

To: Professor Osama Alshaykh and Professor Alan Pisano

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

Team: 1

Date: 11/20/16

Subject: **Luminesense System First Deliverable Test Report**

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

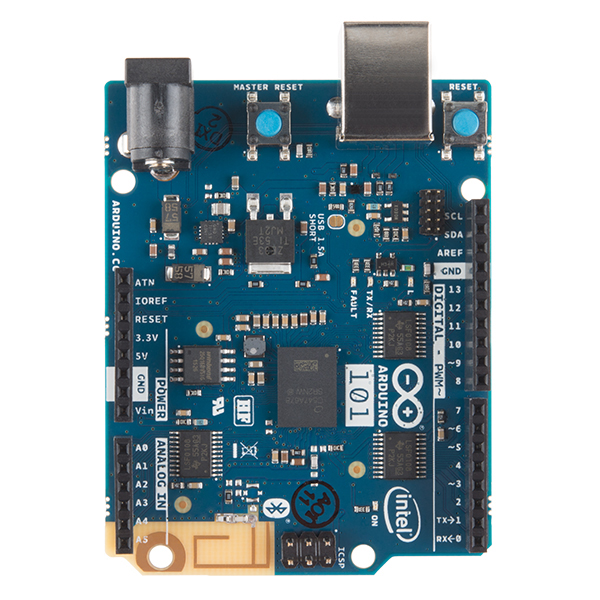
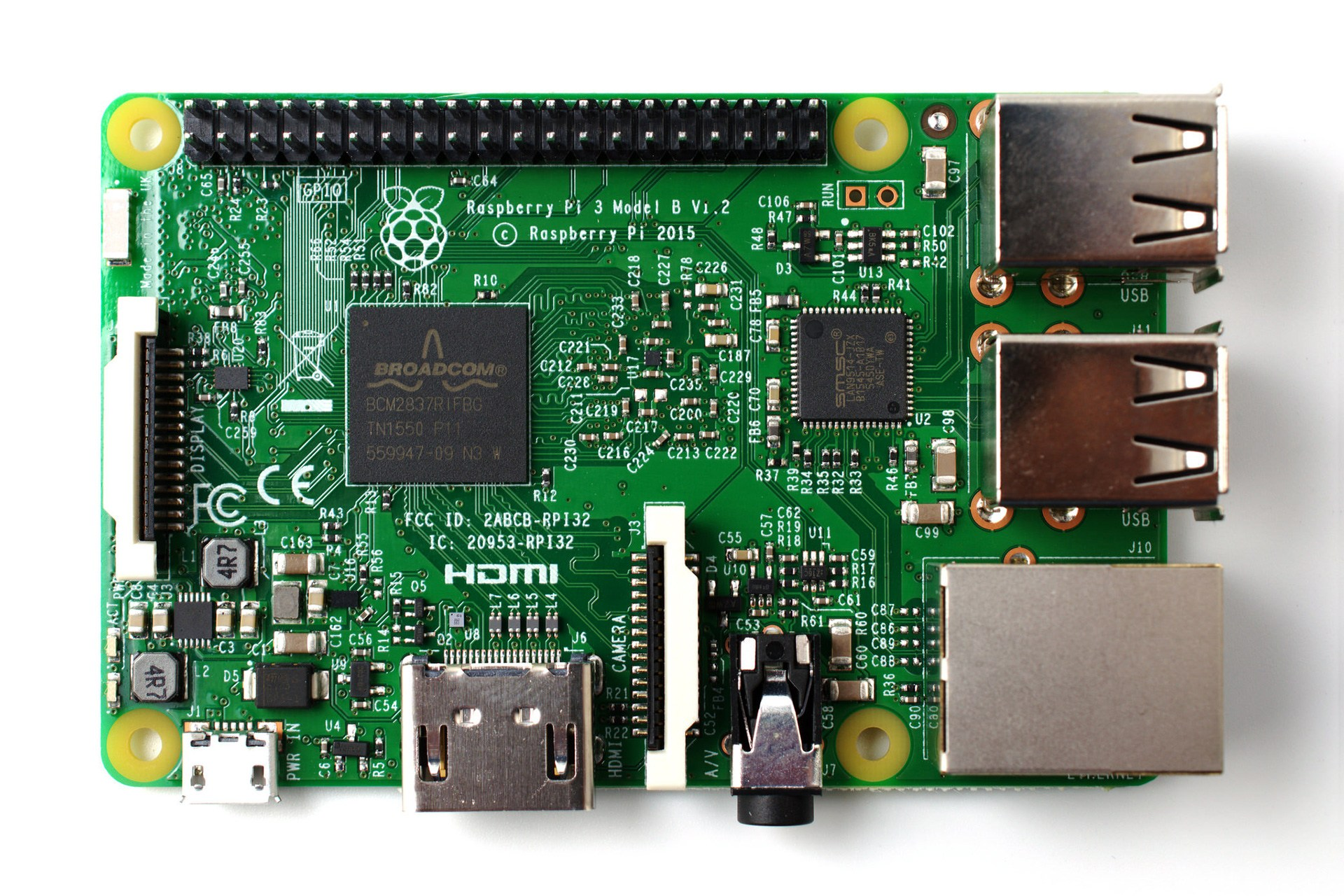
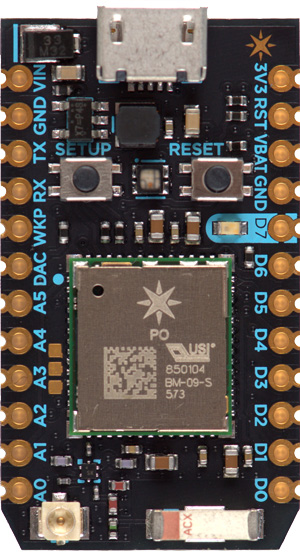


1. **Project Objective**

1.1 Team Luminesense seeks to revolutionize the way that people interact with lighting fixtures in rooms. The project’s primary motivation is to provide an intelligent, gesture based lighting system that provides customizable, convenient, and energy efficient control over these fixtures.

