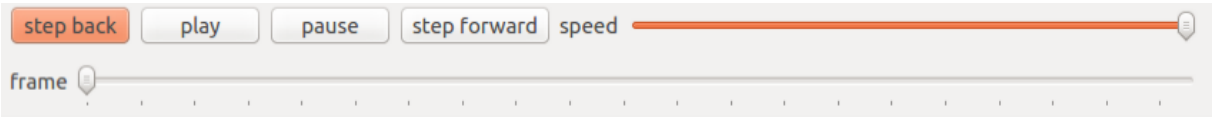


figure 8 : The player

- **"step back"** displays the model(s) at timestep t-1 if current one is t.
- **"play"** plays the animation, it continuously draws the model(s) and increments the timestep.
- **"pause"** pauses the animation.
- **"step forward"** displays the model(s) at timestep t+1 if current one is t.
- The **speed slider** changes the speed of the animation, the more orange you see, the faster it is. One can choose the speed by dragging the slider to the desired position.
- The **frame slider** indicates the current timestep. One can move to the wanted timestep by dragging the slider.

3D viewer

The 3D viewer is situated at the left of the main window. It displays the given data in a 3-dimensional way.

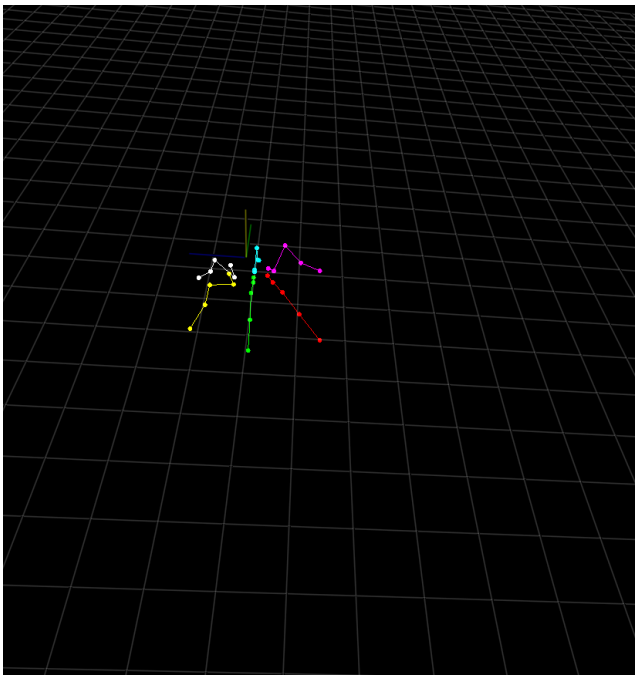


figure 9: A drosophila model represented by the 3D viewer

The points given in the files are represented by small spheres and are linked together by lines using the order of the csv header.

A ground has been added along axis x-z to indicate the “top” of the drosophila and to give an idea of the speed of the model.

The camera and the axis (at the top of the drosophila) will follow the model as the current timestep changes.

One can change the view angle by dragging the 3D viewer and zoom in or out by using the mouse scroll.

Leg appearances

For each model, the leg appearances can be customized using checkboxes and buttons in the central column of the main window.

1. **Show/hide a leg** : check/uncheck the corresponding checkbox.

2. **Change a leg color :** click on the **"**color <i>**"** button next to the name of the leg. A dialog opens and the appearance of the leg changes once a color is picked.

3. **Change the color of all legs in a model**: click the "update all colors" and pick a color. Once the color is selected, all the legs should adopt the new color.

The interface is presented below, numbers refer to the previous enumeration.

