Chronogram Visualization for Tracking Honey Bees

1. Introduction

Honey bees have been a valuable topic in research because of their complex social behavior and their importance to our agriculture [Bee Tracker]. With our software, we aim to provide effective tools to analyze their behavior over a long extended period of time with video tracking and data visualization.

This software provides biologist easy to use tools that facilitate collecting and evaluating patterns of the behavior of bees. The interface itself has the ability to create and record annotations in its video interface by manually labeling bees. This helps save time and resources as collecting big data from thousands of bees can be time and mentally consuming.

This semester, we opted to update the LabelBee web app with a friendlier user interface with the ability for users to register and save analytical data in their accounts. We also wanted to add the ability for users to have access to multiple videos that record bees entering and exiting the honeybee colony. This meant using a front end framework that allow crowd source data analysis, and pairing it with a back end framework that will pull video data from a server, adding to the web interface client and server components.

Another feature that was added to this semester's research pipeline was automatic analysis for bee detection in the video interface, with the ability to differentiate the bee's activity (entering or exiting the colony, doesn't or does have pollen, is it fanning).

Apart from the video interface, one of the software main visualization is a chronogram anograph graph that records and displays a frame log for when the bees enter and exit the colony. As the main tool for annotate the bees behavior, it helps the users visualize the bees' pattern better.

This report will focus mostly in explaining the methods that were used to update the functionality of the chronogram. This includes explaining the technologies that were used and how they were used, including API, libraries, functions and programming languages. It will describe the challenges that we had with the code and how we overcame them. It will also briefly explain the frameworks and API that were also used to turn the software into a client server system. At the end, the report will include the result of this semester's work, a conclusion and our expectations for future work.

2. Related Work

2.1 Past Work

2.1.1 Introduction to the Software Label Bee

The main goal for the Label Bee open source software is to develop a platform that analyses individual insect behavior and acquires new observations into the role of individual diverse behavior on the bee colony performance. It is meant for biologists to make annotations based on observations to joint videos, remote visualization and data acquisition that monitors thousands of marked bees over a extended and continuous period of time [http://bigdbee.hpcf.upr.edu/].

Since the web interface is meant to give access to an abounding amount of high definition videos, it is required we stored them in servers that have the capability to manage data from large datasets. That's why we will be using the resources from the High Performance Computing Facility at Puerto Rico. In the end, the software wants to provide semi-supervised machine learning that has the ability to manually and/or automatically detect different common activities known from honey bee such as pollen carrying or fanning behavior.

The web application that was developed last year already has many of the features mentioned previously. For the purpose of easier understanding, the software's component will be explained in the next section with visual representation of the interface.

2.1.2 First Label Bee Interface

The user's interface is meant to be simple and straightforward for users that either have software development experience or don't have programming experience. The

functionality of the its main elements are shown in Figure 1.

