Horch Lab DLC-to-R Pipeline Walkthrough

On making the bioinformatics more accessible in lab.

Overview

This document focuses on the process of getting from the DLC output file to the graphical representations of the data. In other words, how does the R pipeline take a .csv file and generate a .pdf file for analysis? Big picture, R is a statistical programming language that excels at transforming and visualizing data. We're going to focus on a series of R scripts (which are like bite-sized recipes) that generate the PDFs in question. To that end, we're going to need a script to read in the data, define some useful functions, do a bunch of calculations, and eventually generate and save PDFs of our graphs. The `1_Master.R` script controls this process: if we tell this script the name of our DLC output file, then it'll make the graphs. This walkthrough serves as a supplementary guide to the commented code on GitHub and video tutorial.

Quick Set-up: How to Use the Pipeline

For those interested in simply generating the graphics from the input file, this section parallels the video tutorial in that goal.

Step 1: Download the program. Navigate to https://github.com/mhukill/Crickets-Methods, where the code, along with the entire file structure, can be downloaded. Open `1_Master.R` (we recommend using RStudio).

Step 2: Name your directories. Lines 6-8 of `1_Master.R` specify three directories to be named: one for the .csv input, one for the .pdf output, and one folder that nests these directories. Again, we recommend simply using the GitHub file structure, in which case there is no need to rename anything.

Step 3: Specify your input. Line 14 asks for the name of the target .csv file, without the suffix. To customize the output name, edit line 16; otherwise the output will have the same name as the input with a "_graphs.pdf" as its suffix, instead of ".csv".

Step 4: Enter the decibel range. The pipeline requires manual inputting of the minimum and maximum decibel values. (To understand why this is, see Phase 1: Reading the Data of the next section.) The default step value here is 10 dB, which can be changed at line 33.

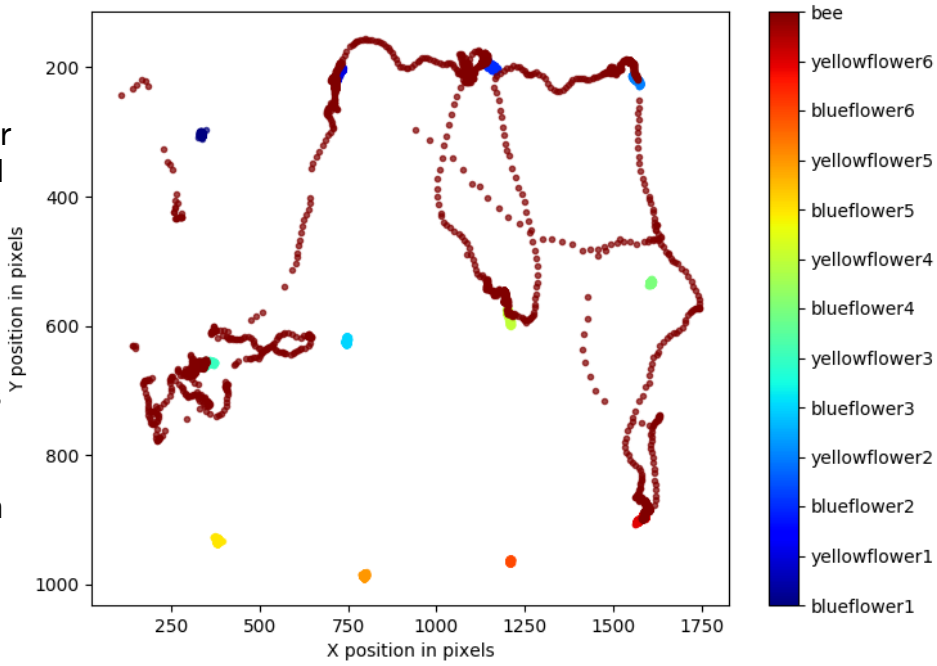
Step 5: Source. Clicking the "source" button with the `1_Master.R` open will now run the entire pipeline. There is no need to open the other scripts; the master script's job is to take care of those processes for you.

Done! To understand in more detail how the pipeline works, please see the remainder of this document.

Phase 1: Reading the Data

Before we begin our visualization and analysis, we need to tell R how to read in our data. Recall that DeepLabCut (DLC) assigns coordinates to several regions of interest—these could be body parts, sound gauges, or the reference pole. We end up with a dataframe (a large table) stored as a comma-separated values (.csv) file. Most folks are used to seeing these types of files via Microsoft Excel. Fortunately, R can handle these files, too; we just need to specify which columns represent which physical regions of interest.