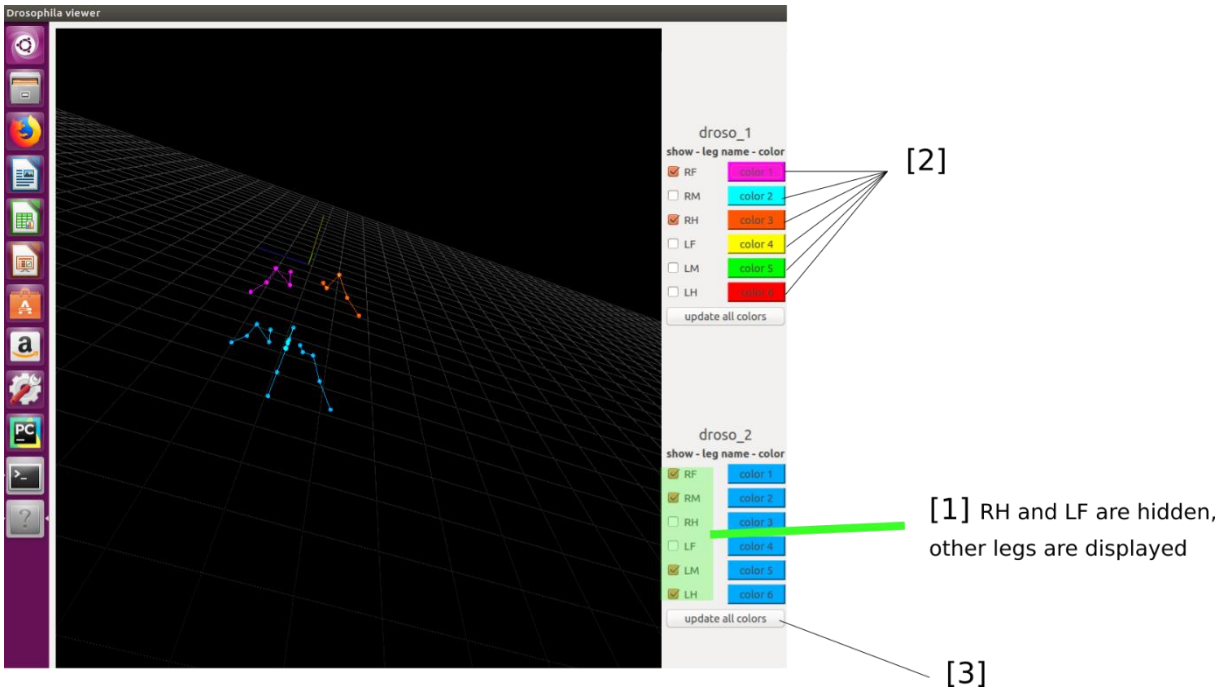


figure 10: customized leg appearance

Plots of interesting fields

Joints whose positions are interesting can be displayed in a plot.

How to specify fields

The user can visualize the evolution of the position of interesting joints (header fields) over timesteps. To do so, one needs to select the interesting fields in the combobox situated at the top right corner of the screen. A list appears once you click on the combobox ("-- select field(s) --"), it contains all the joints of the model(s) that you chose to add to the program.

A field is plotted if and only if it is displayed in a green box in the combobox, otherwise the box is gray and it is not displayed in the plot section.

To select a field to display, simply click on it in the combobox and its box will turn green. If you no longer need to visualize it, click again on its name in the combobox, and its box will turn gray. If the checkbox "for all bugs" is checked, the effect of the click will be extended to the other model, if entered. Consequently, If the user entered models m1 and m2 and the checkbox is checked, turning joint J green/gray for model 1 will turn joint J green/gray for model 2 as well, and vice-versa. The combobox should look as in figure 11.

How to read the plots

If one or several joints are selected, the corresponding curves appear in the three plots below. The ordinate of each plot corresponds to an axis (x, y or z), as indicated by their title, whereas their abscissa indicates the timestep.

In each plot, each field is displayed in a color that is chosen randomly. If the user is not satisfied with the color, turning the box gray and green again is sufficient to get a new one.

One can scroll the plots and export them using a right click, the legend is draggable and the yellow vertical line indicates the current timestep. Note that the label "current time_step" gives the exact timestep.
A table is situated next to each plot in order to give the exact position.
The interface should look as in figure 12.