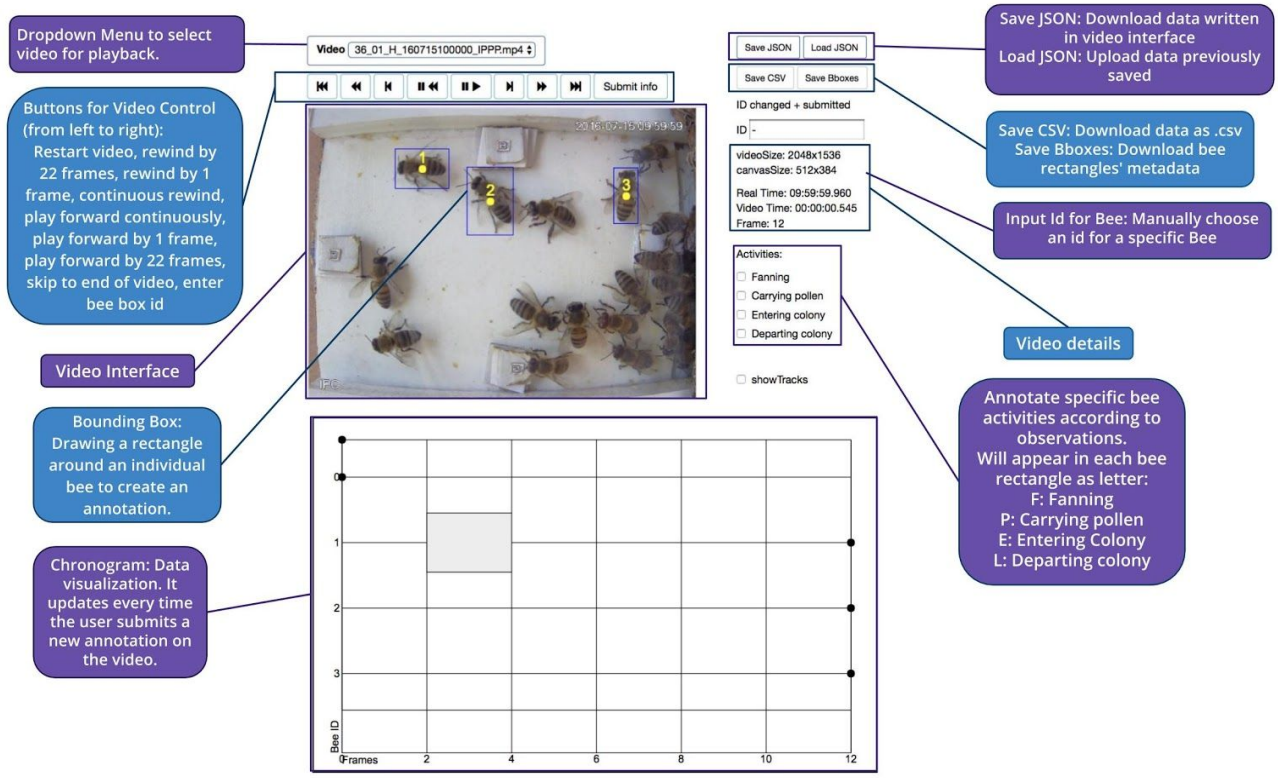

Figure 1: Parts of the software's interface with its description

2.1.3 Software Data Diagram

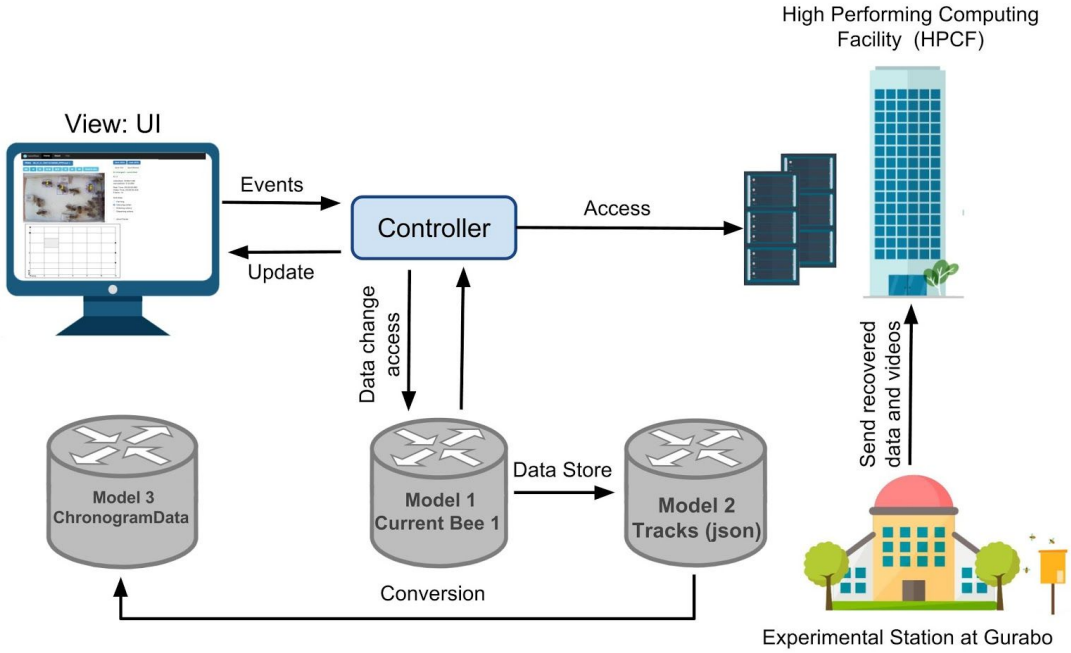

Figure 2: Label Bee Data Flow Architecture

The experimentation with the honey bee hive and data recovery it's first done at the Experimental Station at Gurabo. From there it is stored in our designated servers at the HPCF.

