

To: Professor Osama Alshaykh and Professor Alan Pisano

From: Caroline Jones, Sneha Pradhan, Michael Haley, Tanatsigwa Hungwe, and Zehua (Harry) Zhao

Team: 1

Date: April 30th, 2017

Subject: **Software Readme**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**



1. **Overall Description**

1.1 The Luminesense system consists of five major modules: the wearable transceiver, the central base, the luminaires, the single pixel cameras, and the web application. Their integration amongst each other and utility depends on the current mode of engagement - “gesture mode” or “adaptive mode”. The following README concerns the software components only. The source code for the software is attached in the Resource GitHub folder.

**2.0 Raspberry Pi 3**

2.1 *receive\_arduino*.js

This controls the Central Base - responsible for the system’s data aggregation. The Raspberry Pi runs this JavaScript program to scan and pair with the transceiver via its BLE peripheral address (in gesture mode). After doing so, the base receives all the ID, button, and IMU data sent from the transceiver. Subsequently, the base applies the data and implements the desired commands sent from the user. These commands are relayed to the luminaire(s) and logged into the web application’s database via Wi-Fi. In adaptive mode, the base does not receive any data from the wearable. Instead the contextual data is received from the single pixel cameras. This data informs the base whether a user is present in the room or not. Depending on the information, the base relays its decision to the luminaire(s) (turn “on” or “off”) and database. You must install *noble*, *particle-api-js*, and *pg* via the sudo apt get method. These are responsible for the Bluetooth Low Energy, Particle photon calls, and database querying and posting methods respectively. These are used by the RasPi to aggregate data and issue commands.

2.2 MQTT\_adapt.py

This process establishes a connection with the MQTT broker on the host. It receives input from the raspberry pi sensor reader (the raw data stream from the single pixel sensors) and launches adaptive\_algo.py to check for occupancy. The data stream is then formatted in MH\_data.txt to produce a version the algorithm will constantly check. The process also connects to the central database, sets up the cursor, and initializes the luminaire state to OFF. Depending on the result of each adaptive\_ago.py run, it will update the database to indicate the ON/OFF state using SQL statements (e.g. DELETE, INSERT INTO). Variables COUNTER and STATE keep track of the high level luminaire status. This information will be printed in the console after every detection period.

2.3 adaptive\_algo.py

This launches N subprocesses (where N = number of single pixel sensors) which all run the same function with the unique sensor ID to determine if the corresponding sensor detected movement during the sampling period. This function accesses the latest data stream values in MH\_data.txt and calculates the sensor’s running average of the RGB data in the “accum” variable. These processes keep track of recent accum values that have flagged a possible new entrant. If the average increases by a certain amount for consecutive time intervals, a sensor has detected an occupant. At this point the subprocess exit and sends the status indicating this occupant through a pipe to main(). After all sub processes have completed, adaptive\_algo write the number of sensors that detected an entrant to stdout which can be read by MQTT\_adapt. Install the psycopg2 package in order to write to the PostgreSQL database.

**2.0 Arduino 101**

2.1 wearable\_code.ino

The Arduino contains a C language program required to utilize the additional peripherals. The program samples light signals, performs Fast Fourier transforms (FFT), samples IMU signals, polls button states, and sends data (of size 4 bytes) via BLE. The Arduino contains 196 kB Flash memory and 24 kB SRAM - sufficient to execute computationally complex instructions in reasonable time. In adaptive mode, the wearable is not used to interact with the system at all. In order to run the specific functions such as BLE, FFT an IMU functionality, certain libraries must be utilized: fft\_2.h, CurieIMU.h, CurieBLE.h, and math.h. All code is compiled using the Arduino IDE. This IDE application can be found at <https://www.arduino.cc/>. Additionally, the Arduino 101 board and appropriate USB port must be selected to sketch the code onto the board.