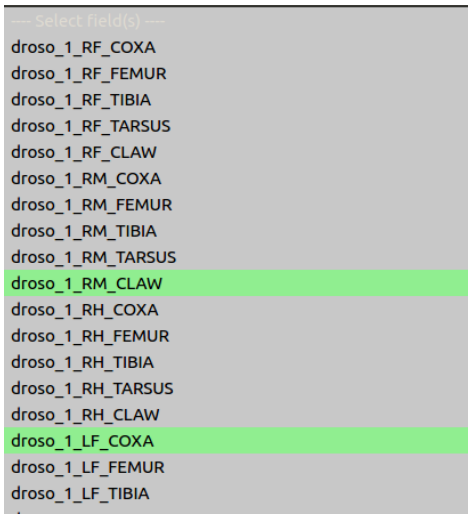


figure 11: a combobox with some fields selected

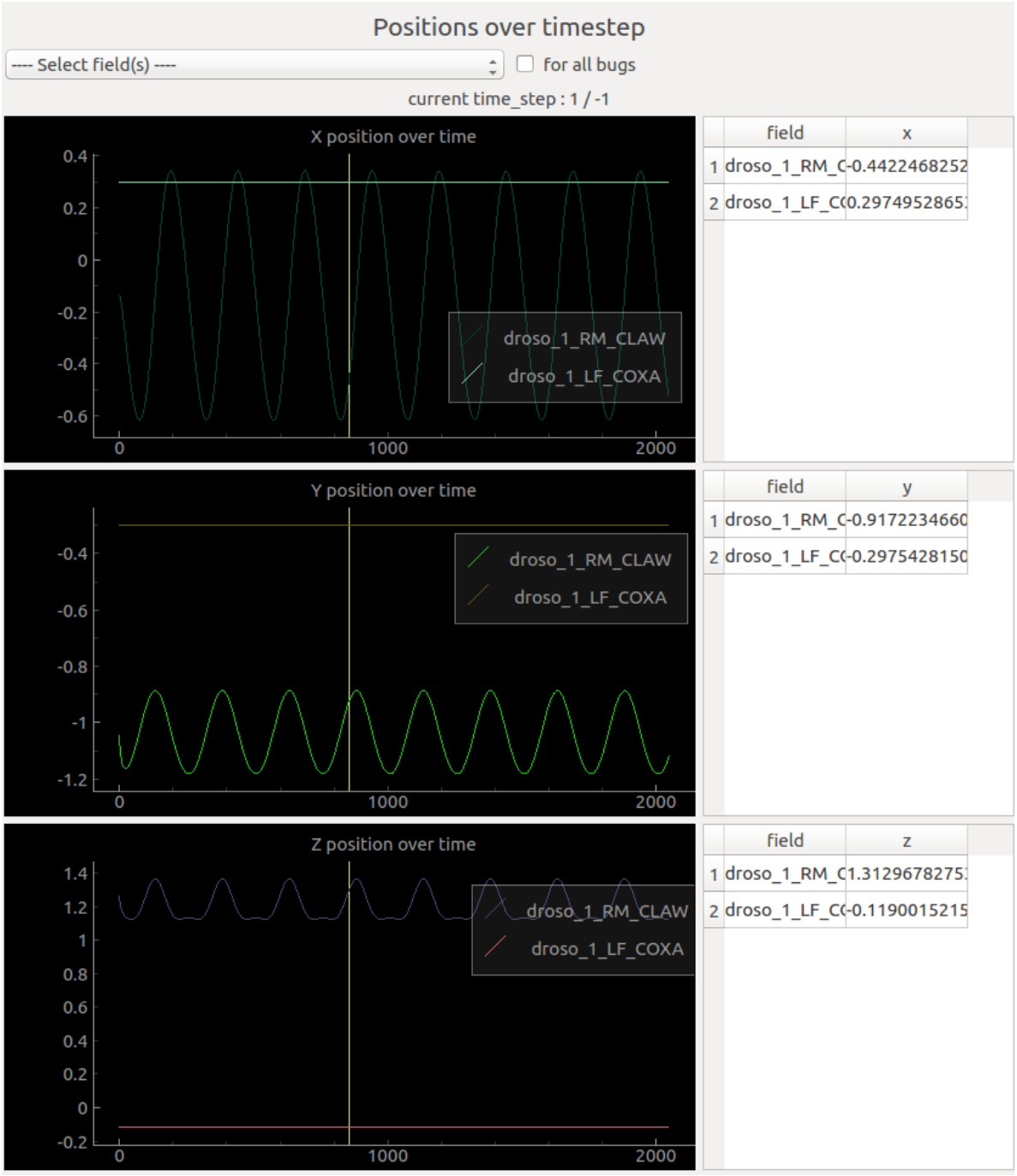


figure 12: plots of x, y, and z positions

Kinematic matching analysis

The user can study how similar two models gaits are by doing a kinematic matching analysis.