The data is managed through Model-View-Controller design pattern, as seen in Figure 2. The Model encapsulates the data and defines the logic and computation that manipulates the data. Because model objects represent knowledge and expertise related to a specific problem domain, they can be reused in similar problem domains. [https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html] The view is a visual representation of the model [https://blog.codinghorror.com/understanding-model-view-controller/]. The view objects can acquire data from the model and allow the user to edit that data. A controller is the link between the a user and the system. It is through the controller that the View learns about the changes in the Model and vice versa.

In the case for the Label Bee software, the view is all of the software's user interface (UI). The input for the user is drawing a box around a bee on the embedded video. Once the info is submitted, that data is stored at the model to the Tracks data structure and then converted to chronogramData. The one that updates that data is the controller, which handles the input by binding the data in the form by different variables [bee tracker].

2.1.4 Label Bee Data Structure

A. Tracks

Tracks is the main model in the system because it makes all the functionality possible [bee tracker]. Once the user submits its first input, the controller creates the data structure with the annotations that were made. As the user keeps making more annotations on the video, the new data is being pushed to Tracks. This data is being saved in the browser until the user downloads it or the page reloads and it will be refresh into a new one. Tracks feeds the data to the other data structure Chronogram Data which is the data that will be visualized in the chronogram.


```
[{"0":{"ID":"0","time":0,"frame":0,"x":840,"y":224,"cx":962,"cy":352,"width":244,"height":256,"bool_acts":[false,false,false,false]}}]
```


Script 1: Example of Track data structure with one bee annotation

Script 1 shows how the data is saved in Tracks according to its description. To visualize it better, the data structure can be represented in a table.

Frame	Bee ID	Time	X coordinates	Y coordinates	cx coords	cy coords	width	height	Activities

Table 1: Representation of the Tracks Data Structure

All the rows highlighted in green are information to an individual bee. The x and y coordinate represents the x and y for the left up corner of the rectangle. The width and height are also from the rectangle that was made by the user. The element bool_acts represent the booleans for an individual bee activities. This list of booleans will depend on wherever the user marks if the bee is doing a specific activity, as shown in Figure 1.

For now, Tracks is the only data structure that can be downloaded by the user directly from the interface.

B. Chronogram Data

A chronogram is an inscription, sentence, or phrase in which certain letters express a date or epoch. The functionality for the chronogramData data structure is to have the data from Tracks organized and filtered by the needed elements to visualize the bees activities on the chronogram graph. When the Tracks data is created, so is the chronogramData. In order to visualize every new change in the chronogram graph, chronogramData is always been restarted with the new input the user submits.

```
[chronoObservation =
  {Activity:"fanning"
   X:"0"
   Y:"0" }
]
```

Script 2: Example of chronogramData Structure with on bee annotation

ChronogramData is a list of objects called chronoObservation. As seen on Script 2, the data that it recovers from Tracks is the bee's activity, the Bee ID which is the Y coordinate and the frame number which is the X coordinate.

2.2 Technology used

2.2.1 D3

The Data-Driven Documents or D3, is an open source JavaScript library for manipulating documents based on data and create data visualization. It is a fast and flexible API that supports large data sets and dynamic behaviors and interaction. [https://d3js.org/] D3 also provides many built-in reusable functions and function factories, such as graphical primitives for area, line and pie charts. D3 works by allowing the user to bind data to a Document Object Model (DOM) and then apply data-driven transformations to create tables or interactive charts. [bee tracker]

In this project, D3.js is used to build the chronogram display.

