**Project Cecropia**

Training Manual for the Integrated Server for the SlothBot

**Yilun "Allen" Chen**

Robotarium
Georgia Institute of Technology
United States
June 30th, 2019

# Contents

# 1 Overview

## 1.1 Abstract

Project Cecropia aims to provide real-life application for the SlothBot swarm. The SlothBots are fall-safe wire-traversing robots developed in Robotarium at Georgia Tech. They will later be deployed to Atlanta Botanical Garden for enviornment monitoring purposes. Project Cecropia offers a solution to allow the collection of the data trasmitted from the SlothBots, along with the capability to store and visualize the data on a web page that it serves.

This document aims to provide an overview of how the project works for future developers of the project. Details of how the project is structured, how it achieves different functionalities, how to deal with emergencies/bugs will be provided.

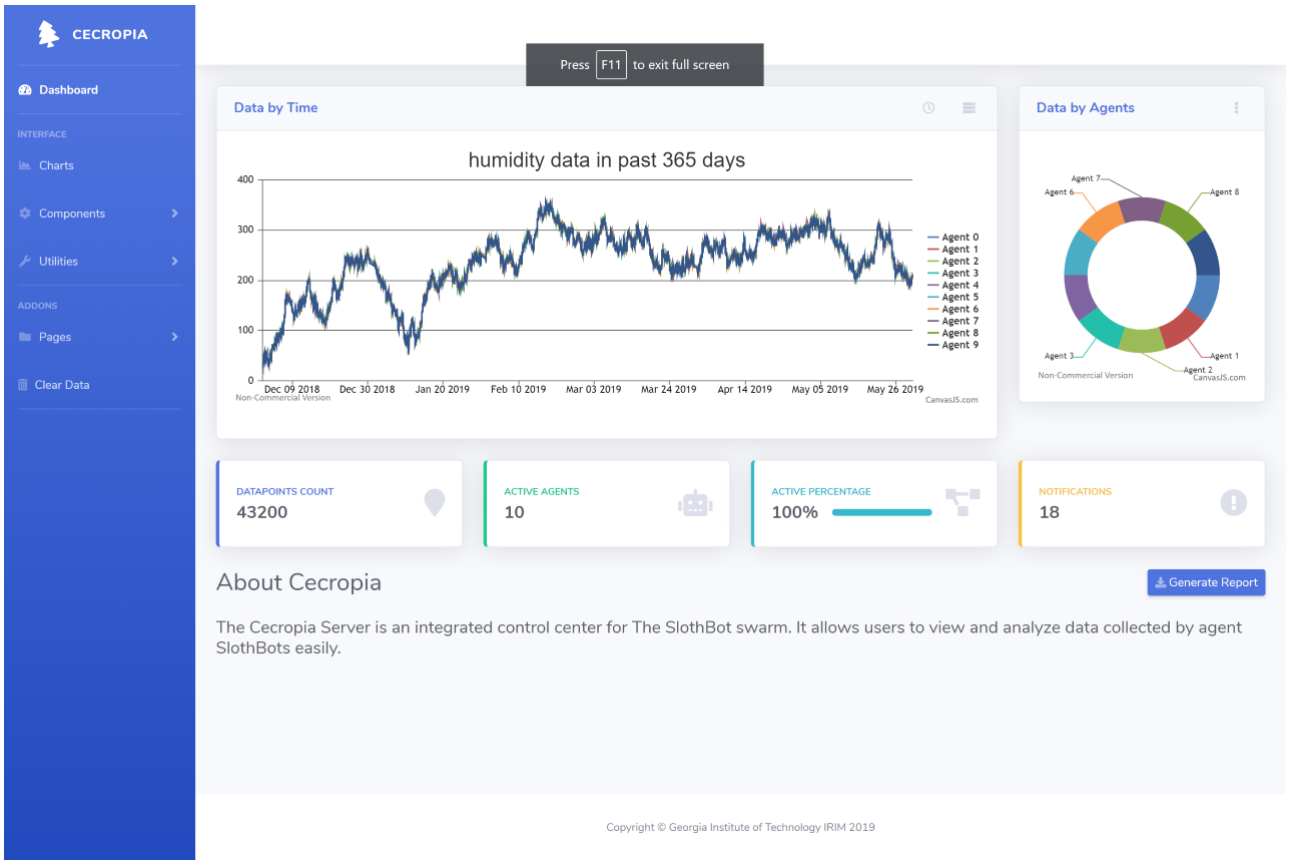


Figure 1. GUI screenshot of the Cecropia system.

## 1.2 Tech Stack

Here is a general overview of how the project is structured. Main building blocks of the project and their rationales are provided, so that future developers of the project may use this as a reference.

**Web Framework:** Express & Node.js
Node.js as a web framework has high integrity when it comes to building web applications. Its package manager NPM offers a wide variety of packages, providing almost all functionalities desired.

**Database:** MongoDB
MongoDB uses JSON-like syntax as the form of its document. This feature/characteristic means that its documents can be easily translated into a form that is easy for JavaScript to use.

**User Interface:** JQuery & Bootstrap
jQuery and Bootstrap offer a wide choice of templates that we can straight up use and build upon, while adding a lot of additional handy functionalities that can be used in overall development process.

**Package Management:** NPM
Almost all JavaScript packages can be found on NPM. It is extremely easy to use and it also offers great mobility - When you try to move the server to another host, you can easily setup the needed environment.

**Data Visualization:** CanvasJS (JavaScript Library)
Canvas JS is a licensed JavaScript library that allows you to create graphs with massive amount of data points. It has superb performance in handling mass data and is highly configurable.