Requirements

The kinematic matching analysis has a few requirements:

- Two models have been entered
- The models are along the same axis
- The models share the same number of fields
- The distance between two successive joints is the same for both models
- If joint j1 comes after joint j2 in the header but both are on the same leg, then j2 is in the transform of joint j1. Therefore, the claw should be the last "joint" to be given.
- The models start their gaits at the same time and in the same position. If we see the gait as a motion cycle, models should be at the same moment in the cycle at each timestep.

If some of those assumptions are wrong, the kinematic analysis will not work.

Method

The analysis maps each joint on the referential of the joint that is just higher.
Given the anatomy of the drosophila (figure 13), moving the coxa in space will move all the other joints, moving the femur will move all the joints except the coxa, and moving the claw will not move any other joint.
In order to focus on the movement of a single joint, say the tibia, we need to get rid of the movements of "higher" joints. This is done by subbing off the position of the femur.

In a general case, we can say that

$$pos_{unbiased}(j,t,m) = pos(j,t,m) - pos(higher(j),t,m)$$

Where j is a joint, $\mathrm{pos(j, t, m)}$ returns the 3D coordinates of j in model m at timestep t and $\mathrm{higher(j, t)}$ returns the joint that is just above j (coxa for femur, femur for tibia, etc.). We do not study the movement of the highest joints, as they are normally fixed over time if we ignore the trajectory.

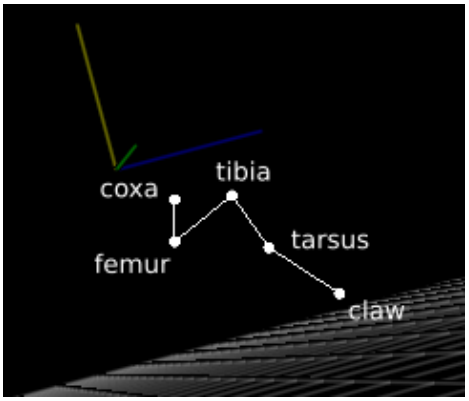


figure 13: A leg of the drosophila model

The idea is to compare the unbiased positions of joint j in each model at each timestep.

Those positions should be very close if the models have similar gaits and should be more distant otherwise.

As we assume that the distance between two successive joints is the same in both models, we can find an upper bound to the distance separating the unbiased positions.

Indeed, the unbiased positions are on a sphere of a radius $dist(j,\ j+1)$ where j and j+1 are the two consecutive joints, so the worst configuration is when they are at the exact opposite (angle of π). In that case, the distance between the two unbiased positions of joint j is upper bounded by $2 * dist(j,\ j+1)$.

Therefore, we can compute the mean matching percentage over timesteps as:

$$mean\_matching(j) = \frac{100}{|T|}\sum_{t=1}^{T}\frac{3D\_dist(pos_{unbiased}(j,t,m_1),\ pos_{unbiased}(j,t,m_2))}{2 * dist(j,\ j+1)}$$

Where 3D_dist is the Euclidean distance in a 3D space, T is the number of timesteps for which both models have data, $m_1$and m_2 are the first and second model respectively.

The overall matching given in the "<name>_kinematic_analysis.txt" file is simply the mean of the mean matching percentages.

Data produced

The data produced by the analysis is put in a folder with the given folder name, situated at the given path.

The summary of the matching analysis is given in a file named "<name>_kinematic_analysis.txt". It contains the matching percentages for all joints, along x, y, z axis and in 3D space.

The overall matching percentage is given at the end of the file.

The folder also contains n other folders, one for each leg, named after the names of the legs. Each of these folders contains plots of matching percentage and position for each joint along each axis (figure 14), as well as a plot of the 3D matching percentage (figure 15).