**2.0 Test Objective and Significance**

2.1 As illustrated below in Figure 1, the test procedure focused on communicating amongst various components of the Luminesense system. First, the luminaire sends a light encoded identification number to the Arduino. Second, the Arduino collects positional data from the its onboard accelerometer and gyroscope. The Arduino then transmits both datasets via Bluetooth Low Energy(BLE) to the Raspberry Pi 3. Third, the Raspberry Pi processes the data into a command and relays this command to the Particle Photon via Wi-Fi. Lastly, the Photon changes the state of the luminaire. This arrangement proves the validity of the communication loop as well as gesture recognition concepts and successful interaction with the client-provided luminaire device that will be adopted in future iterations of the system design.



*Figure 1: The Communication Loop*

2.2 The crux of the Luminesense system design is the innovation behind the “Loop” i.e. the paths of communication amongst the components of the system. Therefore, it was essential to observe, assess and validate this key feature before engaging in further system development. Although the system’s design will increase in complexity, the underlying logic of the final system will reflect the logic investigated in the first deliverable tests.

2.3 **Arduino 101**

The Arduino Curie or Arduino 101 is retrofitted with the Intel Quark chip, a 6-axis accelerometer and gyroscope, and Bluetooth Low Energy. This arrangement appropriately models the wearable device: simultaneously receiving light data from a single Luminaire and sending the IMU and light ID data to the Raspberry Pi via Bluetooth Low Energy. Testing this component was necessary to assure that a user could accurately send light and gesture data to the Raspberry Pi (via Bluetooth Low Energy).

2.4 **Raspberry Pi 3**

The Raspberry Pi is responsible for receiving relevant Arduino data, processing it, and sending resultant data to the Particle Photon. Testing this component was vital to ensure that the Raspberry Pi could precisely receive light ID and gesture data from the Arduino (via Bluetooth Low Energy), process the data and send an appropriate command to the Photon (via Wi-Fi).

