Tools and Framework for Drosophila Models

Author: Lucas Massemín

Supervisors: Pr. Auke Ijspeert, Pr. Pavan Ramdya, S. Ramalingasetty

June 11, 2018
Contents

1 Introduction 2

2 Json editor 2
   2.1 Implementation 3
   2.2 Features 3
   2.3 Problems faced and learning gains 3

3 Drosophila viewer 5
   3.1 Implementation 5
   3.2 Features 5
   3.3 Problems faced and learning gains 8

4 Claws detector 8
   4.1 Implementation 9
      4.1.1 Image preprocessing 9
      4.1.2 Claws detection 9
   4.2 Features 11
   4.3 Problems faced and learning gains 11

5 Analysis of the results 11

6 Conclusion 14
Abstract
The initial goal of this semester project was to design a 3D viewer to visualize the gait data created from simulated drosophila models. The project was thereafter extended to the creation of a claws detector and the design of an editor for json files. This document details the work done to achieve those goals. It focuses on the json editor, the drosophila viewer and the claws detector in order. The tools result to be useful even if the claws detector still needs to be extended.

1 Introduction

This semester project is part of a more important project which consists in the study of the gait of the drosophila melanogaster using neuro-musculoskeletal modelling, see also [1]. Such a modelling includes a parametrization of the muscles and joints whose parameter values are partly decided using an optimization algorithm.

The main goal of this semester project was to facilitate the realization of the modelling, this has been accomplished in several ways, by creating python programs.

The first tool, a json file editor, is designed to facilitate the modification of the joints and muscles parameters that are stored in json files.

Next, a 3D viewer was created to display the gait data of the simulated models.

Finally, a claws detector was built in order to get physical gait data from a walking drosophila.

Figure 1 shows the architecture of the global project.

This document details the work done to reach those goals. The first part focuses on the json editor, we then study the case of the drosophila viewer and finally move on to the claws detector.

2 Json editor

The size of the drosophila melanogaster (fruit fly) makes it hard to measure all the joints and muscles parameters, despite a good knowledge of its anatomy [4]. As said in the introduction, the muscles and joints parameters are stored in json files. The drawback of this method is that it is tedious and prone to errors to modify several parameters at once. Consequently, a tool
was needed to facilitate the parameters modification. This section describes the implementation, the features and the issues associated to the json editor. Further information is given in the documentation.

2.1 Implementation

The json editor (once called webots tweaker) was first designed using tkinter and then translated to PyQt. The two versions are shown in Figure 2 and 3. Those interfaces have been created without using the QtDesigner, which is a graphical interface to create clean graphical interfaces. On the opposite, The final version of the json editor was edited using the QtDesigner.

2.2 Features

Whereas the first two versions were only designed to allow for modification of the json files, the last version also allows the user to create and delete joints and muscles. Furthermore, the processes of creation and modification are speeded up by the possibility to set default values. Finally, One can define ranges to force the values of the parameters.

2.3 Problems faced and learning gains

The creation of a multi-windowed interface takes a lot of time and can be unstimulating in case the windows look alike, which was the case. Moreover, designing such an interface requires to think of all the possible
Figure 2: The very first version of the json editor

Figure 3: The translated version of the json editor
events and to manipulate an important number of components, which cannot be done easily.

The discovering of QtDesigner solved the problem as it allows to easily create and organize interfaces components. I had never done a graphical interface before and programming the json editor taught me how to properly organize a complex interface.

3 Drosophila viewer

Once a drosophila model is defined, one can use it to generate gait data. The gait data consists in a set of 3D coordinates (one per joint) over time.

The visualization of the gait data would be a good indicator of how similar the simulated and original models are, even more if they are displayed side by side. The creation of such an indicator requires to be able to display a drosophila model whose joints are situated at the exact gait coordinates. This task was the initial goal of my semester project.

This section describes the implementation, the features and the issues associated to the drosophila viewer. Further information is given in the documentation.

3.1 Implementation

The drosophila viewer was implemented using PyQt and PyQtgraph. PyQtgraph is a plotting library for python, its 3D viewer is the component used to display the drosophila model(s). The 3D coordinates are stored in a csv file including a header and processed by the drosophila viewer. The joints of the model are displayed as little spheres using the csv file coordinates, whereas the header defines which spheres should be connected together by a line. As we read the rows of the csv file, the spheres are displayed in different positions which gives the feeling that the model is walking.

A screen shot of the drosophila viewer is shown in figure 4.

3.2 Features

Some features are shown in figure 5. We can especially list:

- The 3D viewer
The 3D viewer displays the drosophila model which is the core feature, the player allows the user to navigate amongst the possible positions. The legs can change colors and can be hidden or revealed using the leg appearance part. After selecting one or several joints using the joint chooser, its/their positions are plotted, and their exact values are given in the arrays. Finally, a second drosophila can be added using the bottom button. This might allow to better compare the gaits of two different models. Adding a second drosophila model would result in a window similar to the one displayed in figure 6.

Finally, the program can do a kinematic matching analysis to quantify how close the gaits of the models are. Further features are given in the documentation.
Figure 5: The different features of the drosophila viewer

Figure 6: A drosophila viewer displaying two models
3.3 Problems faced and learning gains

The camera system of PyQtGraph could not focus automatically on the model and needed to be managed manually. I looked into the source code to get more insight and I finally came up with a way to follow an object. Another small issue was a bug in the PyQtGraph library that prevented me to use the classical export method on plots. I solved this problem by designing a custom export-friendly plot widget.