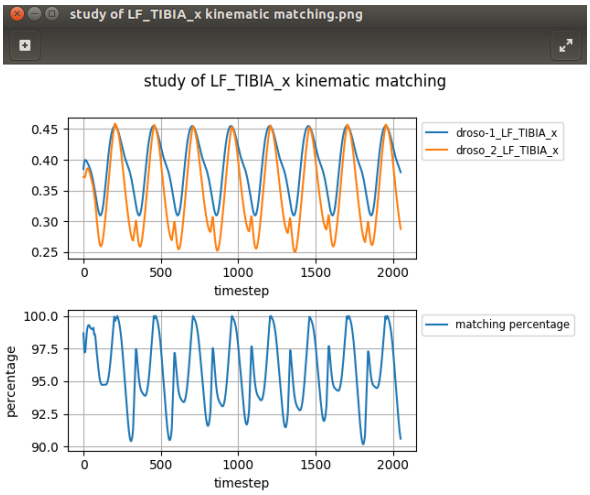


figure 14: plot of distance and matching along x axis

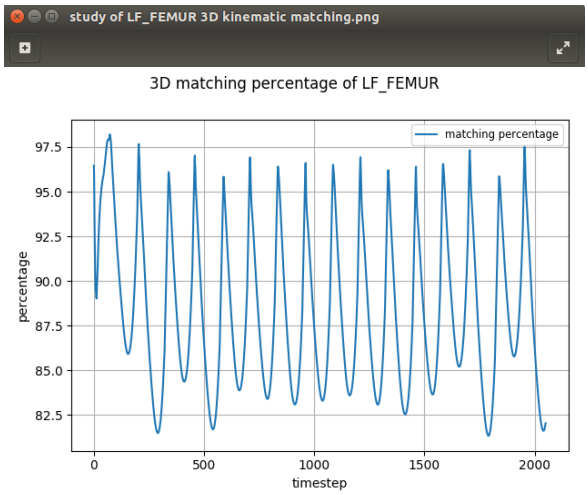


figure 15: plot of 3D matching percentage

Implementation

This section describes the way the drosophila viewer has been implemented.

Qt Designer

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

Those ".ui" files were the 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 at the top level. Note that the use of checkable combobox and custom plots required to modify the generated pyui files.

Classes

The classes used to implement the viewer are presented below.

BugRecord

This class represents a bug record, which is the association of a name with the joints and positions written in a csv file. Furthermore, as soon as a 3D viewer is given, it is responsible for drawing itself on it. Note that legs are indexed from 0.

The name is given to the constructor, as well as the name of the csv file. An overview of the functions is given in the table below, more details can be found in the source code.

Name of the function	description
load_data (self)	Retrieve the positions from the csv file and store them in the 'data' field
hide_leg (self, leg_nb)	The leg with index leg_nb will not be displayed when function 'draw' is called.
show_leg (self, leg_nb)	A leg with index leg_nb that was previously hidden is displayed again at each call of 'draw'
init_plots_blobs (self)	Initializes the components displayed in the graphics view (one GLLinePlotItem and GLScatterPlotItem per leg
color (self, leg_nb)	Returns the color of the leg indexed leg_nb
leg_name (self, leg_nb)	Returns the name of the leg indexed leg_nb
timesteps (self)	Return the number of temporal samples for the record
associate_plot (self, plot)	The drosophila will be displayed in the plot (a GLViewWidget) given in parameter
set_leg_color (self, leg_nb, color)	the leg indexed leg_nb will be drawn with the given color
set_global_color (self, color)	All the legs will be drawn with the given color
value_at (self, field_name, timestep)	Returns the position of the given NOT TRUNCATED field "field_name" at the given timestep
values_of (self, field_name)	Returns the positions (x,y,z) of the TRUNCATED field "field_name" over all timesteps
draw (self, timestep)	Draws the drosophila on the associated plots (no error if no plot is associated) at the position it had at the given timestep

BugAdder

This class is derived from the QDialog class, its interface has been created with QtDesigner.
Instantiating this class is equivalent to ask the user for a new drosophila model.
The constructor takes a number, 1 if it is the first model to be selected, 2 overwise. Depending on the number, the interface will change, as we saw in the "Features" section.

CheckableCombobox

This class is derived from the QComboBox class, it represents a combobox whose fields can be checked (multiple selection) and is used to select the joints to plot.
The implementation is done using a QStandardItemModel that store checkable QStandardItem represented by strings, the fields names.

The few functions used to achieve this goal are presented in the table below.

Name of the function	description
string_to_item (self, string)	Transforms a string into a checkable item to put in the QStandardItemModel
set_values (self, values)	Given the list of strings 'values', transforms them into several QStandardItem and populate a QStandardItemModel.
add_values (self, values)	Same as set_values, but the QStandardItem are added to the current QStandardItemModel so that previous items are not deleted.

KinematicStudy

This class is dedicated to the kinematic matching analysis. It is instantiated using two BugRecord (one for each model) and information concerning the path of the produced data. The analysis begins immediately after the constructor is called.

The processing of the data follows what has been described in the "kinematic matching analysis" section and uses the functions described below to achieve the result.