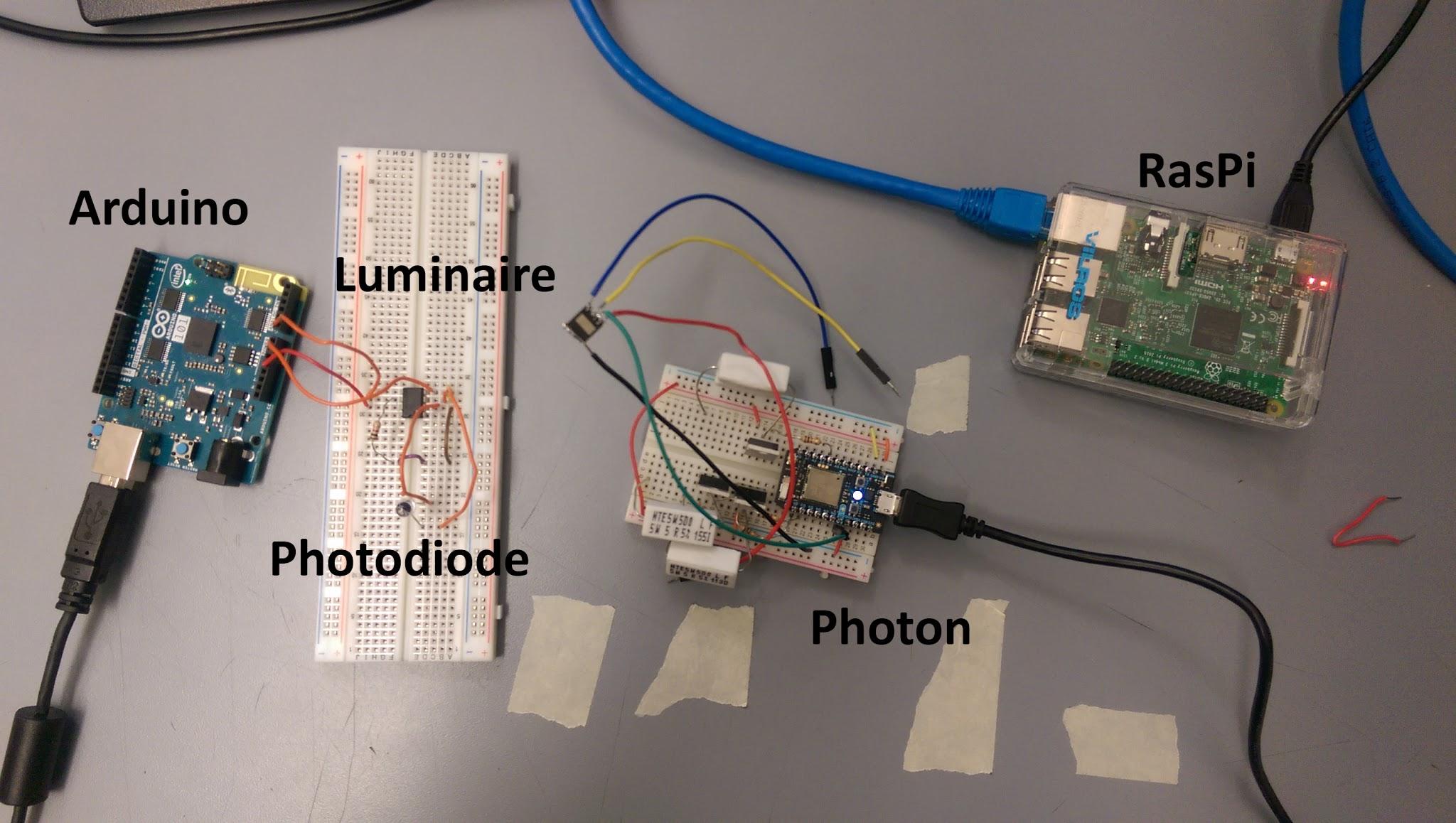
2.5 **Particle Photon**

The Photon is responsible for receiving and processing relevant Raspberry Pi commands. Testing this component was crucial to confirm that the Photon could clearly receive command data from the Raspberry Pi (via Wi-Fi), process the data and cause an appropriate change to the state of the Luminaire (in this case, to turn it on or off).

**3.0 Equipment and Setup**

3.1 **Overall Arrangement**

The entire arrangement of components as outlined and described in Figure 1 remained fixed throughout the testing procedure as shown in Figure 2 below.



*Figure 2: The implemented Communication Loop for testing*

3.2 **Arduino 101**

To test the functionality of the Arduino, the *test1\_receive.ino* code is run. The program reads PWM light signals from the Luminaire. The analog signals were sensed by a photodiode connected to the Arduino’s A0 port. We perform a division operation on the analog readings to obtain the specific ID of the light (values range from 0 - 255). The Arduino’s positional data was collected using the onboard IMU. This data was then processed using Madgwick’s filter algorithm. The algorithm sets the IMU sample rate to 25 Hz, collects raw accelerometer and gyroscope readings in the x, y, and z directions and finally converts the raw data to angular velocity and acceleration values. This data helps determine the board’s orientation. For this assignment we separated the orientation in 4 different possibilities - connector up, connector down, analog pins up and digital pins up. As we were only testing for on and off, we decided to only use the connector up and connector down orientations. The light ID and the orientation data was then sent to the Raspberry Pi via Bluetooth Low Energy, dependent on the status of the state machine (for example, there are is no need to send two “on” commands in a row to the same light). The validity of the orientation data, and hence, the Arduino’s functionality is determined by comparing the data received by the Raspberry Pi and the data printed on the Serial monitor.

3.3 **Raspberry Pi 3**

The functionality of the Raspberry Pi is proven by receiving light ID and orientation data from the Arduino via Bluetooth Low Energy. To connect to the Arduino via Bluetooth and receive the appropriate data, we used a javascript library called noble. To connect to the particle cloud, we used the particle js api. In the Raspberry Pi, we do all our coding in Javascript. The data received is then translated into a command consisting of two arguments: light ID (the id of the Luminaire) and gesture (“u” - up or “d” - down). The Raspberry Pi identifies these arguments as characteristics with the help of UUIDs, which is typical for Bluetooth Low Energy communications. Each characteristic has an UUID. The Arduino will update the bluetooth characteristics as it receives input, which the Raspberry Pi monitors for changes. These “changes” are mapped into a translated command. The Raspberry Pi’s Terminal program is launched, where the command is passed as the arguments of a JavaScript function call to the Photon (the call is done via Wi-fi). The legitimacy of the translated command, and hence, the Raspberry Pi’s functionality is determined by comparing the data received by the Raspberry Pi and the resultant translated data printed on the Terminal window.