4 Claws detector

The use of physical data to optimize a drosophila model is a powerful approach that has already been used in [3], An experiment has been done at EPFL to capture the gait of a drosophila on a video by using 7 cameras. This was the first step towards physical gait data acquisition, but a precise knowledge of the gait requires the 3D positions of all the joints to be known over time which is a body tracking problem. Once the body tracking has been done for the different cameras, one can simply use the different pixels coordinates combined to the camera positions to find the joints positions in 3D. Note that this dataset would be even more valuable than the one retrieved in [3], as the latter only detected the duration of the stance phases for each leg.

I started to read papers about tracking 3D articulated bodies, belief propagation algorithm[2] seemed especially appropriate and interesting to implement at first sight. After getting better insight, I realized that this tracker would require too much time to be implemented. Moreover, this tracker requires to label the first frame or to give any frame that has no self-occlusion, whereas Mr. Ramdya wanted me to create a detector with no user input. As the end of the semester was getting closer, Mr. Ramdya and I decided to first focus on the claws detection rather than on the detection of all joints.

For this simpler task, I wanted to challenge myself and to come up with my own solution without being influenced by a literature review, which was really exciting. A postponed literature review done by curiosity had me realize that there was nothing of any help.
4.1 Implementation

The claws detector is coded in python with the help of the cv2 library and consists of two main phases. The first one is the preprocessing of the image in order to get only interesting legs parts. The second is the retrieval of claws coordinates using a connected components analysis. I strongly recommend you to read the documentation in order to get a complete high-level description of what was done.

4.1.1 Image preprocessing

The preprocessing of the image gets rid of the ball and focuses on the legs parts, as mentioned earlier. An important work is to tune the parameters to obtain both clean and robust results. Whether a parameter value is better than another was based on the visualization of the results for the first 100 frames captured by the camera. An example of output for the preprocessing of the image in figure 7 is shown in figure 8.

4.1.2 Claws detection

The claws detection part consists in finding the claw associated to each leg by using the preprocessed image.

My first approach was to detect and process the lines in the image in order to infer the claws positions using them. The advantage of this approach is that it could have been used later to find the joints rotation angles. Despite being powerful for the joints rotation angles, this method was not adapted to the claws detection and I had to find another way to spot the claws. The vanilla method is simply to consider the point in each component that is the most distant to the thorax and to consider it as a claw. This works very well when no legs intersect on the image, but the drawback is that it can detect only one leg in case of an intersection. Indeed, intersecting legs are considered as a single connected component. Both cases are illustrated in figures 9 and 10 respectively. Note how only one of the hinge claws of the intersecting legs is detected.

A better approach was not to wrongly assume that a component label can only represent one leg, but to rather try to determine how many legs are associated to each component. To do so, we consider the preprocessed image row by row and detect when several sequences of the same component label are separated by black background. The maximum number of such sequences
Figure 7: Original image of a drosophila

Figure 8: Result of the preprocessing of the image in figure 7
is a direct indicator of how many legs are associated to the component label. There may be noise in the label sequences of course, this is also taken into account. If we find that two legs are associated to the label, we process it in order to spot both claws. The detector only finds all the claws for the intersections of two legs, the intersections with more legs involved being rare.

4.2 Features

The claws detector can spot the claws in any given image that match the requirements. It can save a corresponding image with the claws spotted as shown in figures 11 and 12. It can also merge several images into a video.

4.3 Problems faced and learning gains

I have spent a lot of time on the lines approach before moving on to the current approach, it taught afterwards to be more flexible about my solving strategies. The fact that some legs had the same color than the ball and the existence of the ball irregularities made it challenging to identify the outer legs. I was only able to solve this after I imagined a new method of background detection based on the connected component analysis.

5 Analysis of the results

The json editor and the drosophila viewer only had to match time requirements and to be fully functional. Both time and functionality constraints were respected. The claws detector accuracy has been visually estimated, although it could have been possible to create labels for a subset of images in order to estimate the mean distance between the detected claws coordinates and the actual ones. The ideal processing time expectations were much harder to achieve than for the other programs. An approximation of the current processing time is given in table 1. The current processing time allows for a quite fast claws detection but is not short enough to consider real-time detection. Currently, the claws detector does not use previous images at all when predicting the claws positions. One way to speed up the process would be to
Figure 9: The vanilla claws detection for non-crossing legs

Figure 10: The vanilla claws detection for crossing legs
Figure 11: The input image of the claws detector

Figure 12: The output image of the claws detector
<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-diff</td>
<td>0.0080</td>
</tr>
<tr>
<td>exact ball mask computation</td>
<td>0.0140</td>
</tr>
<tr>
<td>outer legs preprocessing</td>
<td>0.0010</td>
</tr>
<tr>
<td>Inner legs preprocessing</td>
<td>0.0070</td>
</tr>
<tr>
<td>naïve merging (no dilatation/erosion/components removal)</td>
<td>0.0130</td>
</tr>
<tr>
<td>final cleaning</td>
<td>0.0079</td>
</tr>
<tr>
<td><strong>Total pre-processing time</strong></td>
<td><strong>0.0570</strong></td>
</tr>
<tr>
<td>claws detection on preprocessed image</td>
<td>0.0460</td>
</tr>
<tr>
<td><strong>Total time</strong></td>
<td><strong>0.1090</strong></td>
</tr>
</tbody>
</table>

Table 1: The processing time of the claws detector for one image

first check that a claw moved before trying to detect its position again.

6 Conclusion

This project led to the creation of tools that do facilitate the neuro-musculo-skeletal modeling of drosophila models. The joints and muscles parameters should be tweaked using the json editor and will define a simulated model whose gait data can be visualized in the drosophila viewer. The claws detector was a step towards the retrieval of physical gait data but needs to be extended to the other cameras so that 3D positions of claws can be found. All the programs come with their specific documentations.
References


