CSC 453

Introduction to Internet of Things (IoT) Systems

April 26, 2019

Jason Benckert, Cameron Finley, Stanton Parham, WeiRui Wang

Overall Percentage of Contribution

Cameron Finley - 25%
Jason Benckert - 25%
Stanton Parham - 25%
Weirui Wang - 25%

Subtasks from Task Plan

Task	**Component Weightage**	**Cameron Finley**	**Jason Benckert**	**Stanton Parham**	**Weirui Wang**
Design how the user should interact with the system	0.1	25%	25%	25%	25%
Write the code that runs on RPI which are attached to the sensors	0.1	10%	80%	10%	
Set up an IBM Cloud project for the system	0.1	50%		50%	
Create the UI for the end device that the user will be using	0.1	70%		30%	
Finish RPI code and breadboard	0.1	10%	80%	10%	
Complete UI (key binds, buttons, audio input)	0.1			40%	60%

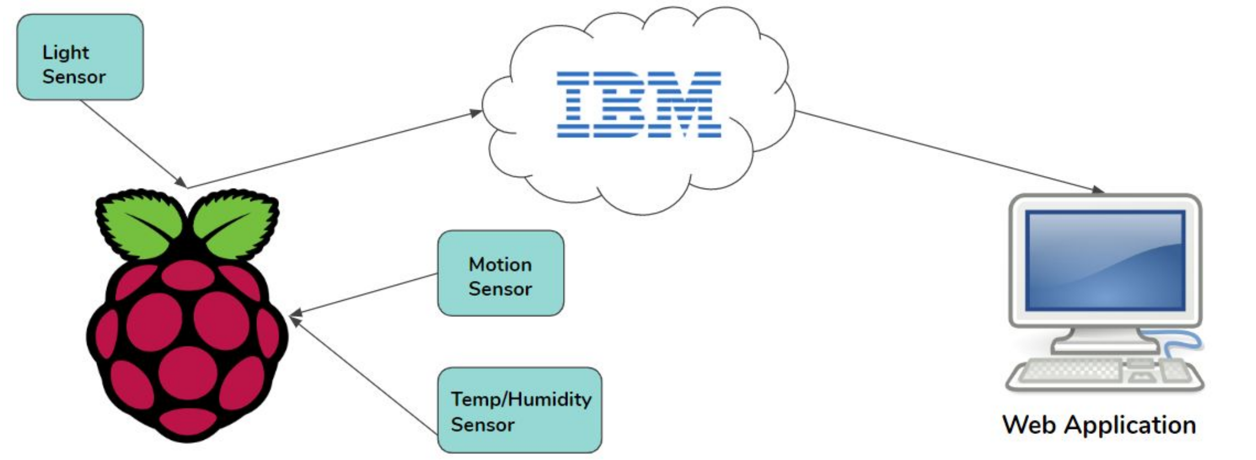
Testing and debugging the entire IoT system	0.2	25%	25%	25%	25%
Presentation	0.1	25%	25%	25%	25%
Write Final Report	0.1		10%		90%

