**Memo**

To: Professor Alan Pisano, Professor Osama Alshaykh, Fulya Ekiz Kanik

From: Caroline Jones, Michael Haley, Sneha Pradhan, Tanatsigwa Hungwe, Zehua Zhao

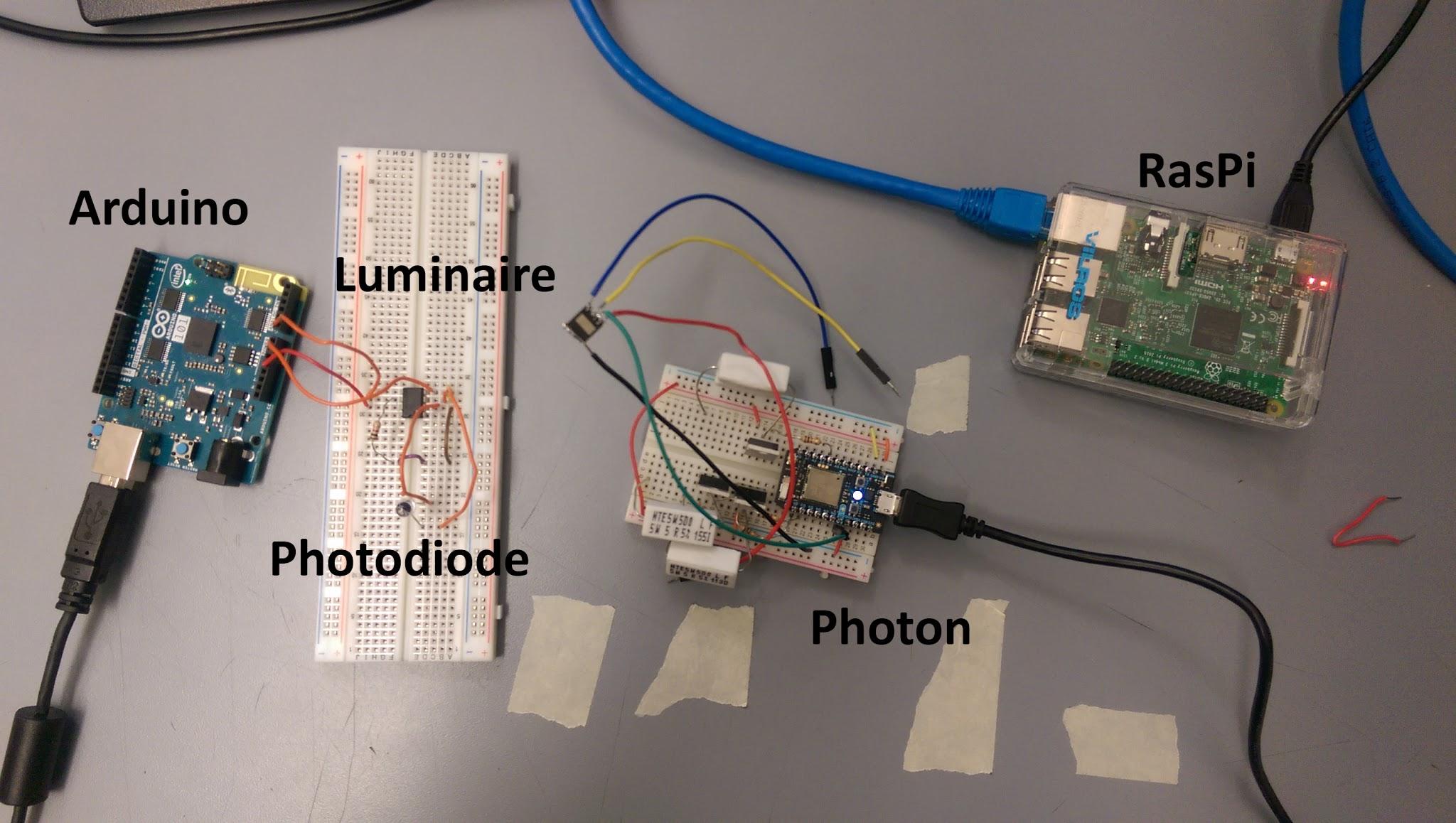
Team: 1

Date: 11/18/16

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

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





1. **Luminaire to Arduino Light Readings**

**1.1 Description and Goal:**

The “Luminaire” is a device provided by the client. It is a powerful light emitting diode which emits a light code that is uniquely identifiable by pulse width modulation. The luminaire is driven by a particle photon, a Wi-Fi based microcontroller. In order to interact with the luminaire prototype, the Luminesense system must be able to sample and decode these light codes. For this purpose, a photodiode circuit was attached to the Arduino 101, a microcontroller with inbuilt Bluetooth Low Energy (Bluetooth 4.0), a six-axis accelerometer, and a gyroscope. The following arrangement models the basic functionality of the wearable device: The photodiode receives light analog light signals, and subsequently, Arduino uses these light intensity readings to obtain the ID of the luminaire.

**1.2 Procedure:**

Utilizing the Particle Photon Web IDE, flash the *togglelights.ino* code onto the Photon. This code initialises the luminaire (connected to D7 port of Photon) to the “off” state. Subsequently, upload to and run *test1\_receive.ino* on the Arduino Uno 101. This code that reads analog input from the photodiode connected to the Arduino’s A0 port. The readings from the photodiode are then sent via Bluetooth 4.0 to the Raspberry Pi. This details regarding the Bluetooth data transmission will be explained further in section 3.0.

**1.3 Verifiable Result:**

A main requirement of the client is for the Luminesense system to seamlessly integrate with their luminaire technology. Showcasing detection and transmission of light intensity data of these luminaires proves that integrating finalized luminaire products into the Luminesense system is feasible. Furthermore, this foundation in order advances the development of more complex system features such as the identification of multiple lights sources.