**3.0 Photon**

3.1 photon\_code.ino

The luminaire is an electric light unit consisting of a Cree MC-E LED, and a Particle Photon. The LED is capable of producing multi-colored lights (from the visible spectrum) such as red, green blue light. The Photon is a microprocessor that executes instructions sent from the central base. The Photon contains a Cypress BCM43362 Wi-Fi chip, 1MB Flash, and 128KB RAM - sufficient to run C++ programs to control the LED states. The commands are sent to the Photon by the central base via calls to Particle’s JavaScript API (in *receive\_arduino.js*) . The commands are interpreted by the Photon’s program and the user’s desired command is implemented.The Photons are identified by either name, Device ID or frequency - with reference to a luminaire map. Each Photon runs the same C++ program (with the exception of the LED frequency modulation setting, called *freq*). Photons require initialization on the command line. This requires Node.js version of at least 6.0.0. Once installed, you can initialize Photons to desired networks and create an account on the web IDE at **build.particle.io/build**. Here is where all the code development and compilation is done on the Photons with the simple click of a button.

**4.0 Web Application**

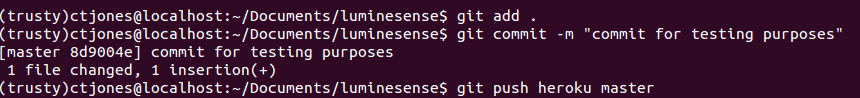
4.1 index.ejs

The web application was based on a Bootstrap Template (*bootstrap.css*). We changed the front-end to make it look more sleek as well as simple. The website application that accompanies the luminesense system is currently live at [luminesense.herokuapp.com](http://luminesense.herokuapp.com/). This serves as a portal for the user to interact with their system. The application serves two purposes: energy usage reporting to users and remote control over the system (e.g. customize the gesture library). The energy usage graph displays the energy usage of the system over time, with kilowatt-hours on the vertical axis (y-axis) and time on the horizontal axis (x-axis). Using this, users can track their energy usage over time.