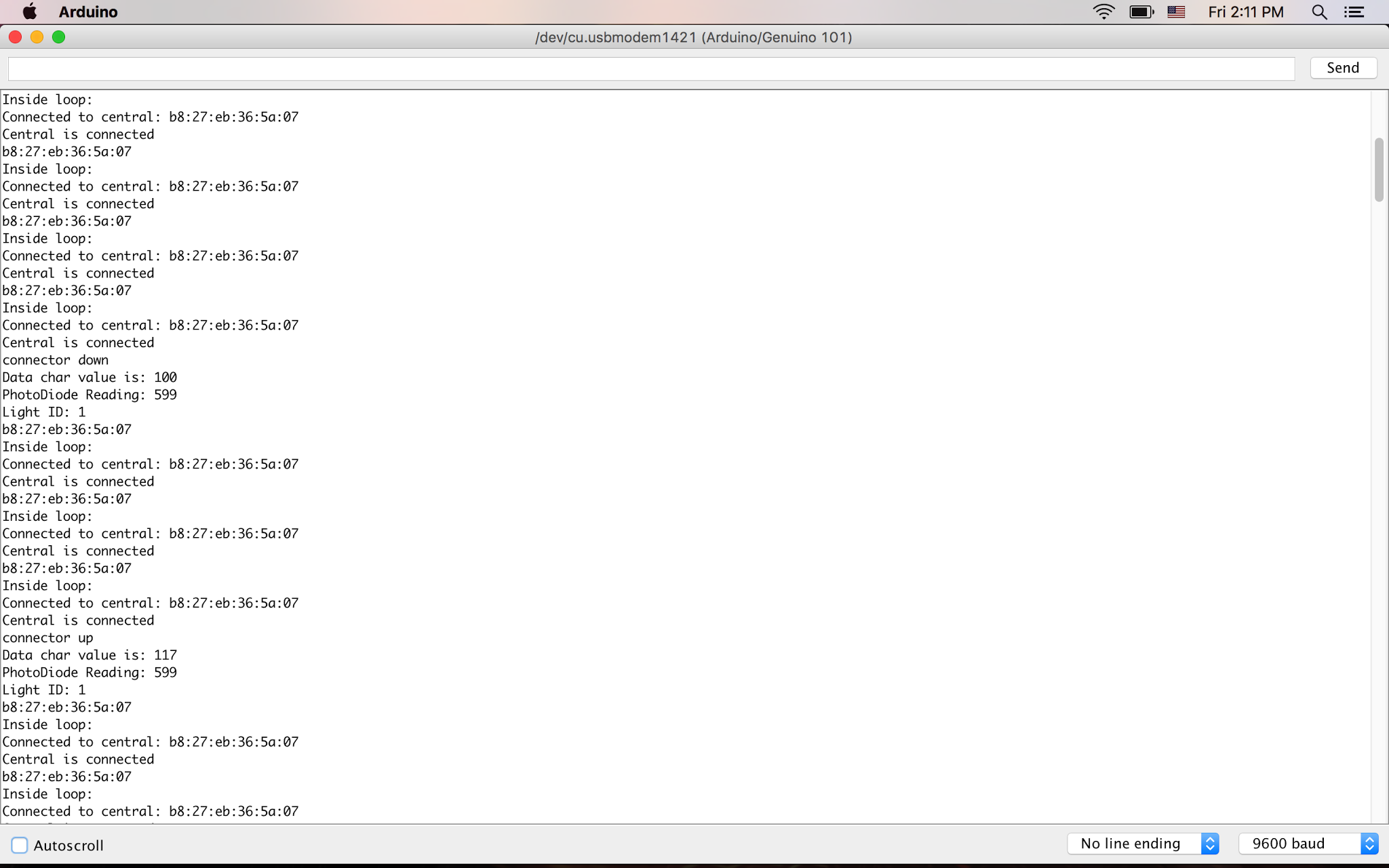
3.4 **Particle Photon**

The Photon’s functionality is determined by receiving a command from the Raspberry Pi via a Particle API JavaScript call to one of its functions (via Wi-fi). The luminaire is connected to the D7 port of the Photon. The Photon contains a function, *togglelights* which is responsible for changing the state of the Luminaire based on the arguments (light ID and gesture). The command sent from the Raspberry Pi denotes these arguments. If the gesture is “u” - up, the Luminaire turns on. If the gesture is “d” - down, the luminaire turns off. The light ID determines which luminaire is affected by the gesture. The validity of the Photon’s functionality and the commands is tested by observing the state of the luminaire (“on” or “off”) and the CLI (Command Line Interface) call outputs printed on the Terminal window.

**4.0 Measurements and Data**

4.1 **Arduino 101**

During the testing of the Arduino, the light signal data was sampled using a photodiode, providing a value in the range 0 - 1023. A division operation was applied to the data to calculate the a digitized value - the light ID. The positional data (x, y, z coordinate values) of the Arduino was sampled by the IMU whilst Madgwick’s algorithm calculated the orientation of the board (facing “up” or facing “down”). The data collected by the algorithm was printed to the Serial monitor. Figure 3 below shows the data observed via the Arduino’s Serial monitor.



*Figure 3: The Serial Monitor outputting light and gesture data*

4.2 **Raspberry Pi**

During the testing of the Raspberry Pi, light ID and orientation data was sent from the Arduino via Bluetooth to the Raspberry Pi. The Raspberry Pi identifies these arguments as characteristics with the help of UUIDs. Each characteristic has an UUID. The Arduino will update the bluetooth characteristics as it receives input, which the Raspberry Pi monitors for changes. These “changes” are mapped into a translated command which is sent to the Photon on a byte-by-byte basis. To save time and resources during Bluetooth transmission sessions, a single character was used to identify the gesture (ASCII value equivalent: u = 117; d = 100) . The translated command data was printed to the Terminal window as shown in Figure 3 above as *Data char value*.