Remember all those reference frames you had to label? Well, DLC certainly does! It used your labels to generate coordinate pairs for each region of interest for all frames. This is an example figure from an experiment in Patty Jones's ecology lab where a bee foraged on different flowers. We've plotted the bee's location as coordinate pairs through time—notice, we're just dealing with simple points in the *xy*-plane! Our crickets work the same way.



Looking at the `3_Reader.R` script, we read in out data as a dataframe (line 2), and name it "data." The remainder of the script reads certain columns of the dataframe and assigns them to specific objects. For example, line 9 reads the fifth column of the dataframe and assigns it to a vector named "wax_x", to reflect the fact that these series of numbers show the *x* coordinate of the wax on the cricket through time. (Note, R creates objects when we assign things to them. We did not need to declare wax_x as a vector in a separate step, unlike other languages.)

```
9  wax_x <- data[,5]
```

Thus, because the sound level sensor is reported in coordinates, we need to specify how to convert those values into decibels. Fortunately, Audacity uses a linear

scaling in its sound bar, which is what DLC tracks. This means we only give a minimum and maximum decibel value, and R will do the rest (Audacity has done the majority of the math in linearizing the dBs).

Phase 2: Defining our Functions

Independently of reading our data, we can teach R, in general, some things we'd like it to do for us. R is a function-based language, so abstractly defining procedures and formulae is its bread and butter. Most of our functions revolve around calculating area: because the cricket behavior can be so erratic, it's often difficult to know exactly what to measure positioning against. We'll get into this more later, but let's briefly consider one of these functions.

For example, consider the function on line 4 of `2_Functions.R`, Pick's Theorem. This formula calculates the area of the polygon formed by two vectors. So, we input two vectors (line 4), and output the area (line 14), after doing some math (the lines in between). Without getting bogged down in the specifics of how the subsequent functions operate, notice that they all include Pick's Theorem at some point. That's right—they're all functions of functions! This is where R shines: imbedded functions. We can break down complicated procedures into bite-sized tasks that are controlled by larger tasks, all operating on general inputs and outputs.

```
3  ## Pick's Theorem: the underlying area calculation formula used in this pipeline
4  picks_theorem <- function(vector1, vector2){ # IN: two vectors; OUT: area of the polygon they form
5    sum1 <- 0
6    for(i in 1:(length(vector1)-1)){
7      sum1 <- sum1 + (vector1[i]*vector2[i+1])
8    }
9    sum2 <- 0
10   for(i in 1:(length(vector2) - 1)){
11     sum2 <- sum2 + (vector2[i]*vector1[i+1])
12   }
13   product <- abs(0.5*(sum1 - sum2))
14   return(product)
15 }
```

We give R vector1 and vector2, which are generic stand-ins for two vectors, and receive product (shown in the "return" line). Note that R doesn't need to be told which data types vectors 1 and 2 are. While convenient, this opens the code up to bugs. If we accidentally give R matrix1 instead of vector1, it'll keep going until it gets stuck, which isn't necessarily obvious for us to catch!

There are also a few graphical functions defined in this script. See the code comments for more details.

Phase 3: Calculating our Quantities of Interest

With our data and function repertoire in-hand, we can proceed to calculating the quantities of interest. See the explanations in the methods paper. The coding details of

these calculations are less important than the idea that we've given R a standard protocol for finding our quantities of interest, and the `4_Calculator.R` script does the work with break-taking rapidity.

Conformation: All relevant quantities are of the form viz_mark_coord, where "viz" stands for visualizing the cricket (conformation helps us see what the crickets look like), "mark" stands for the region of interest that DLC is tracking (such as the cricket's abdomen, left leg, the wax on the stick, etc.), and "coord" reflects either the *x*- or *y*-coordinate. For example, viz_a_x stores all of the *x*-coordinates of the abdomen used in the conformation section. The script then selects certain shots to be graphed. Notice, line 108 defines how many total conformations to plot. These conformations are then evenly spaced throughout the video, and represent the average position across the shot (12 video frames). Lines 108 through 191 perform the relevant calculations.

Body Angle: Next we have to find the angle between the cricket's abdomen and the center line. Lines 9 through 28 go through those calculations, with line 13 being the critical geometric calculation. We then set the center line (where the wax is) as zero, and make the angles positive or negative depending on their deviation from that line.

Variance (twitch): As for the variance, lines 38 through 105 walk through the careful application of our previously defined area formulae. Recall that at the heart of these calculations lives Pick's Theorem.

Phase 4: Generating the Graphic

Finally, we've made it to `5_Grapher.R`. This script takes those generated quantities from the previous script and actually produces the PDF file of interest. Fundamentally, we generate four plots in base R graphics, which are stitched together using the par(mfrow) functions and saved as a PDF. The details of each graph are highly specific, but the code is broken down into segments for those interested enough in how each graphic panel is constructed.