**2.0 Arduino Gestures**

**2.1 Description and Goal:**

A key feature of the Luminesense product is the ability to correctly recognize gestures the user performs to control the luminaires accordingly. This is the main reason why the Arduino 101 was selected for the wearable: it features a 6-axis accelerometer and gyroscope, designed specifically for wearables. Observing and analyzing the accelerometer and gyroscope data allows the system to match it to a specific gesture. Future iterations of the system have been envisioned to utilize a machine learning library to accomplish this task. However, for the basis of this test, only two gestures are supported: “on” and “off”. These gestures are distinguished by the orientation of the Arduino - specifically if it its connectors are facing upward or downward. This information is transmitted to the Raspberry Pi for decision making, further explained in section 3.0.

**2.2 Procedure:**

Upload to and run *test1\_receive.ino* on the Arduino Uno 101 board via the Arduino IDE. The Arduino prints the gesture data to the Serial Monitor using Serial.Print() and simultaneously sends the specific command to the Raspberry Pi. Each gesture is transmitted as a character - where ‘u’ signifies “on” and the character ‘d’ signifies “off.” This simplifies data transmissions (which occur on a byte-by-byte basis) of messages over bluetooth low energy, which is elaborated further in sections 3.0 and 4.0. Upon rotating the Arduino back and forth, different gestures are observed, recognized and outputted by the Arduino. The Arduino should also only send a gesture in the case of a change in gesture state, preventing redundant messages from being transmitted (such as performing an “on” command to an already illuminated luminaire).

**2.3 Verifiable Result:**

The gesture codes printed to the console provide evidence of the system’s ability to successfully recognize gestures performed by the user after interpreting accelerometer and gyroscope data. The importance of gesture state change is also observed in the printing of the commands, as this saves battery life by reducing the number of necessary transmissions over bluetooth. We hope to build upon this foundation by building a machine learning library for the gestures which the user can train to support custom gestures.