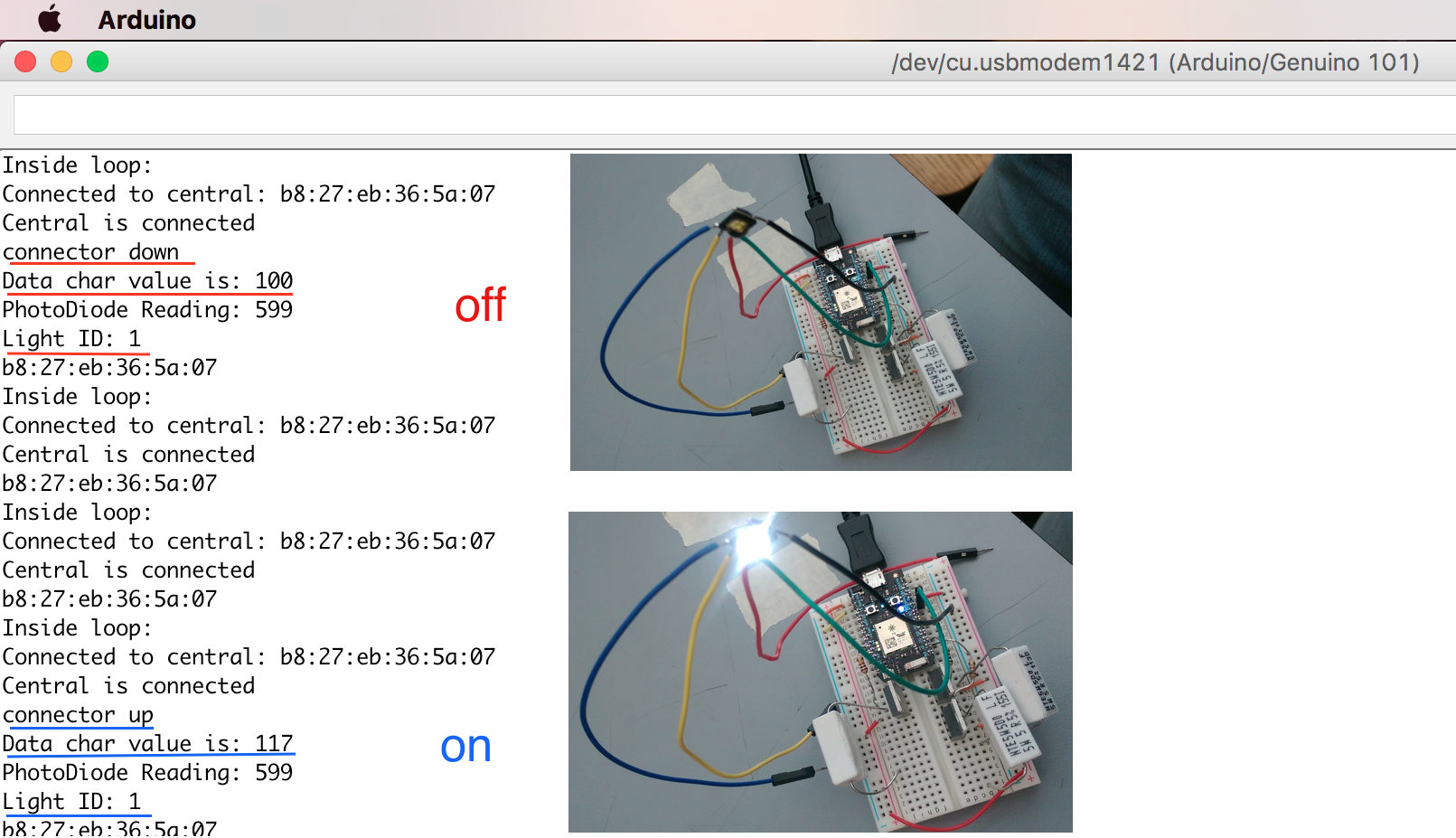
4.3 **Particle Photon**

During the testing of the Photon, command data was sent from the Raspberry Pi via a CLI call to the Photon’s function (over Wi-fi). The Photon’s code *togglelights.ino* contains a function which accepts these arguments and changes Luminaire states based on the input commands.The luminaire states were identified by the return value of the call to the togglelights function. Return value of 1 - Luminaire is on. Return value of 0 - Luminaire is off. Return value of -1 - Luminaire state does not change from the previous state. Luminaire state and input command data was collected and outputted to the Terminal window as shown below in Figure 5.