**Server/Data Visualization API:** DataParser (Customized JavaScript function)
A customized JavaScript class that takes in all the data points retrieved, sort them by timestamp and dates and outputs a CanvasJS-friendly cluster of data to be put directly into the data-visualization process.

**Testing Platform:** RDP.py (Customized Python Script)
A script to generate continuous mock data and send it over to the server. Depends on the pacakge 'request'.

# 2 Design Roadmap

## 2.1 Design Constraints

### 2.1.1 Power Consumption:

Due to the moving nature of the SlothBots, they will not be relying on stationary charging stations, but instead, their only source of power is the solar panels they carry with them. This means that we want to minimize the power consumption of the SlothBots so that they run longer before they take another 'power nap'.

### 2.1.2 Mass Data Handling:

Because the number of robots deployed is unknown, and that we are collecting a huge amount of various kinds of data, we must ensure that the server is capable of retrieving only the necessary data, and that it has the capability of properly handle these data.

### 2.1.3 Connection-Failure-Safe:

In case that the server goes offline, the data already collected should be preserved.

### 2.1.4 Centralized Data Processing:

An entry point to all the data collected is required. This means that all the data collected by the agent robots must be re-collected by one machine and can therefore be processed.

## 2.2 Current Solutions

### 2.2.1 Stationary Server

As of for now the server is deployed onto a stationary computer that runs continuously under a stable network with internet access. This provides a resolution to seemingly conflicting design constraints by putting the load of performance-heavy works onto a device that has access to a steady source of power. Agent Robots can now only be responsible for transferring the data, without wasting the power to process them.

### 2.2.2 Integrated Database

The server will be interacting with an independent database that runs alongside it. This means that even if the server encounters an error, the database will still be preserved. Also, because MongoDB is a NoSQL database, even if the server is injected with false/garbage data, the database will still be able to function properly.

## 2.3 Expandabilities

### 2.3.1 Network Connection Check on the SlothBots When Sending Data

Currently the programs that run on the SlothBots are still work-in-process. Ideally they would automatically detect the quality of their internet connections and in case they are unable to send over the data, they would save the data in a cache, and once they regain proper connection, send all the data over to the server since the connection breaks down. The server is already capable of dealing with multiple entries per request and it is also capable of sorting the data points by their time stamps.

### 2.3.2 Database With Better Performance

MongoDB is not the best performance-wise and space-wise when it comes to massive data (More than hundreds of data entries, for example). If we were to deal with this amount of data entries, it is a good idea to switch over to SQL databases like MySQL. This, however, requires the developers to develop a query system to let the server and the database communicate properly.

# 3 Building Blocks

This section aims to introduce the readers what each individual module in the project does, so that in case of errors, the developers can quickly locate the problem and solve it. Only the modules used in real deployment will be discussed.

## 3.1 app.js

- Modules Management: It loads in all the required modules.
- Establish Connection: It setup express and starts to serve on desired IP.
- Database Connection: It attempts to connect to MongoDB via port 27017.
- It routes different request to different sites.
  - /visitor: Dashboard of the system, where users can obtain data.
  - /dataPost: Interface between robots and the system. Used to post data.
  - /clearData: Used to clean up all data stored in the database.

## 3.2 dashboard.html

- It is a html file that will first be processed on the server side (replace data place holder by real data) then sent over to the client's side to be rendered.

## 3.3 dataParser.js

- It takes in all the data entries and put data points of individual agent into its own bucket.
- It sorts the data points by their timestamps.
- It generates strings to be replaced in dashboard.html file that is readable by CanvasJS.js to visualize.

## 3.4 Miscellaneous

- canvasjs.min.js: Should be left untouched.
- All files under node_modules: Dependencies for the project to function properly. Should be left untoucched.
- All files under vendor: GUI components of the project. Should be left untouched.
- sb-admin-2.min.css: CSS file for the dashboard. Should be left untouched.

# 4 Troubleshooting

## 4.1 Address Not Available

This corresponds to the following lines in app.js. Check if the your IP address aligns with the setting.

```
const hostname = '192.168.137.1';
const app = express();
const port = 80;
```

## 4.2 connect ECONNREFUSED

This means that your mongoDB is not set to connect on the right port. The default port is 27017.

## 4.3 Data Visualization Failure

Many errors can lead to failure to visualize data. Try to use RDP.py (Random Data Poster) as a testing tool. If the server logs "new data received" in console then it means that the connection with the database is intact. If no data is received, check the connection with the database, or the internet connections of your devices.

Assuming that your connection is intact: At this point, check if the data from RDP.py can be displayed, then check the formatting of your post requests. However if it not being able to display RDP data, it means that dataParser is malfunctioning and a debugging session with DataParser.js is highly recommended.

# 5 Data Formatting Examples

## 5.1 Post Requests

Post Requests of the agent robots should follow this format:

```
{"dataType":"temperature","timeStamp":2019011020,"dataValue":255,"signature":1}
```

## 5.2 Document Entry in MongoDB

A document (or data entry) in MongoDB has the following format:

```
{"_id":"5cfe784f903ad52ee0cad536","dataType":"temperature",
 "timeStamp":2019011020,"dataValue":255,"signature":1}
```

## 5.3 String Processed by DataParser

The data after being processed by DataParser.js should have the following format:

```
{
"type":"line",
"showInLegend":true,
"legendText":"Agent 0",
"xValueType":"dateTime",
"dataPoints":[
  {"x":1558360800000,"y":86},
  {"x":1558368000000,"y":91},
  {"x":1558375200000,"y":97},
  {"x":1558382400000,"y":97},
  {"x":1558389600000,"y":99},
  {"x":1558396800000,"y":93},
  {"x":1558404000000,"y":92},
  {"x":1559253600000,"y":47}
  ]
}
```