Name of the function	Description
save_figure (self, datas, labels, save_path, title, vlines_xs=None, y_axis_labels=None)	Saves a figure containing subplots created from the datas and the labels, subplots are aligned vertically.
local_to (self, to_map, referential)	Returns an array of 3D unbiaised positions given the positions 'to_map' and the positions 'referential' of the higher joint
distance (self, vector1, vector2, axis=-1)	Returns the distance between the two given 3D positions along the given axis
mean_matching_perc (self, positions1, positions2, max_dist, axis=-1)	Computes the mean matching percentage as described in the "kinematic matching analysis" section

Ui_DrosoViewer

This is the interface of the main window, it has been generated using QtDesigner. The corresponding .ui file can be opened in the QtDesigner to make changes, but remember further changes will be needed, as you cannot specify all the widgets in the QtDesigner.

Ui_BugAdder

This is the interface of the BugAdder class, it has been generated using QtDesigner. The corresponding .ui file can be opened in the QtDesigner to make changes.

DrosoViewer

The DrosoViewer class is derived from the QMainWindow class, it is the main class of the program and links all the previously seen classes. The table below introduces the main functions.

Name of the function	Description
load_droso (self, droso_nb)	Asks the user for a drosophila model, either the first (droso_nb = 1) or second to be entered
init_droso (self, droso_nb)	Initializes the buttons, show/hide, plot and checkbox for the corresponding drosophila model
init_main_window (self)	Initializes the elements of the main window
change_plot_fields (self, index)	Called when an item in the combobox is checked/unchecked, uses its index to add/remove its corresponding plots
delete_field_to_plot (self, name)	Removes a field corresponding to the given name in the attribute 'fields_to_plot' so that it is no longer plotted
delete_table_item (self, field_name)	Deletes entries in tables corresponding to the given field_name
add_table_row (self)	Adds a row to each of the tables
init_plots (self)	Initializes the plots (setup title, legend and timestep vertical line)
add_plot (self, field_name)	Plots the curves of coordinates x, y, z corresponding to the given field_name in the 3 plotItems
delete_plot (self, field_name)	Removes the curves of coordinates x, y, z corresponding to the given field_name in the 3 plotItems
update_lines (self)	Shifts the vertical timestep lines of the plots to the current timestep
init_tables (self)	Initializes the tables
update_tables (self)	Updates the values in the tables according to the current timestep

show_hide_droso_2 (self, visibility=False)	Shows/hides the components of the interface related to the second model depending on given visibility (True = show, False = hide)
show_hide_widget (self, widget, visibility=False)	Shows/hides a widget of the interface depending on given visibility (True = show, False = hide)
init_speed_slider (self)	Initializes the speed slider
init_frame_slider (self)	Initializes the frame slider
choose_color (self, button, droso_nb)	Changes the color of a leg, called when the corresponding button is clicked
choose_global_color (self, button)	Changes the color of all legs of a drosophila, called when corresponding button is clicked
draw_floor (self)	Draws the floor in the 3D viewer
draw_axis (self)	Draws the axis in the viewer
set_waiting_time (self)	Converts the current position of the slider into a waiting time and set it
frame_chosen (self)	Changes the current timestep according to frame-slider position and draws the drosophila (produces all required timestep-related changes)
show_hide_legs (self, checkbox)	Called by the show/hiders checkboxes, shows/hides corresponding leg
step_back (self)	Called by the step back button, decrements the timestep and makes appropriate changes if animation is not playing
update_frame_slider (self)	Moves the frame slider to set it to current timestep
step_forward (self)	Called by the step forward button, increments the timestep and makes appropriate changes if animation is not playing
pause_animation (self)	Pauses the animation
play_animation (self)	Plays the animation
increment_timestep (self)	Increments the timestep (tore) and changes current timestep label
decrement_timestep (self)	Decrements the timestep (tore) and changes current timestep label
update_timestep (self)	Updates the label that indicate the current timestep
init_viewer (self)	Positions the camera at a good angle, set the timestep and draw the axis
init_leg_names (self, droso_nb)	Initializes the labels indicating the names of the legs of first model if droso_nb = 1, of second model otherwise
draw_dro_with_axis (self)	Draws the drosophila(s), manages camera, tables, axis and vertical lines of plots

The constructor first asks the user for a model thanks to the BugAdder class. Once a valid model is selected, it initializes and display the main window by connecting the interface components to the appropriate functions.