**5.0 Conclusions**

5.1 **Arduino 101**

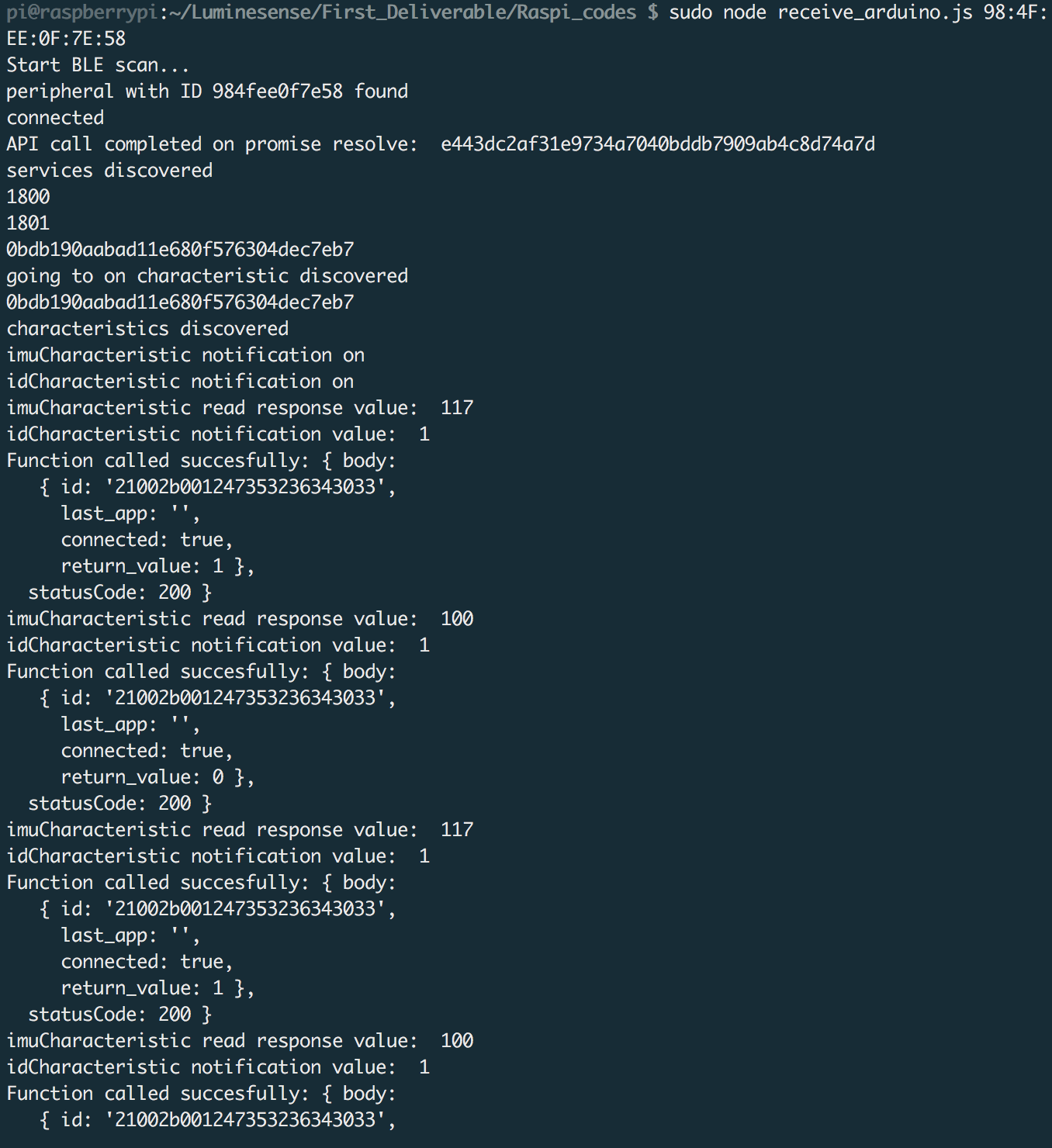
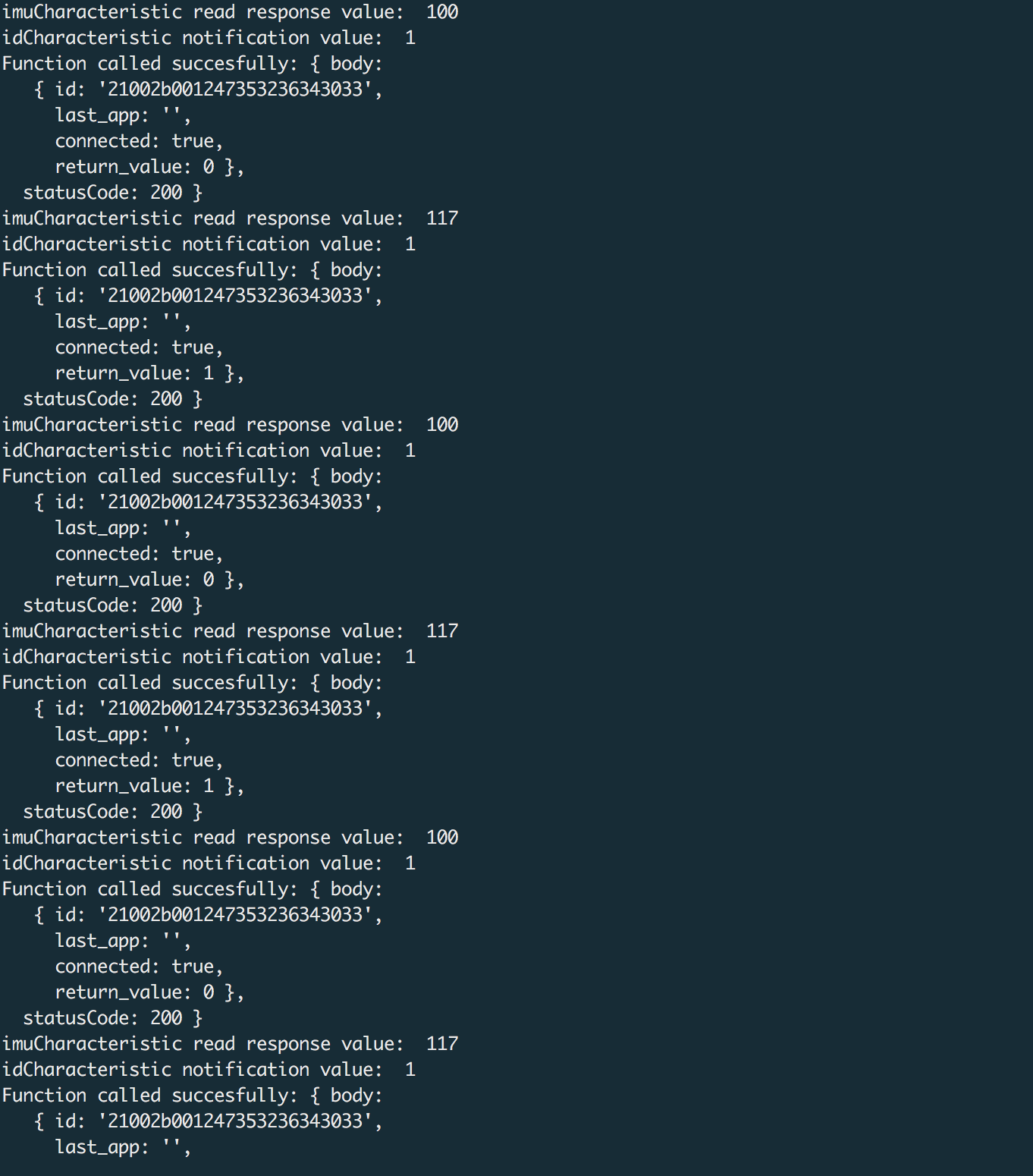
Conditions for successful testing included agreement between data sent by the Arduino and observed Luminaire state. This is illustrated below in Figure 4.