4.2 gestures.ejs

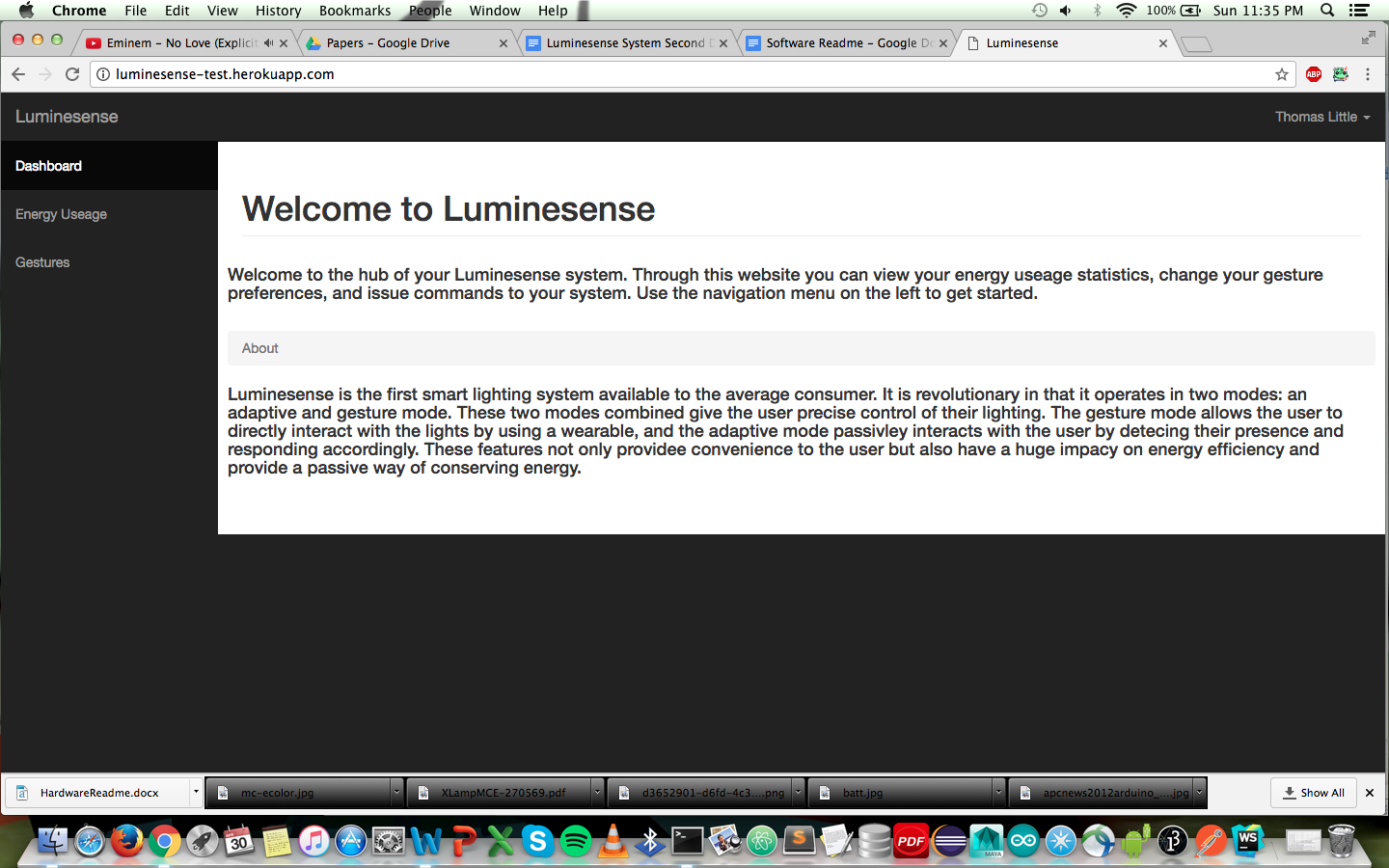
Users can also issue commands to luminaires on the system through the web application, and change their gesture preferences for commands. This allows the user to interact with the luminaires if their wearable is not operable for some reason. Through the testing of the web app, we show how the user can interact with and customize their system. The test also shows that the user is able to gather energy information about their system, and control their system in the event that they do not have access to their wearable device.

4.3 Web app deployment



*Figure 1: Shell commands*

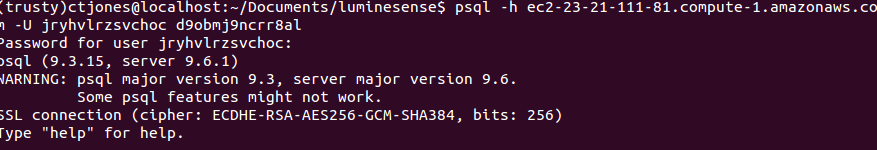
This pushes the code to the Heroku repository, and prompts the update of the web application. Then, by navigating to the url of the web application at [https://luminesense.herokuapp.com/](https://luminesense-test.herokuapp.com/) we see the home page (*index.ejs*) of the application, shown below:



*Figure 2: Home page*

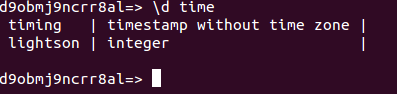
To further show the Heroku cloud and PostgreSQL integration, we can directly connect to the

PostgreSQL database using the PostgreSQL command line interface, shown below:



*Figure 3: PostgreSQL CLI*

From the PostgreSQL command line interface, it is also possible inspect tables. Shown below is the inspecting of the time table, which contains the timestamp and integer columns:

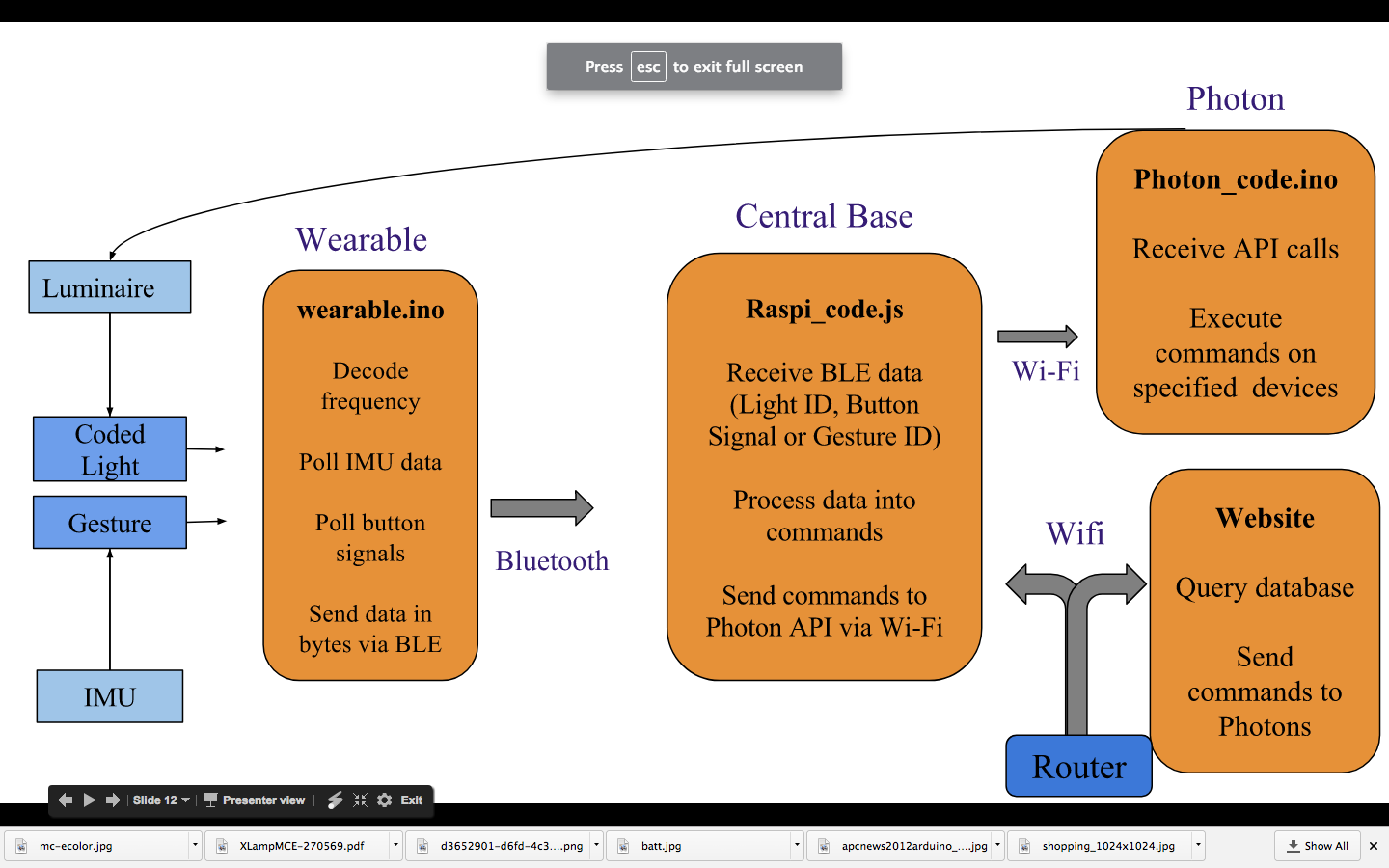


*Figure 4: PostgreSQL table column viewing*

**4.0 Software Dependency**

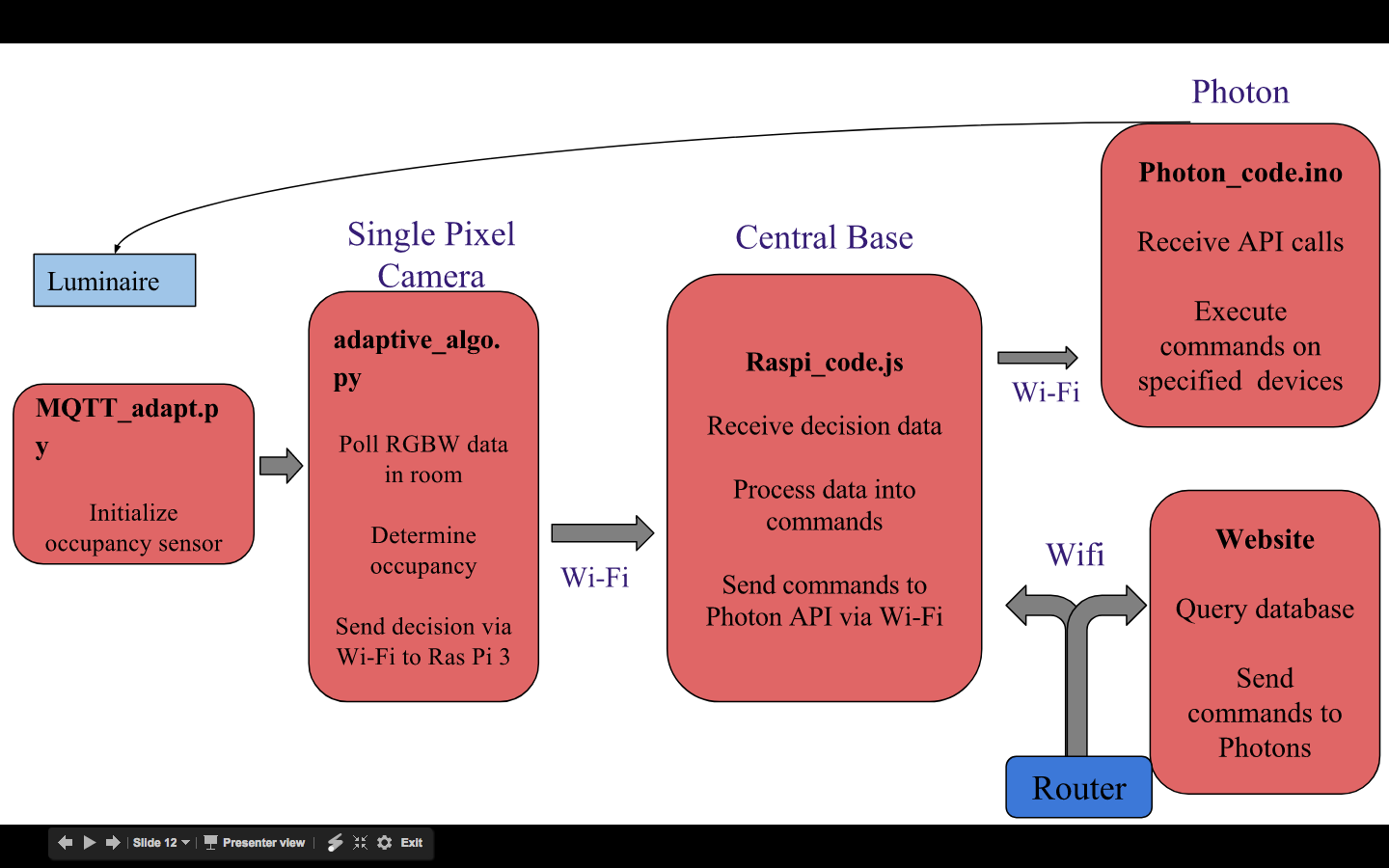
The software used by the system interacts in numerous ways depending on the scenario i.e. gesture or adaptive modes.