**3.0 Arduino to Raspberry Pi Connection**

**3.1 Description and Goal:**

The wearable sends the gesture (‘u’ or ‘d’ held in *orientationString* variable) and light id (held in the *light\_id* variable) data to the Raspberry Pi. This communication link is key because the Raspberry Pi will be responsible for all decision making i.e. aggregating all the contextual data from the wearable. Moreover, the Raspberry Pi is the link between the wearable and the Particle Photon. Communication between the Arduino and the Pi is accomplished through Bluetooth 4.0 or Bluetooth Low Energy (BLE), supported by both the Raspberry Pi 3 and the Arduino 101. BLE functions via peripheral device and one or multiple central devices. The peripheral devices updates “characteristics” which contain data. The central device can read these characteristics and this is how the communication is done between the two devices. In this case the peripheral device is the Arduino, and the central device is the Raspberry Pi. The Arduino has two characteristics, namely *imuCharacteristics* and *idCharacteristics*. The *imuCharacteristics* stores the gesture commands and the *idCharacteristics* stores the light id. The Raspberry Pi connects to the Arduino by specifying the Arduino’s MAC address as well as the universally unique identifier (UUID) of the characteristics.

**3.2 Procedure:**

Boot up the Raspberry Pi and ssh into it via the Terminal program on a local machine. Navigate the Luminesense/First\_Deliverable/Raspi\_Codes directory and run *receive\_arduino.js* via node following with the MAC address of the Arduino 101 board (98:4F:EE:0F:7E:58). The command would be ‘sudo node receive\_arduino.js 98:4F:EE:0F:7E:58’. The first thing the JavaScript code does is scan for the device with the given MAC address and attempt to connect to it. The Arduino includes a service for each central device that enables the given central device to detect the respective characteristics. The Raspberry Pi is able to identify its characteristics with the help of UUIDs. Each characteristic has an UUID. On the Arduino’s end, make sure that the *test1\_receive.ino* file is running as it requires an active Arduino. The two will connect, the Arduino will update the bluetooth low energy characteristics as it receives input. Subsequently, the Raspberry Pi monitors for changes.

**3.3 Verifiable Result:**

Results can be viewed by comparing the data observed on both ends of the communication link. Viewing the stream of data being transmitted on the console of the Arduino from the Arduino IDE. Additionally, the Raspberry Pi data can also be observed through the command line interface. The JavaScript contains calls to console that help confirm results. The data stream from the Arduino match the data stream that the Raspberry Pi prints to the command line. This verifies the integrity of the bluetooth connection.

**4.0 Raspberry Pi to Photon Connection**

**4.1 Description and Goal:**

The final link of the deliverable is to check the functionality of luminaire control and Raspberry Pi to Photon communication. To validate the feasibility of the loop communication it is essential to demonstrate data transmission between Raspberry Pi and the Photon. This connection is via Wi-Fi using the Particle API JS. The Photon and the Pi both support Wi-Fi communication. The Raspberry Pi authenticates and connects to Photon using an email and password login. The Photon is uniquely identified by a device ID and an access token; these attributes are also used in the authentication process. The Raspberry Pi passes a command as a string argument to a Particle API JS function call to the Photon. The Photon currently holds the code *togglelights.ino* that receives the command and switches the luminaire on or off depending on the command.

**4.2 Procedure:**

During this, make sure that the *receive\_arduino.js* from part 3.0 is still running in the Raspberry Pi. Power the Particle Photon and open the IDE and flash the code *togglelights.ino* to the Photon. The communication from the JS code in the Raspberry Pi happens over the Particle Cloud API. The *togglelights.ino* code enables the cloud connection from the Raspberry Pi and also contains the *togglelights* function that parses the input from the Raspberry Pi. The input command is received as a string in the form “int,char” - where “int” is the light ID, and char is the gesture command. After parsing the data, the *togglelights* function sets the D7 pin (pin the luminaire is connected to) to HIGH or LOW depending on the gesture command.

**4.3 Verifiable Result:**

This step both concludes the communication loop and confirms the correct transmission of data from one end of the system to another. The light is identified, the gesture is recognized via Arduino. Both data sets are transmitted to the Raspberry Pi. The Raspberry Pi parses these data sets into a command, and passes them to the Particle Photon. The Photon consequently changes the state of the luminaire. The luminaire turns off when the connector of the Arduino is face downwards and it turns on when the connectors of the Arduino face upwards.