*Figure 4: Arduino data sent compared with observed Luminaire state*

5.2 **Raspberry Pi**

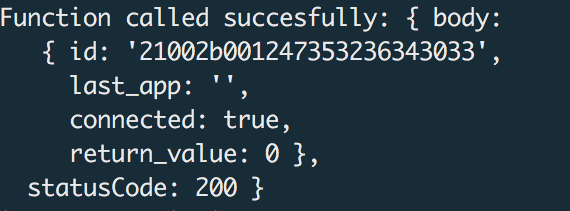
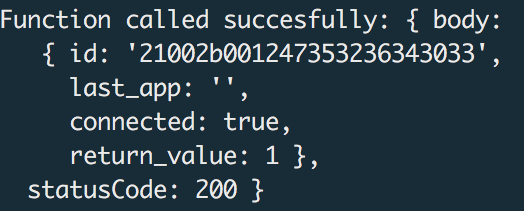
Conditions for successful testing included agreement amongst the data received by the Raspberry Pi, and the Particle API JavaScript call output printed on the Terminal window. This is illustrated below in Figure 5.

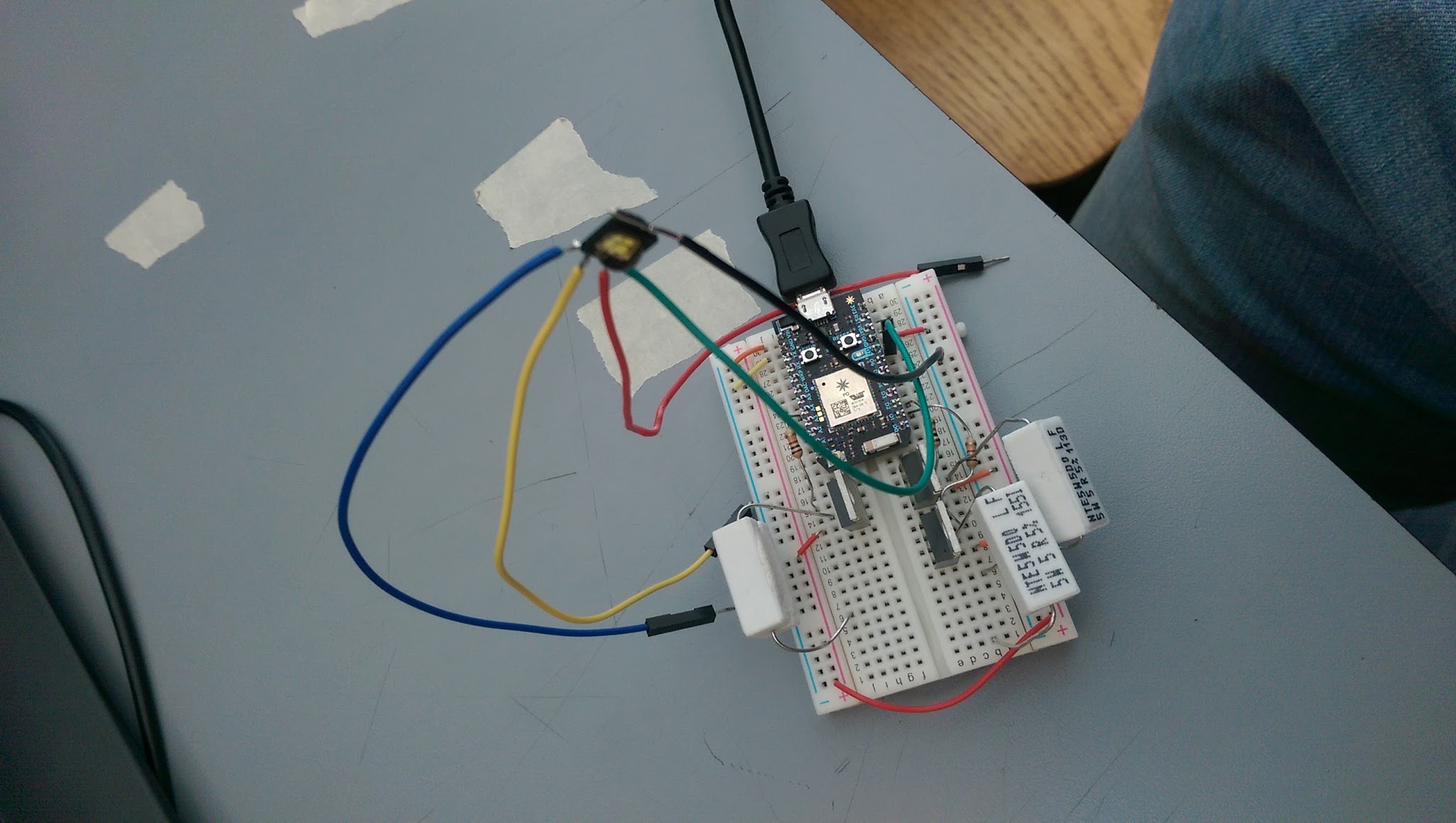
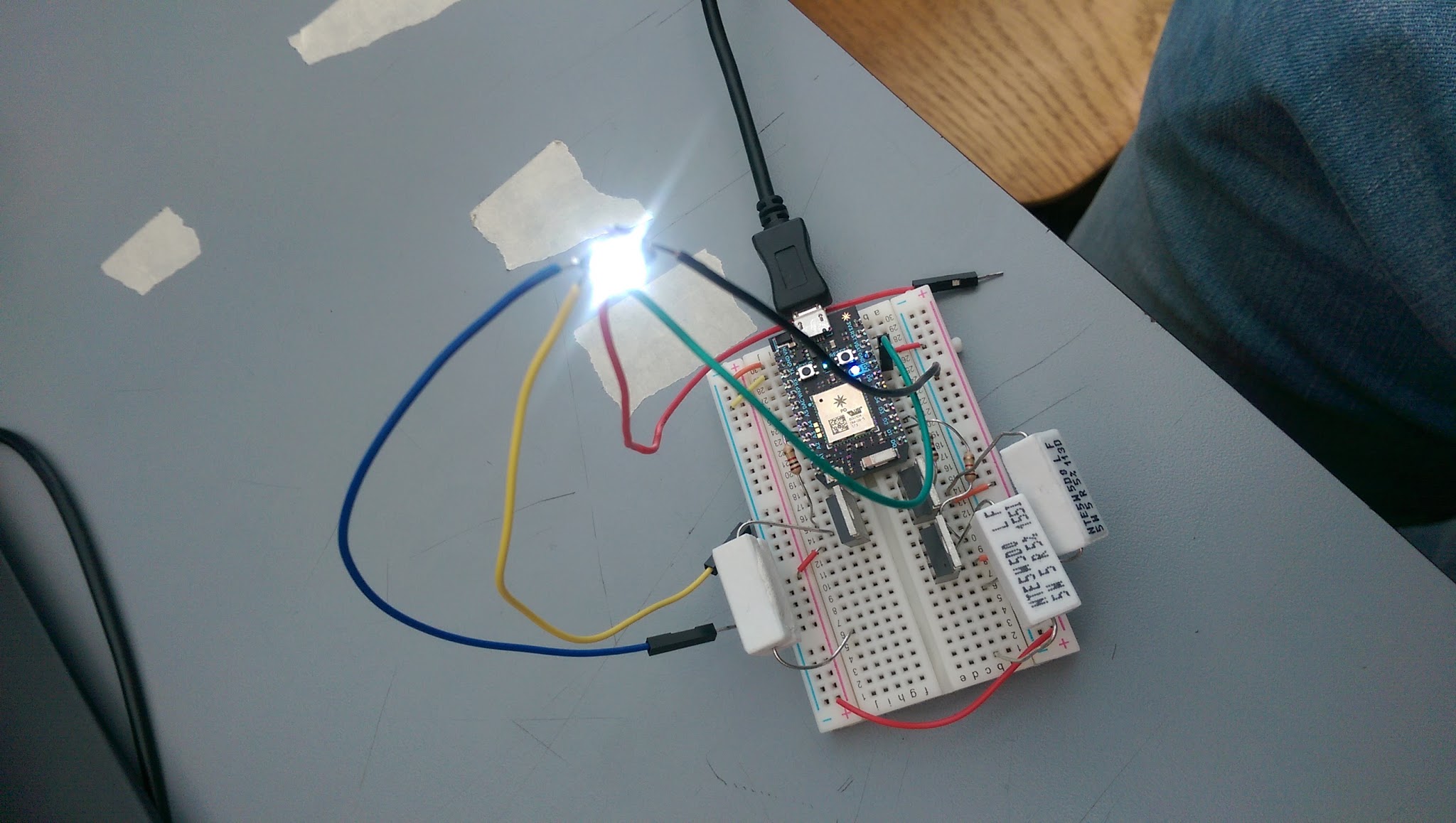


*Figure 5: Raspberry Pi data received and the CLI output of the function call to the Photon.*

5.3 **Particle Photon**

Conditions for successful testing included agreement between the observed state of the luminaire and the Particle API JavaScript call outputs printed on the Terminal window. This is illustrated below in Figure 6.

**ON** **OFF**



*Figure 6: Luminaire state compared with command outputs*

5.4 **Coda and Future Work**

The data presented prior to and on the day of testing implies that the implementation of the test design is successful. Future, more complex iterations of the design will build upon the architecture of this design. For example, being able to detect and distinguish between multiple luminaires, and implementing more gestures. The integration of subsequent design features will be further discussed in future design reviews.