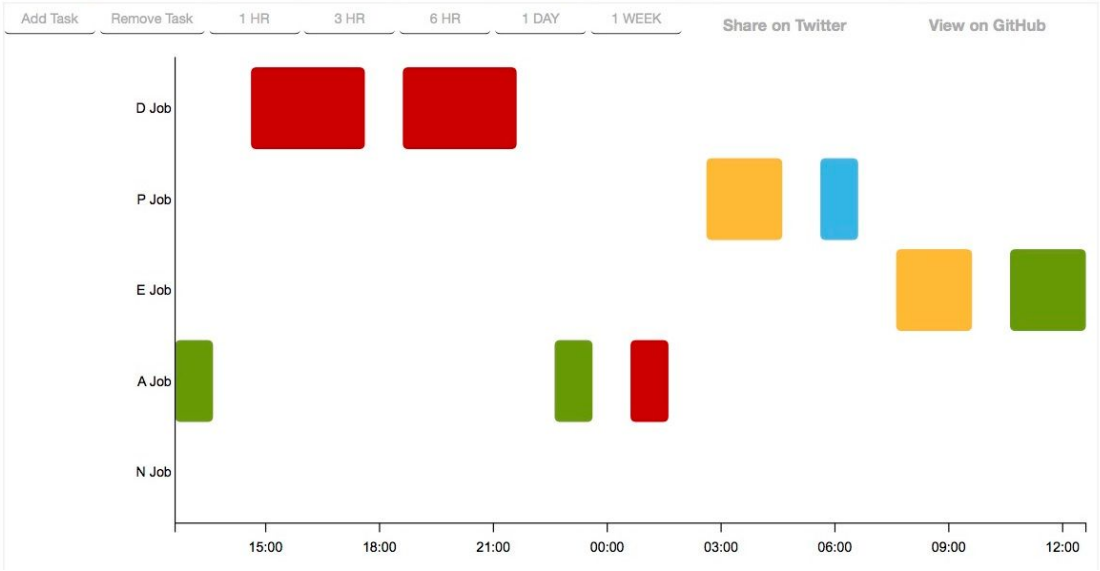

Figure 3: A Gantt Chart made with D3, which was taken as an example to build Label's Bee chronogram data

2.2.2 Json

JSON or JavaScript Object Notation is a lightweight text-based open standard designed for human-readable data interchange.[https://www.tutorialspoint.com/json/] Its is meant to exchange and store data between the browser and the server.

The data structure Tracks is downloaded as a JSON to display the annotations the user made. The user can also upload a JSON file into the interface with the same data structure as Tracks to display already saved data.

3. Methods

3.1 Main D3 objects used in the project

The D3 library allows us to manipulate elements of a web page in the context of a data set. These elements can be HTML, SVG, or Canvas elements. Depending on the type of visualization you want to create to represent your data, you can take advantage of D3 different functions and objects. In our case, the model of our graph is very similar to a scatterplot.

First, a SVG (Scalable Vector Graphic) canvas is created in the html page. This is the maximum dimensions of the height and the width that the chronogram graph is going to have. It is also the designated place on the html page that it's going to appear. Since the main goal for the chronogram data is to visually represent when the identified bees were in the video, the bee ID is going to be the Y coordinate and the frame number is going to be the X coordinate. The Y and X axis for the graph are created by

adding each coordinate variable a scale function. Scales are a convenient abstraction for a fundamental task in visualization: mapping a dimension of abstract data to a visual representation [https://github.com/d3/d3-scale/blob/master/README.md#scaleLinear]. In the case for the X axis (frames), since the data is continuous,a linear scale function is added to the variable. Linear scales are a good default choice for continuous quantitative data because they preserve proportional differences.

```
xScale = d3.scale.linear()
                .range([0, width])
                .domain([0, d3.max(chronogramData, function(d) {
                     return Number(d.x);})
                ]);
```

Script 3: Code to add linear scale function to the X coordinate

Script 3 presents D3 programming format. After adding the linear scale function, the range and domain have to be specified. For the case of x, the range begins in 0 and ends in the width of the graph. For the domain function, it also begins on 0 but since it is dependant on the largest frame number, which is going to be constantly changing with the user's input, the maximum domain is going to have access to the data in chronogramData. *D3.max* returns the maximum number in a given array, which in our case are the x values.