Gesture Mode

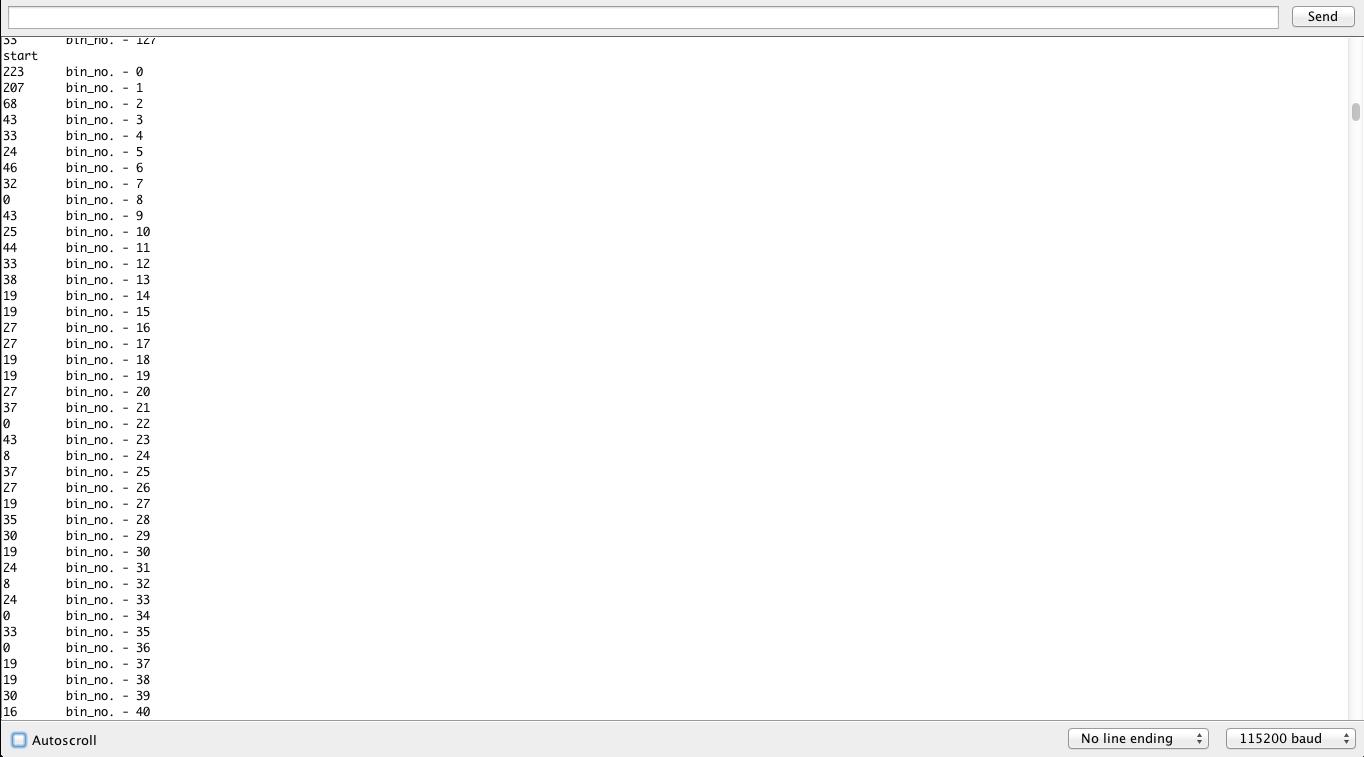


*Figure 5: Gesture Mode software program dependency (L-R)*

Adaptive Mode



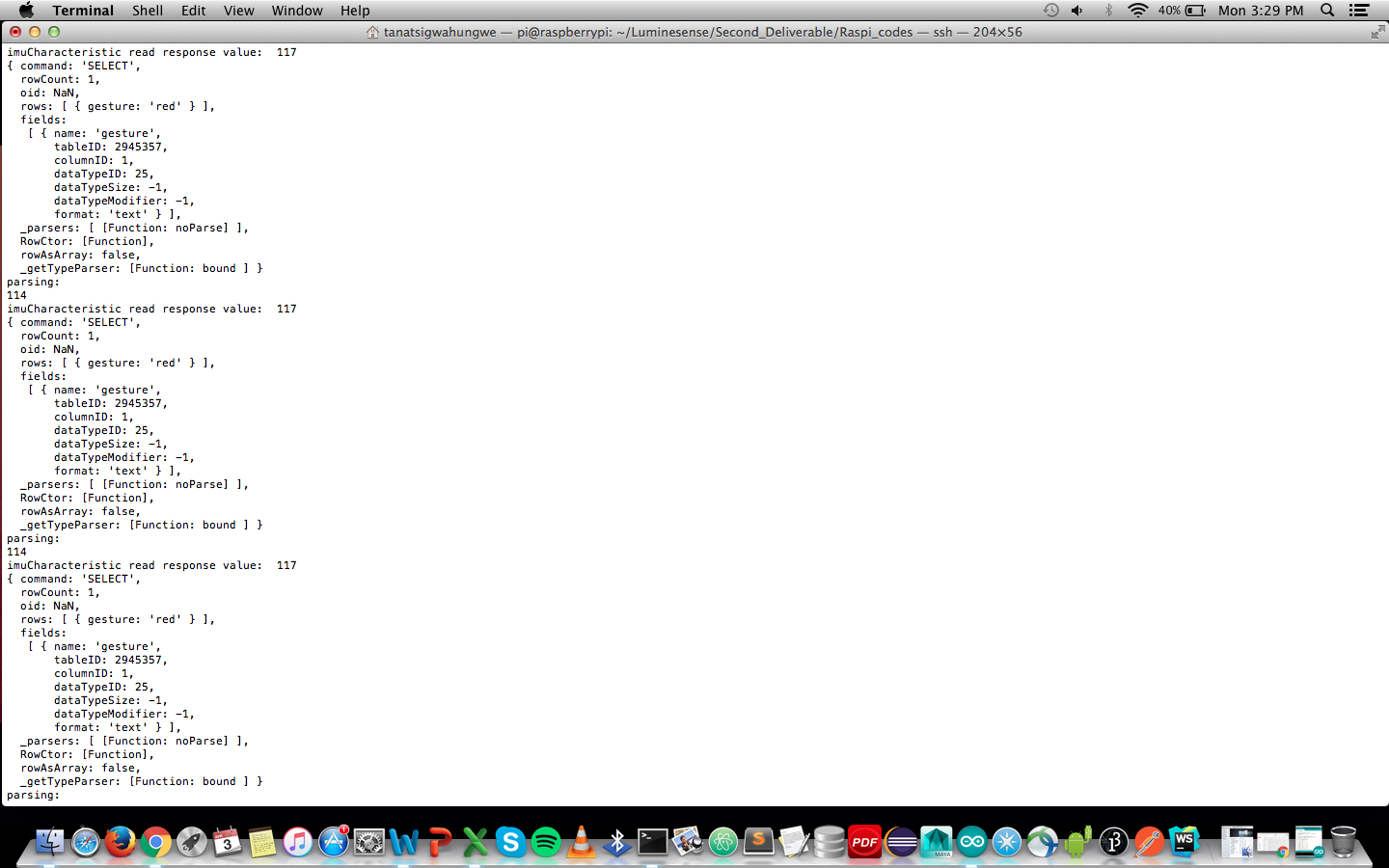
*Figure 6: Adaptive Mode software program dependency (L-R)*



*Figure 7: FFT results from early version of wearable\_code.ino*



*Figure 8: receive\_arduino.js code luminaire selection procedure*



*Figure 9: Gesture command data from receive\_arduino.js*