Introduction

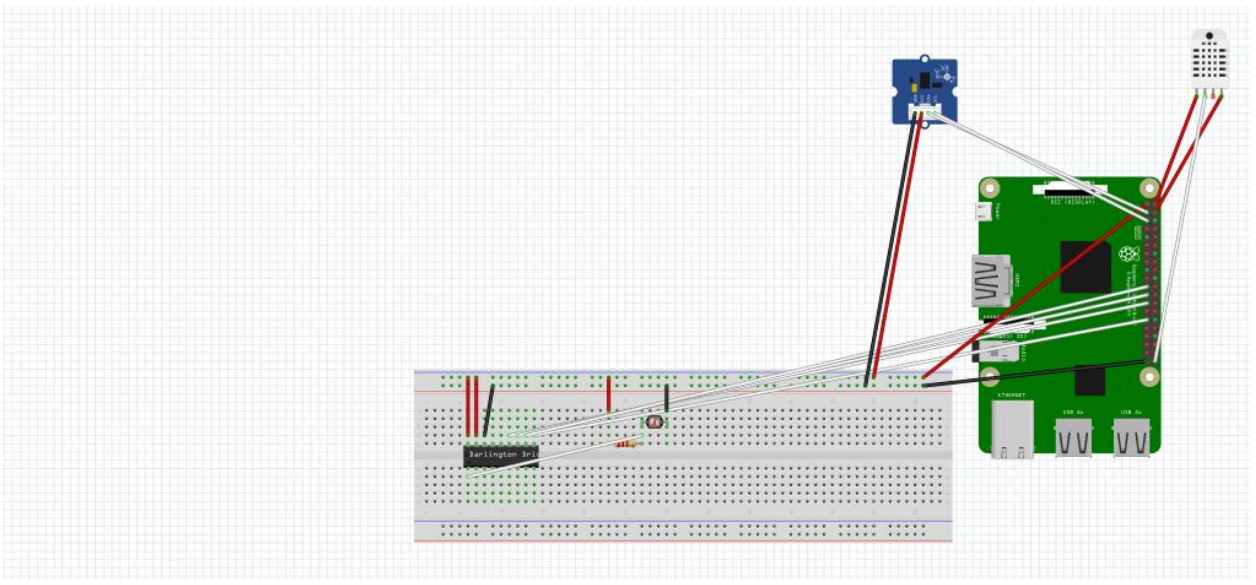
The objective of our project is to “auralize” and “musicalize” IoT data in an effort to help the visually impaired “visualize” the data. We have created sounds based on data collected from sensors in a particular environment that essentially auralizes that environment and its characteristics. This technology could be used by anyone for entertainment or utility purposes although it is mainly directed towards the visually impaired. In the case that a visually impaired person cannot easily navigate and respond to their environment, our product could potentially help them. This is not necessarily a new idea since things like this have existed for ages (e.g. crosswalk alarms), but the way in which we have designed and implemented it certainly is.

Design

For the project, our system uses an MQTT broker for data transfer between the client’s interface and sensors connected to a Raspberry Pi. The user can control the web application’s audio by toggling the sounds produced by different sensors on and off. This allows the user to hear only the sounds that can best help them.



Above you can see a diagram of our system. The light, motion, temperature, and humidity sensors are connected to a Raspberry Pi which is connected to the IoT service in IBM Cloud. The web application is subscribed to the topics for these values on the IBM Cloud and receives them at a regular interval.



Above is the wiring diagram for the sensors to our Raspberry Pi. Wiring the individual sensors was relatively simple even though the overall diagram looks a bit cluttered due to all of the sensors being present. It should be noted that due to limitations of fritzing the sensor brands aren't the same as the real thing by have the same input and output. To clarify, the motion sensor is a MPU6050 and the humidity/temperature sensor is a DHT 11.

Implementation

For the implementation of the project, we used Python on the Raspberry Pi to acquire the sensor data from each sensor while using the IBM IoTf library (based on the Paho MQTT library) to connect and send the sensor values to the broker on IBM's Cloud service. For the web application, we used basic HTML and JavaScript. We adapted the pure Paho MQTT JavaScript library (uses web sockets for MQTT) to work with the IBM Cloud IoT platform. We also use Tone.js (a music/sound composition library which uses the Web Audio API) to produce all sounds.

RaspBerryPiSensorInput.py is the only file that is needed to run on the Raspberry Pi. As mentioned before, the program is responsible for acquiring data from the sensors and sending them to the MQTT broker.

The web application consists of index.html, connection.js, javascript.js, and a CSS file. index.html provides the structure of the user interface with various checkboxes and buttons. connection.js sets up the MQTT client with configuration values for connecting to IBM Cloud. javascript.js contains all of the code that sets up our synthesizers and interface interactions. It also contains the logic for receiving new messages from the MQTT broker.

Results and Discussion

From the implemented project, we have observed a few interesting findings. One of the most significant findings from the implementation is the difficulty associated with reflecting specific sensor value changes as audio. A prime example of this was the temperature and humidity values because they change very slowly and the sensor we were using could only guarantee new values every 2 seconds. This caused the sounds associated with those values to appear quite stagnant or at least "lagging behind" when compared to the sounds for the values collected from the motion and light sensors. Our main method of circumventing this problem was to use these values in ways different than the more quickly changing values such as applying effects to existing sounds and producing slowly changing background sounds. We also attempted to evaluate the temperature and humidity values with higher precision than the other values so as to detect even the slightest of changes.

Related Work and References

"Tone.js." Tone.js, tonejs.github.io/.