At the Y axis, we only want to display the dee ID's that the user has specifically annotated, and not the numbers in between. In this case, since it's categorical data, the scale that will be added to the Y axis is ordinal scale, which specifies an explicit mapping from a set of data values to a corresponding set of visual attributes [d3 wiki].

```
yScale = d3.scale.ordinal()
           .domain(chronogramData, function(d) {
                   return Number(d.y);})
           .rangeRoundBands([0, height - margin.top - margin.bottom],
0.1);
```

Script 4: Code to create the Y Ordinal scale.

As seen in Script 4, the domain function is similar to the Y scale domain. The function *.rangeRoundBands()* sets the scale's range to the specified array of values while also setting the scale's interpolator to interpolateRound, which returns an interpolator between the two numbers a and b and rounds the resulting value to the nearest integer [d3 wiki].

Once the scale variables have been made, it's pretty straightforward to create the axis in D3. The x and y axis variables are equal to the *d3.svg* elements, that also have the *d3.axis()* and *d3.scale()* function. The svg variable is then created and by writing svg = d3.select("#svgVisualize"), we specified to D3 where we want our svg elements to display in the html page. *D3.select* selects the first element that matches the specified selector string [d3 wiki]. In this case, we pass the string "#svgVisualize" as the parameter, which in the html script, it's an id. The width and height graph variables are added to the svg variable with the *d3.attr* function.

For organizational purposes, we create another svg element called chart, and append it to the already created svg element. This will help make the display of the graph more manageable. The y and x axis elements are appended to the chart element. The circle and rectangles shapes that represent the identified bees appearances are also appended to the chart element.

```
  <svg id="svgVisualize" >
     <g class="display">
          <g class="x axis"> </g>
          <g class="y axis"> </g>
          <circle>
          <rectangle>
     </g>
</svg>
```

Script 5: HTML results of the svg elements created with D3 functions.

D3 functions converts the code into svg and css elements to display it in the View. The first element svg id is the canvas for the graph. The next element with the class display is the chart element that was append to the svg canvas.

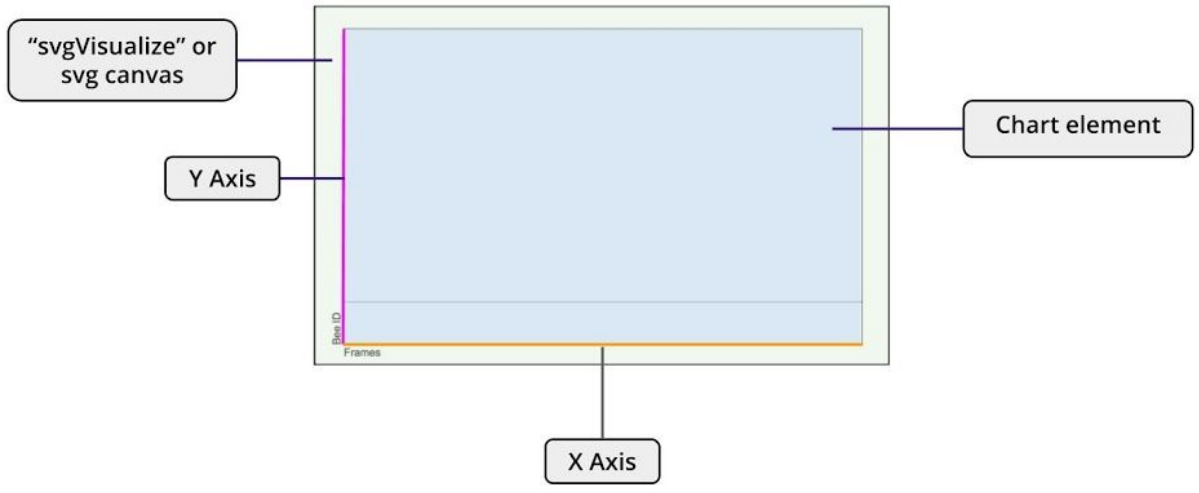

Figure 4: Svg elements for the chronogram

By appending the different svg elements, they become a sub-element of the svg canvas.

Reference:
beeHive:happymeluv, https://www.vecteezy.com/vector-art/89934-spring-bee-vector-illustration